# Jquery

Jquery lesson based off of:

<https://www.youtube.com/playlist?list=PLWKjhJtqVAbkyK9woUZUtunToLtNGoQHB>

Concatenate classes/ids to logical AND select them. They can be specified in any order:

**$('div#a.c.b')** matches to a div with an id of **a** and has the classes **b** and **c**

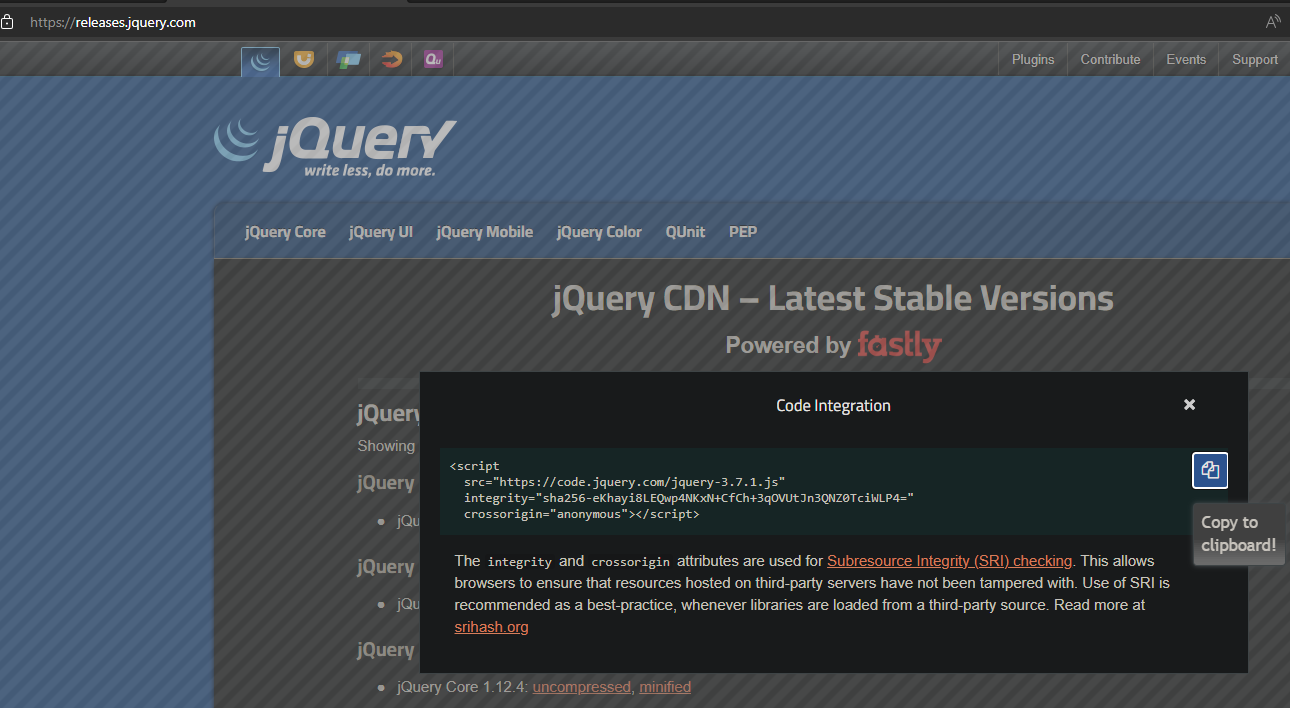
<div id ="a" class="b c"></div>

https://stackoverflow.com/a/1041352

**var** list = $( "div, p, span" ) // Logical OR, selects a div OR p OR span element

<https://api.jquery.com/multiple-selector/>

Jquery is a javascript library meant to make it easier to do more while writing less code. It simplifies AJAX calls and DOM manipulation. You have the option to either download it and host it yourself or add its CDN in a script tag to the HTML document.

