jQuery Best Practice

1.Script Positioning

Traditionally, <script> tags that are used to load external JavaScript files have appeared in the <head>, along with <link> tags to load external CSS files and other meta information about the page. The theory was that it's best to keep as many style and behavior dependencies together, loading them first so that the page will come in looking and behaving correctly. For example:

```
<html>
<head>
<title>Script Example</title>
<-\- Example of inefficient script positioning \-->
<script type="text/javascript" src="file1.js"></script>
<script type="text/javascript" src="file2.js"></script>
<script type="text/javascript" src="file3.js"></script>
<link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<p>Hello world\!</p>
</body>\\
</html>
```

Better Approach

Because scripts block downloading of all resource types on the page, it's recommended to place all <script> tags as close to the bottom of the <body> tag as possible so as not to affect the download of the entire page. For example:

```
<html>
<head>
<title>Script Example</title>
<link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<p>Hello world!</p>
<-- Example of recommended script positioning -->
<script type="text/javascript" src="file1.js"></script>
<script type="text/javascript" src="file2.js"></script>
<script type="text/javascript" src="file3.js"></script>
</body>
</html>
```

2. Data Access: Caching/Storing jQuery Objects

A good rule of thumb is to always store out-of-scope values in local variables if they are used more than once within a function.

This function contains three references to document, which is a global object. The search for this variable must go all the way through the scope chain before finally being resolved in the global variable object. You can mitigate the performance impact of repeated global variable access by first storing the reference in a local variable and then using the local variable instead of the global.

```
$(function() {
 $('#elem').hide();
 $('#elem').html('bla');
 $('#elem').otherStuff();
});
```

Better Approach

The updated version of this function first stores a reference to $('#elem') in the local "$localElement" variable. Instead of accessing a global variables three times, that number is cut down to one. Accessing "$localElement" instead of $('#elem') is faster because it's a local variable. Of course,this simplistic function won't show a huge performance improvement, because it's not doing that much, but imagine larger functions with dozens of global variables being accessed repeatedly; that is where the more impressive performance improvements will be found.

The dollar notation on all jQuery-related variables($localElement) helps us easily distinguish jQuery variables from standard JavaScript variables at a glance (e.g. string, integer, etc).

```
1. $(function() {
 var $localElement = $('#elem');//stores in localElement
 $localElement.hide();
 $localElement.html('bla');
 $localElement.otherStuff();
});


OR

$(function() { $("#elem").hide().html('bla') .otherStuff();});
```

3.Using local variables to store references to a collection and its elements during loops

When the length of the collection is accessed on every iteration, it causes the collection to be updated and has a significant performance penalty across all browsers.

```
//slower

function loopCollection() {
 var coll =$('div');
 for (var count = 0; count < coll.length; count++){
   //do Nothing
 }
 }
```

Better Approach

The way to optimize this is to simply cache the length of the collection into a variable and use this variable to compare in the loop's exit condition:

```
function loopCacheLengthCollection()
{
 var coll = $('div'),
 len = coll.length;//store length in variable
 for (var count = 0; count < len; count++){
  //do nothing
 }

}
```

4.Minimizing Repaints and Reflows

Once the browser has downloaded all the components of a page HTML markup, JavaScript, CSS, images it parses through the files and creates two internal data structures:

1.A DOM tree-A representation of the page structure

2.A render tree-A representation of how the DOM nodes will be displayed

When a DOM change affects the geometry of an element (width and height)--such as a change in the thickness of the border or adding more text to a paragraph, resulting in an additional line -the browser needs to recalculate the geometry of the element as well as the geometry and position of other elements that could have been affected by the change. The browser invalidates the part of the render tree that was affected by the change and reconstructs the render tree. This process is known as a reflow. Once the reflow is complete, the browser redraws the affected parts of the screen in a process called repaint.

When you have a number of changes to apply to a DOM element, you can reduce the number of repaints and reflows by following these steps:
1. Take the element off of the document flow.(hide it)
2. Apply multiple changes.
3. Bring the element back to the document.(display again)

```
var table = $( "#myTable" );
var parent = table.parent();

table.detach();//Detach Elements to Work with Them

// ... add lots and lots of rows to table

parent.append( table );
```

5.Selectors

1. Use ID selector whenever possible. It is faster because they are handled using $('#id').
2. When using class selectors, don't use the element type in your selector.

```
var $products = $("div.products"); // SLOW
var $products = $(".products"); // FAST
```

