17 Essential jQuery Interview Questions *

Explain what the following code will do:

```
$( "div#first, div.first, ol#items > [name$='first']" )
```

View the answer →Hide answer

This code performs a query to retrieve any `<div>` element with the *id* `first`, *plus* all `<div>` elements with the *class* `first`, *plus* all elements which are children of the `<ol id="items">` element and whose `name` attribute ends with the string `"first"`. This is an example of using multiple selectors at once. The function will return a jQuery object containing the results of the query.

The code below is for an application that requires defining a click handler for all buttons on the page, including those buttons that may be added later dynamically.

What is wrong with this code, and how can it be fixed to work properly even with buttons that are added later dynamically?

```
// define the click handler for all buttons
$( "button" ).bind( "click", function() {
    alert( "Button Clicked!" )
});

/* ... some time later ... */

// dynamically add another button to the page
$( "html" ).append( "<button>Click Alert!</button>" );
```

View the answer →Hide answer

The button that is added dynamically after the call to `bind()` will *not* have the click handler attached. This is because the `bind()` method only attaches handlers to elements that exist at the time the call to `bind()` is made.

This problem is solved with functions that use “event bubbling” to match events on both current and future elements. In the past, this was done by replacing `bind()` with `live()`. `live()` was deprecated in jQuery 1.7 though. `delegate()` works similarly to `live()` but also provides control over how far an event must bubble up the DOM.

However, the recommended method is to use `on()`, which can behave like `bind()`, `live()`, or `delegate()` depending on syntax. The following code attaches the handler to all current and future buttons:

```
// define the click handler for all buttons
$( document ).on( "click", "button", function() {
    alert( "Button Clicked!" )
});

/* ... some time later ... */

// dynamically add another button to the page
$( "html" ).append( "<button>Click Alert!</button>" );
```

What’s the deal with the `$` in jQuery? What is it and what does it mean?

Also, how can jQuery be used in conjunction with another JavaScript library that also uses `$` for naming? Bonus credit if you can provide two answers.

View the answer →Hide answer

Since `$` has no special meaning in JavaScript, it is free to be used in object naming. In jQuery, it is simply used as an alias for the `jQuery` object and `jQuery()` function.

However, jQuery has no monopoly on use of `$`, so you may encounter situations where you want to use it in conjunction with another JS library that also uses `$`, which would therefore result in a naming conflict. jQuery provides the `jQuery.noConflict()` method for just this reason. Calling this method makes it necessary to use the underlying name `jQuery` instead in subequent references to jQuery and its functions.

Here’s an example from the jQuery documentation:

```
<script src="other_lib.js"></script>
<script src="jquery.js"></script>
<script>
$.noConflict();
// Code that uses other library's $ can follow here.
</script>
```

Alternatively, you can also use a closure instead of the $.noConflict() method; e.g.:

```
(function ($) {
  // Code in which we know exactly what the meaning of $ is
} (jQuery));
```


Given the following HTML:

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

and the following CSS:

```
div#expander{
  width: 100px;
  height: 100px;
  background-color: blue;
}
```

Write code in jQuery to animate the `#expander` div, expanding it from 100 x 100 pixels to 200 x 200 pixels over the course of three seconds.

View the answer →Hide answer

This could be done in jQuery as follows:

```
$( "#expander" ).animate(
  {
    width: "200px",
    height: "200px",
  },
  3000 );
```

What is method chaining in jQuery? Provide an example.

What advantages does it offer?

View the answer →Hide answer

Method chaining is a feature of jQuery that allows several methods to be executed on a jQuery selection in sequence in a single code statement. For example, the following snippets of code are equivalent:

Without chaining:

```
$( "button#play-movie" ).on( "click", playMovie );
$( "button#play-movie" ).css( "background-color", "orange" );
$( "button#play-movie" ).show();
```

With chaining:

```
$( "button#play-movie" ).on( "click", playMovie )
                        .css( "background-color", "orange" )
                        .show();
```

Notice that with chaining, the button only needs to be selected one time, whereas without chaining, jQuery must search the whole DOM and find the button before each method is applied. Thus, in addition to yielding more concise code, method chaining in jQuery offers a potentially powerful performance advantage.

Note: To be entirely precise, it should be noted that method chaining in jQuery is not the *only* way to avoid repetitively searching the entire DOM. One could also set a variable equal to the initial DOM search results (i.e., in the above example, one could set a variable equal to `$( "button#play-movie"` and then call the `on()`, `css()`, and `show()` methods on that variable). But that said, chaining is still the more concise and efficient option and doesn't require caching the result in a local variable.

Explain what the following code does:

```
$( "div" ).css( "width", "300px" ).add( "p" ).css( "background-color", "blue" );
```

View the answer →Hide answer

This code uses method chaining to accomplish a couple of things. First, it selects all the `<div>` elements and changes their CSS width to 300px. Then, it adds all the `<p>` elements to the current selection, so it can finally change the CSS background color for both the `<div>` and `<p>` elements to blue.

What is the difference between `jQuery.get()` and `jQuery.ajax()`?