<script src="https://code.jquery.com/jquery-3.7.1.js" integrity="sha256-eKhayi8LEQwp4NKxN+CfCh+3qOVUtJn3QNZ0TciWLP4=" crossorigin="anonymous"></script>

When using jQuery, the syntax is as follows:

**$(selector).action()**

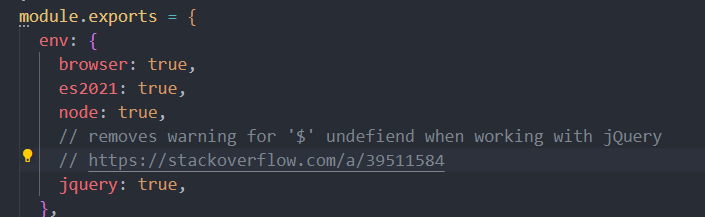
### Eslint warnings

You may notice in your js files that $ is throwing a warning. In the eslint.js rules, add jQuery: true to the env

https://stackoverflow.com/a/39511584

Random Trouble shooting

If js isn't working, check that you used script src="" and not source src=""



Sample code

// Syntax for jQuery

// $(selector).action()

// use a document.ready event to wrap your code

// this prevents js from executing before the DOM has finished loading

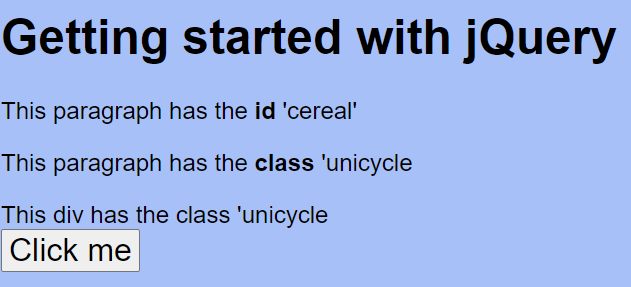
**$**(document).**ready**(function () {

**$**("button").**click**(function () {

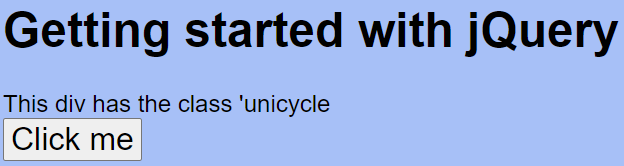
**$**("p").**hide**();

  });

});



Clicking the button results in the paragraph elements being hidden.



<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

  <!-- linking to jQuery library -->

  <script src="https://code.jquery.com/jquery-3.7.1.js" integrity="sha256-eKhayi8LEQwp4NKxN+CfCh+3qOVUtJn3QNZ0TciWLP4="

    crossorigin="anonymous"></script>

    <link rel="stylesheet" href="style.css">

</head>

<body>

  <h1>Getting started with jQuery</h1>

  <p id="cereal">This paragraph has the <strong>id</strong> 'cereal'</p>

  <p class="unicycle">This paragraph has the <strong>class</strong> 'unicycle</p>

  <div class="unicycle">This div has the class 'unicycle</div>

  <button>Click me</button>

  <!-- Adding js file -->

  <script src="index.js"></script>

</body>

</html>

In the javascript file, we first wait for the page to load with document.ready(). All of the code within the ready() method will only run once the DOM has finished loading.

First we select a "button" element, add an event listener for a click action and then call the .hide() method to hide all elements matching the $("p").

If you would like to select an id instead of targeting elements, use $("#<idGoesHere>"). Adding the # lets Jquery know that we want to target an ID instead, this is similar to document.getElementById(""), but with much less to write. It matches the selection syntax needed to select an id when using document.querySelector() (returns node list).

Selecting an id

// on button click hide element with id="cereal"

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

**$**("button").**click**(function () {

**$**("#cereal").**hide**();

  });

});

Selecting a class requires you to prefix the class with a .