3. Use find for Id->Child nested selectors. The .find() approach is faster because the first selection is handled without going through the selector engine.

```
// BAD, a nested query for Sizzle selector engine
 var $productIds = $("#products div.id"); // SLOW
  // GOOD, #products is already selected by document.getElementById()
so only div.id needs to go through  selector engine
 var $productIds = $("#products").find("div.id"); // FAST
```

4. Give your Selectors a Context.

```
// SLOWER because it has to traverse the whole DOM for .class
 $('.class'); // SLOW
 // FASTER because now it only looks under class-container.
 $('.class', '#class-container'); // FAST
```

5. Be specific on the right-hand side of your selector, and less specific on the left.

```
// Unoptimized
 $("div.data .gonzalez"); // SLOW
 // Optimized
 $(".data td.gonzalez"); // FAST
```

6.Beware Anonymous Functions

Anonymous functions bound everywhere are a pain. They're difficult to debug, maintain, test, or reuse. Instead, use an object literal to organize and name your handlers and callbacks.

The Object Literal

An object literal is perhaps the simplest way to encapsulate related code. It doesn't offer any privacy for properties or methods, but it's useful for eliminating anonymous functions from your code, centralizing configuration options, and easing the path to reuse and refactoring.

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

    $( "#magic" ).click(function( event ) {
        $( "#yayeffects" ).slideUp(function() {
            // ...
        });
    });

    $( "#happiness" ).load( url + " #unicorns", function() {
        // ...
    });

});
```

Better Approach

```
// BETTER
  var PI = {

    onReady: function() {
        $( "#magic" ).click( PI.candyMtn );
        $( "#happiness" ).load( PI.url + " #unicorns", PI.unicornCb );
    },

    candyMtn: function( event ) {
        $( "#yayeffects" ).slideUp( PI.slideCb );
    },

    slideCb: function() { ... },

    unicornCb: function() { ... }

 };

$( document ).ready( PI.onReady );
```

The Module Pattern

The module pattern overcomes some of the limitations of the object literal, offering privacy for variables and functions while exposing a public API if desired.In the example below, we self-execute an anonymous function that returns an object. Inside of the function, we define some variables. Because the variables are defined inside of the function, we don't have access to them outside of the function unless we put them in the return object. This means that no code outside of the function has access to the privateThing variable or to the changePrivateThing function. However, sayPrivateThing does have access to privateThing and changePrivateThing, because both were defined in the same scope as sayPrivateThing. This pattern is powerful because, as you can gather from the variable names, it can give you private variables and functions while exposing a limited API consisting of the returned object's properties and methods.

```
// The module pattern
var feature = (function() {

    // Private variables and functions
    var privateThing = "secret";
    var publicThing = "not secret";

    var changePrivateThing = function() {
        privateThing = "super secret";
    };

    var sayPrivateThing = function() {
        console.log( privateThing );
        changePrivateThing();
    };

    // Public API
    return {
        publicThing: publicThing,
        sayPrivateThing: sayPrivateThing
    };
})();

feature.publicThing; // "not secret"

// Logs "secret" and changes the value of privateThing
feature.sayPrivateThing();
```

7.Do Not Use Inline Javascript

DO NOT use behavioral markup in HTML (JavaScript inlining), these are debugging nightmares. Always bind events with jQuery to be consistent so it's easier to attach and remove events dynamically.

```
<a id="myLink" href="#" onclick="myEventHandler();">my link</a> //BAD
```

Better Approach

```
$("#myLink").on("click", myEventHandler); // GOOD
```

8.Miscellaneous

1. Use only one Document Ready handler per page. It makes it easier to debug and keep track of the behavior flow.
2. DO NOT use *http* requests on *https* sites. Prefer schemaless URLs (leave the protocol *http/https* out of your URL)
3. DO NOT put request parameters in the URL, send them using data object setting.

```
// Less readable...
$.ajax({
    url: "something.php?param1=test1&param2=test2",
    ....
});
// More readable...
$.ajax({
    url: "something.php",
    data: { param1: test1, param2: test2 }
});
```

4. Don’t Act on Absent Elements,check on length property.

```
// BAD: This runs three functions before it realizes there's nothing
in the selection
$("#nosuchthing").slideUp();

// GOOD
var $mySelection = $("#nosuchthing");
if ($mySelection.length) {  //check on length property
    $mySelection.slideUp();
}
```

Tip:There is website to test Javascript code performance- Ex: http://jsperf.com/jqeury-selector-test

References

- "High performance Javascript" book by Nicholas C. Zakas
- https://learn.jquery.com/code-organization/concepts/
- http://www.jameswiseman.com/blog/2010/04/20/jquery-standards-and-best-practice/