View the answer →Hide answer

`jQuery.ajax()` is the all-encompassing Ajax request method provided by jQuery. It allows for the creation of highly-customized Ajax requests, with options for how long to wait for a response, how to handle a failure, whether the request is blocking (synchronous) or non-blocking (asynchronous), what format to request for the response, and many more options.

`jQuery.get()` is a shortcut method that uses `jQuery.ajax()` under the hood, to create an Ajax request that is typical for simple retrieval of information. Other pre-built Ajax requests are provided by jQuery, such as `jQuery.post()`, `jQuery.getScript()`, and `jQuery.getJSON()`.

Which of the two lines of code below is more efficient? Explain your answer.

```
document.getElementById( "logo" );
```

or

```
$( "#logo" );
```

View the answer →Hide answer

The first line of code, which is pure JavaScript without jQuery, is more efficient and faster. Executing the second line of code, which is jQuery, will trigger a call to the JavaScript version.

jQuery is built on top of JavaScript and uses its methods under the hood to make DOM manipulation easier, at the cost of some performance overhead. It is a good idea to remember that jQuery is not always better than plain old JavaScript. Always consider whether using jQuery really provides a useful advantage for your project.

Explain and contrast the usage of `event.preventDefault()` and `event.stopPropagation()`. Provide an example.

View the answer →Hide answer

`event.stopPropagation()` stops an event from bubbling up the event chain, whereas `event.preventDefault()` only precludes the browser's default action on that event from occurring, but the event still propogates up the event chain.

For example, consider the following code snippet:

```
// in this example, 'foo' is a div containing button 'bar'

$("#foo").click(function() {
   // mouse click on div 'foo'
});

$("#bar").click(function(e) {
   // mouse click on button 'bar'
   e.stopPropagation();
});
```

Due to the call to `stopPropagation()` in the button's click handler, the event never propogates to the div so its click handler never fires. It effectively stops parent elements from knowing about an event on their children.

In contrast, if you replaced the above call to `stopPropagation()` with a call to `preventDefault()`, only the browser's default action would be precluded, but the div's click handler would still fire.

(Note: Although the `stopPropagation()` and `preventDefault()` methods are mostly used in jQuery event handling implementations, they apply to plain JavaScript as well.)

What selector would I use to query for all elements with an ID that *ends* with a particular string? Also, how would I modify the selector to retrieve only `<div>` elements whose IDs end with that same string? Provide an example.

View the answer →Hide answer

Let's say you want to retrieve all elements whose IDs end with "txtTitle". This could be done using the following selector:

```
$("[id$='txtTitle']")
```

To retrieve only `<div>` elements whose IDs end with "txtTitle", the selector would be:

```
$("div[id$='txtTitle']")
```

What is accomplished by returning `false` from (a) a jQuery event handler, (b) a regular JavaScript onclick event handler for an anchor tag, and (c) a regular JavaScript onclick event handler for a non-anchor tag (e.g., a div, button, etc.)?

View the answer →Hide answer

(a) Returning false from a jQuery event handler is effectively the same as calling *both* `preventDefault()` *and* `stopPropagation()` on the passed jQuery event object.

(b) Returning false from a regular JavaScript onclick event handler for an anchor tag prevents the browser from navigating to the link address *and* stops the event from propagating through the DOM.

(c) Returning false from a regular JavaScript onclick event handler for a non-anchor tag (e.g., a div, button, etc.) has absolutely no effect.

jQuery provides a useful `.clone()` method to create a deep copy of matching elements.

Answer the following questions:

1. What is meant by a “deep copy”?
2. What is normally *not* included in the cloned copy? How can some of this behavior be controlled?
3. What is a potential “gotcha” when using the `.clone()` method? (HINT: What is an element attribute that you would generally *not* want to clone?)

View the answer →Hide answer

1. What is meant by a “deep copy”?

The `.clone()` method performs a deep copy of the set of matched elements, meaning that it copies the matched elements *as well as their descendant elements and text nodes*.

2. What is normally *not* included in the cloned copy? How can some of this behavior be controlled?

Normally:

- Objects and arrays within element data are not copied and will continue to be shared between the cloned element and the original element. To deep copy all data, you must copy each one “manually”.
- Any event handlers bound to the original element are not copied to the clone.

Setting the optional `withDataAndEvents` parameter to `true` makes copies of all of the event handlers as well, bound to the new copy of the element.

As of jQuery 1.4, all element data (attached by the `.data()` method) is also copied to the new copy.

As of jQuery 1.5, `withDataAndEvents` can be optionally enhanced with `deepWithDataAndEvents` to copy the events and data for all children of the cloned element.

3. What is a potential “gotcha” when using the `clone()` method? (HINT: What is an element attribute that you would generally *not* want to clone?)

Using `.clone()` has the potentially problematic side-effect of producing elements with duplicate `id` attributes, which are supposed to be unique. Therefore, when cloning an element with an `id` attribute, it’s important to remember to modify the `id` of the clone, before inserting it into the DOM.

Explain the `.promise()` method in jQuery, including how and why it would be used.