// select a .class

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

**$**("button").**click**(function () {

**$**(".unicycle").**hide**();

  });

});

## Document.ready() is deprecated, use this instead

In jquery, most DOM events have an equivalent jQuery method

Examples:  
dblclick, mouseenter, mouseleave, mousedown, mouseup, hover

The hover method is a little special, uses two coma separated functions:

  // hover is a combination of two functions, mouseenter and mouseleave

**$**("p").**hover**(

    function () {

      console.**log**("You entered.");

    },

    function () {

      console.**log**("You left");

    }

  );

});

## Form events

There are also some form events for jquery, focus, blur, change

// document.ready() is deprecated, wrap code in a function prefixed with $ instead

**$**(function () {

  // If a paragraph element is clicked, print message to console

  // $("p").click(function () {

  //   console.log("You clicked a paragraph");

  // });

  // using the on method, any number of events can be added here

**$**("p").**on**("click mouseEnter mouseleave", function () {

    console.**log**("You interacted with a paragraph");

  });

  // more advanced on usage, passing an obj

**$**("p").**on**({

**click**: function () {

      console.**log**("clicked");

    },

**mouseover**: function () {

      console.**log**("hovered");

    },

  });

  // remove an event listener with .off

**$**("p").**off**("click");

  // limit with .one()

**$**("p").**one**({

**dblclick**: function () {

      console.**log**("You double clicked! This message will only display once");

    },

  });

  // in jquery, most DOM events have a jQuery method

  // examples:

  // dblclick, mouseenter, mouseleave, mousedown, mouseup, hover

  // $("p").mouseleave(function () {

  //   console.log("Your mouse left the paragraph");

  // });

  // $("p").mouseenter(function () {

  //   console.log("Your mouse entered the paragraph");

  // });

  // hover is a combination of two functions, mouseenter and mouseleave

**$**("p").**hover**(

    function () {

      console.**log**("You entered.");

    },

    function () {

      console.**log**("You left");

    }

  );

  // form events: focus, blur, change

  // focus listens for an form element to be in focus

**$**("input").**focus**(function () {

    console.**log**("You selected the input field");

  });

  // blur listens for user to leave field

**$**("input").**blur**(function () {

    console.**log**("You left the input field");

  });

  // listens for "enter" or for user to stop focusing on field

  // state must be modified from where it was when the user initially interacted with it

**$**("input").**change**(function () {

    console.**log**("you entered something into the input field");

  });

  // these events are shorthand for the .on() method

});

## Effects

These are like transition effects/animations. It is very simple to toggle an element's visibility with the .hide() and .show() methods. They can take a string value of slow, normal or fast or a numeric value in ms as either a string or Number data type.

\*.hide() removes the element from the normal flow of a document so it could become an accessibility issue in cases where you needed an element to be accessible to screen readers.

Show and Hide use a combination of slide and fade to make an element invisible gradually.

### Slide

Slide can either be slideUp("") or slideDown("") and takes the same transition values as show/hide. They do not incorporate the fade effects of hide/show.

SlideUp creates a screen whip effect from the bottom up while slide down creates a while from the top down. Slide up hides an element while slide down hides the element.

### fadeOut/fadeIn

Like the previous methods, these can be used to effect an element's visibility. Elements being faded still have their full height until they are fully transparent. This may cause an odd popping in/out of existence effect as space is made/removed for the element.

### Toggle

The toggle method will hide/reveal an element based on its current state. It seems to use the hide/show methods. Like the previous examples the transition speed can be specified.

There are variations of the toggle method like slideToggle/fadeToggle, which will then take the previous state into account.

## Callbacks

Callbacks can be used to make code execute after the previous method has completed. The syntax is fairly simple:

**$**(".p4").**fadeIn**(2000, function () {

**$**(this).**addClass**("cyan");

  })

You can pass a jQuery selector in a callback function and add the stop() method to it to stop an animation. in this example, we are listening for a click event on an img tag. If one occurs, we stop any animations being done to the h1 element in this example. This can also prevent classes being added to an element i.e. if we called stop() on the .p4 element before its animation completed, it would never get the class of "cyan".