Consider the code snippet below. If there are 5 `<div>` elements on the page, what will be the difference between the start and end times displayed?

```
function getMinsSecs() {
  var dt = new Date();
  return dt.getMinutes()+":"+dt.getSeconds();
}

$( "input" ).on( "click", function() {
  $( "p" ).append( "Start time: " + getMinsSecs() + "<br />" );
  $( "div" ).each(function( i ) {
    $( this ).fadeOut( 1000 * ( i * 2 ) );
  });
  $( "div" ).promise().done(function() {
    $( "p" ).append( "End time: " + getMinsSecs() + "<br />" );
  });
});
```

View the answer →Hide answer

The `.promise()` method returns a dynamically generated Promise that is resolved once all actions of a certain type bound to the collection, queued or not, have ended.

It takes two optional arguments:

- `type` - By default, type is “fx”, which means that the returned Promise is resolved when all animations of the selected elements have completed.
- `target` - If a target object is specified, `.promise()` will attach to it and then return this object rather than create a new one.

In the code sample provided, the difference between the start and end times displayed will be 10 seconds. This is because `.promise()` will wait for all `<div>` animations (in this case, all `fadeOut()` calls) to complete, the last of which will complete 10 seconds (i.e., 5 * 2 seconds) after the animation starts.

What is the proper way in jQuery to remove an element from the DOM before its Promise is resolved?

View the answer →Hide answer

A returned Promise in jQuery is linked to a Deferred object stored on the `.data()` for an element. Since the `.remove()` method removes the element’s data as well as the element itself, it will prevent any of the element’s unresolved Promises from resolving.

Therefore, if it is necessary to remove an element from the DOM before its Promise is resolved, use `.detach()` instead and follow with `.removeData()` after resolution.

Explain the difference between the `.detach()` and `.remove()` methods in jQuery.

View the answer →Hide answer

The `.detach()` and `.remove()` methods are the same, except that `.detach()` retains all jQuery data associated with the removed elements and `.remove()` does not. `.detach()` is therefore useful when removed elements may need to be reinserted into the DOM at a later time.

What's the difference between `document.ready()` and `window.onload()`?

View the answer →Hide answer

The `document.ready()` event occurs when all HTML documents have been loaded, but `window.onload()` occurs when all content (including images) has been loaded. So generally the `document.ready()` event fires first.

What's the difference between `prop()` and `attr()`?

View the answer →Hide answer

Both `prop()` and `attr()` are used to get or set the value of the specified property of an element attribute, but `attr()` returns the default value of a property whereas `prop()` returns the current value.

* There is more to interviewing than tricky technical questions, so these are intended merely as a guide. Not every "A" candidate worth hiring will be able to answer them all, nor does answering them all guarantee an "A" candidate. At the end of the day, hiring remains an art, a science — and a lot of work.
Submit an interview question
Submitted questions and answers are subject to review and editing, and may or may not be selected for posting, at the sole discretion of Toptal, LLC.

Name
Email
Enter your question here
Enter your answer here
All fields are required
I agree with the Terms and Conditions of Toptal, LLC's Privacy Policy
Submit a Question

Thanks for submitting your question.
Our editorial staff will review it shortly. Please note that submitted questions and answers are subject to review and editing, and may or may not be selected for posting, at the sole discretion of Toptal, LLC.
Looking for jQuery experts? Check out Toptal's jQuery developers.
View full profile »
Luís Martinho
Portugal
As an entrepreneur, Luís understands the importance of proactivity and results, and has learned the meaning of responsibility and accountability. He is more of a
than a specialist, though he loves the detail and deep understanding that comes from intense focus and work on development projects.

jQueryPythonDjango
Hire Luis
View full profile »
Mark Wong-VanHaren
United States
Mark is an entrepreneur, engineer, CTO, and artisan with decades of startup experience, including co-founding Excite.com. He makes complex problems simple with expressive, maintainable code. He believes in building small, well-tested, functional pieces, loosely joined by a well-documented contract.
jQueryRubyJavaScriptPythonClojure+8 more
Hire Mark
View full profile »
Stefan Progovac
United States
Stefan is a highly skilled iOS developer with a master's degree in physics. He loves both technological and artistic sides of software engineering. He has had the pleasure to work on some popular enterprise-level apps used by millions of people for companies like Target, Best Buy, and Coca-Cola. He believes that app development is truly an art.
jQuerySwiftObjective-C+10 more
Hire Stefan
Toptal connects the top 3% of freelance talent all over the world.

Join the Toptal community.

Hire a developer
or
Apply as a developer

Highest In-Demand Talent

- iOS Developers
- Front-End Developers
- UX Designers
- UI Designers
- Financial Modeling Consultants
- Interim CFOs
- Digital Project Managers

About

- Top 3%
- Clients
- Freelance Developers
- Freelance Designers
- Freelance Finance Experts
- Freelance Project Managers
- Freelance Product Managers
- About Us

Contact

- Contact Us
- Press Center
- Careers
- FAQ

Social

Hire the top 3% of freelance talent™