  // stop an animation using a callback

  // the element we are stopping the animation on does not need to be related to the element with the click event

**$**("img").**on**("click", function () {

**$**("h1").**stop**();

  });

### delay()

The delay method adds a delay between animations.

**$**(".p3").**hide**(500).**delay**(1000).**show**(300);

## Animate

The animate method of jquery performs a custom animation of a set of CSS properties. This method changes the state of one element to another with css styles. The CSS value is changed gradually to create that effect. Only numeric values can be animated. String values cannot be animated.

The animate method takes CSS properties, followed by three optional arguments: duration, easing and complete.

duration - duration in ms

easing -ease in/out

complete - callback function

There's an alternate syntax where the arguments passed to animate are properties and options.

options is a js object of various k=v pairs. Options are typically calling various callback functions at different points in the animation. A little more control than the previous syntax.

**$**("#go").**click**(function () {

**$**("#block").**animate**({

    opacity: 0.5,

    marginLeft: "+=50",

    height: "400px",

  }, 3000, "linear", function () {

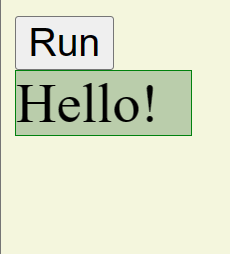
**$**(this).**after**("<div>Animation complete.</div>");

  });

})

In this javascript, we are targeting the element with an id of go. After it is clicked, we select the #block element and add an animation where the opacity is set to 50% and we continuously add 50px to the left margin. We set the height to 400px; The duration of the animation is 3s, it is a linear animation and then we call a function which selects the #block element and adds a div with the text animation complete after the element in the DOM. (<div id="block">...</div><div>Animation complete</div>)

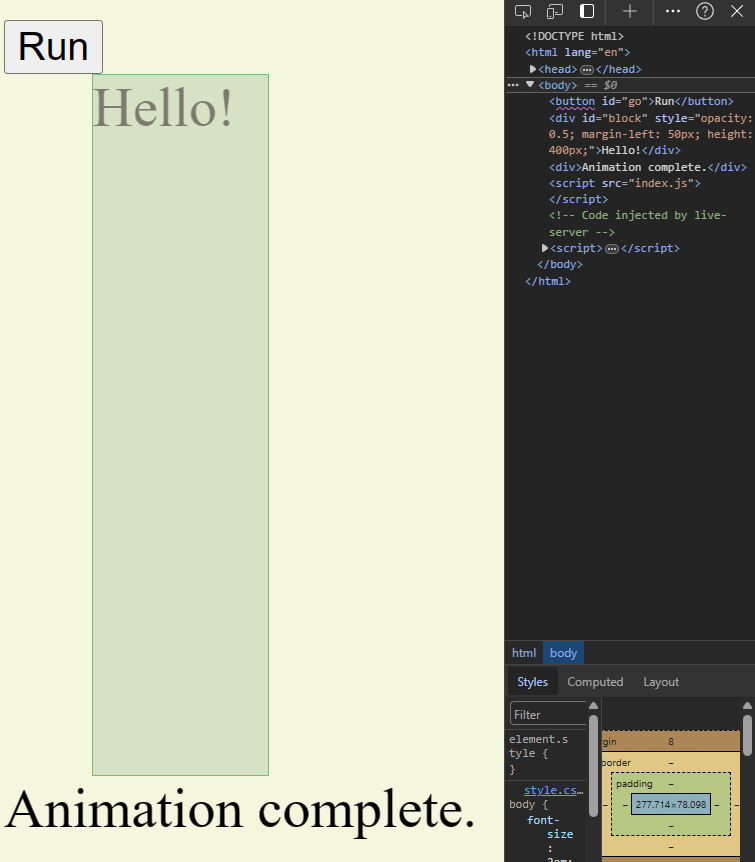
Initial state:



In progress state:

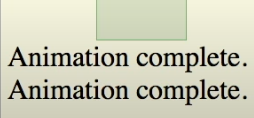


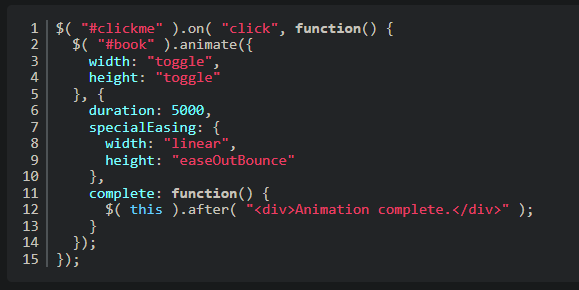
Completed state:



Default easing is "swing", there is no easeIn/easeOut, according to api.jquery, there is a specialEasing property that can be used instead, but requires the use of a plugin.

Clicking the animate button again, will continue to add 50px to the left margin, but because the other properties are already at their defined values, there is no animation for them to transition to; height is already 400px and opacity already at 0.5; A new "Animation complete." is appended to the DOM at the end.





## Chaining animations()

// chaining animations

**$**("#go").**click**(function () {

**$**("#block")

  .**animate**({width: "90%"}, 1000)

  .**animate**({fontSize: "40px"}, 1000)

  .**animate**({borderLeftWidth: "30px"}, 1000)

})

When method chaining animations, the previous animation must complete before the next one runs.

## Get and set with http, text, and attr

Selecting Text within a paragraph tag using jquery:

  <p id="test">Learn code with <a href="https://freecodecamp.org" id="fcc">freeCodeCamp</a>!</p>

  <button id="btn">Show Content</button>

index.js:

**$**(function () {

**$**("#btn").**click**(function () {

    // select text within p tag using jquery

    // this only collects the text content and not any of the html

    console.**log**(**$**("#test").**text**());

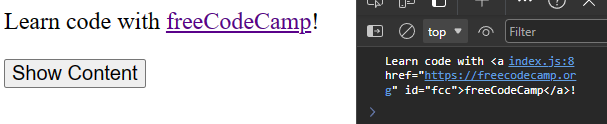
  });

});



Notice that the text method did not print any of the anchor tag to the console.

If we instead use the **.html()** method instead, the method will retrieve the html content and print it to the console.



    console.**log**(**$**("#test").**html**());

The index.js:8 information is telling us that this print statement came from index.js on line 8; helpful for debugging print statements

The **.attr()** method can be used to fetch the value of an html attribute.

    console.**log**(**$**("#fcc").**attr**("href"));

This code will loom for the element with an id of fcc and selects the href attribute. It returns the freecodecamp url associated to the "href" key.

## Setting content

By providing an argument to the .text method, we can replace the existing text for the selected element.

  // select btn2 to set content

**$**("#btn2").**click**(function () {

    // when a string is passed to text, it will replace the

    // existing text for the selected element

**$**("#test").**text**("freeCodeCamp is awesome!");

  });

This function selects the element with the id of #test and sets its text content to the string passed to the text() method. An empty string will clear the text content. (parentheses must be empty to retrieve text content)

You cannot pass HTML to the text method. It will print the literal characters for the html tags instead of writing it as if it were HTML.



  // add HTML with .html()

**$**("#btn3").**click**(function () {

    // use the .html() method to add html tags

**$**("#test").**html**("freeCodeCamp is <b>awesome!</b>");

  });



## Update an input field

HTML

  <p>Name: <input type="text" id="name" value="Bobby Tables"></p><br>

  // set value of an input field with .val()

**$**("#btn4").**click**(function () {

**$**("#name").**val**("Spongebob Squarepants")

    // print the value to console

    console.**log**(**$**("#name").**val**());

  })

All of these methods also come with callback functions. These callback functions have two parameters, the first one is the index of the current element out of the list of elements selected, and the current/old value. You can return the string which you can use as the new value from the function.

  // using a callback

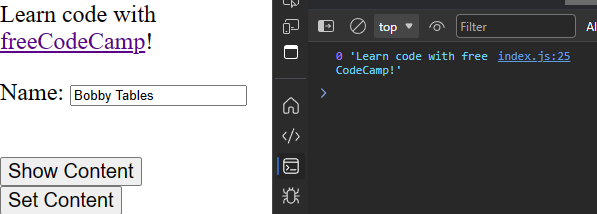
**$**("#btn2").**click**(function() {

**$**("#test").**text**(function (i, origText) {

      console.**log**(i, origText);

    })

  })



Clicking on the setContent button runs this code. It prints 0 for the index of the element matching the #test selector and also returns the text content. Because we used the .test() method, no HTML was retrieved.

### Concatenate to existing text

Using the callbacks from the above example, we can append (or prefix) a new string to the end of the existing text content.

  // using a callback

**$**("#btn2").**click**(function () {

**$**("#test").**text**(function (i, origText) {

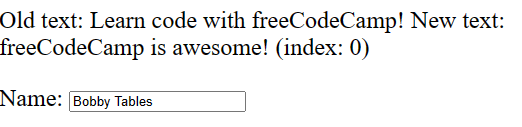
      // console.log(i, origText);

      // append text

      return "Old text: " + origText + " New text: freeCodeCamp is awesome! (index: " + i + ")";

    });

  });



This time, select "p" instead.

**$**("#btn2").**click**(function () {

      // selecting all p elements instead of a single id

**$**("p").**text**(function (i, origText) {

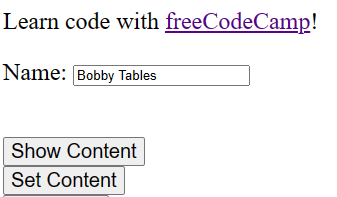
      // console.log(i, origText);

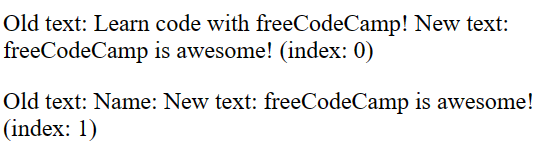
      // append text using a return statement

      return "Old text: " + origText + " New text: freeCodeCamp is awesome! (index: " + i + ")";

    });

  });





The previous code written within paragraph tags has been replaced. Because we the callback gave us the original text, we can print what was originally there. In this example, "**learn code with freeCodeCamp**" and "**Name:**" have been replaced with "freeCodeCamp is awesome and the index of the selected p element.

## Add and Remove DOM elements

The append and prepend methods add new elements as child elements to the target. The after and before methods add new sibling elements.

<body>

  <h2>jQuery: Adding and Removing elements</h2>

  <div class="target">

    <p>This div is a target for all sorts of things.</p>

    <p>It doesn't even know what could happen to it</p>

  </div>

  <script src="index.js"></script>

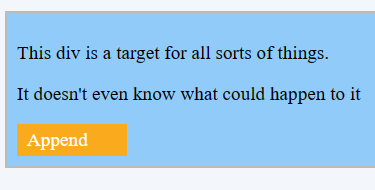
</body>

Here is an example where we append a div with a class of child, and the text content will be "Append", to the div with a class of "target".

**$**(function () {

**$**(".target").**append**("<div class='child'>Append</div>");

});



The append method is the equivalent of using the html() method with a callback:

// Adding and removing elements

**$**(function () {

**$**(".target").**html**( function (i, original) {

    return original + "<div class='child'>Append</div>";

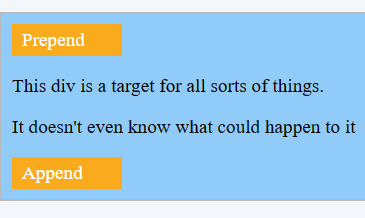
  })

});

It uses much less syntax and is a lot clearer.

Prepend is the exact same, but prepends the contents to the original contents of the target as a child element:

**$**(".target").**prepend**("<div class='child'>Prepend</div>");



These methods can also be method chained instead of selecting the same target repeatedly:

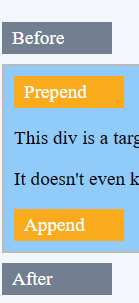
**$**(".target")

    .**append**("<div class='child'>Append</div>")

    .**prepend**("<div class='child'>Prepend</div>")

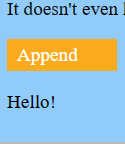
    .**before**("<div class='sibling'>Before</div>")

    .**after**("<div class='sibling'>After</div>");



It's possible to add multiple elements at a time by comma separating the strings passed to one of these methods.

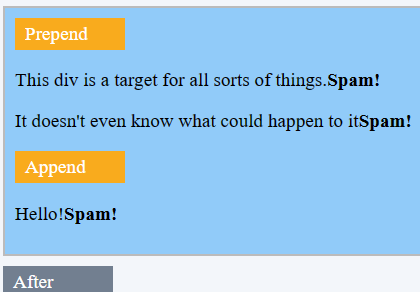
    .**append**("<div class='child'>Append</div>", "<p> Hello!</p>")



(also appears that a list can be passed instead multiple args)

Be careful with targets, if you're broadly selecting tags, you may end up adding content where you never intended for it to be.

**$**("p").**append**("<b>Spam!</b>")



Spam was appended to the end of every "p" element because the target was too broad.

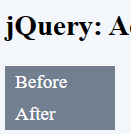
To remove elements and content, there are mainly two jquery methods: remove() and empty().

Remove() removes the selected element and its child elements.

Empty() removes the child elements from the selected element.

    // .remove() removes the target and all of its child elements

**$**(".target").**remove**();

 // before and after were added before remove() was called. They are siblings to .target which is why they were not removed()

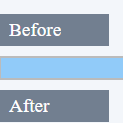
Removing all "p" elements:

**$**("p").**remove**();

 // only divs remain

    // .empty() keeps the target but removes all of child elements

**$**(".target").**empty**();



The remove() method can take an argument which will match to a class. This can be used to selectively remove only certain elements with the specified class:

    // using a filter to selectively remove an element

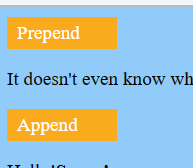
    // selects all p elements and removes the one with a class of .filter

**$**("p").**remove**(".filter")

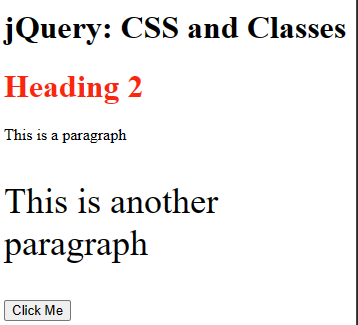
// HTML

    <p class="filter">This div is a target for all sorts of things.</p>

    <p>It doesn't even know what could happen to it</p>



## CSS and Classes



  <h1>jQuery: CSS and Classes</h1>

  <h2 class="red">Heading 2</h1>

  <p>This is a paragraph</p>

  <p class="big">This is another paragraph</p>

  <button>Click Me</button>

  <script src="index.js"></script>

.red {

  color: red;

}

.big {

  font-size: 36px;

}

In JS, we added a click event to the button element. We can use the .css() method as a getter to fetch the value of various css properties.

**$**(function () {

**$**("button").**click**(function () {

    // using the .css() method to fetch values of various css properties

    // Remember that in js, we use camelCase while CSS uses kebab-case

    // only returns the value of the first p element

    console.**log**( **$**("p").**css**("fontSize"));

  })

})



To specify a different element rather than the first encountered, you can concatenate the class (or id) after specifying the element.

    // using . to select p element with a class of big

    console.**log**(**$**("p.big").**css**("fontSize"));

This selects the first p element with a class of big, and fetches the font-size Css property and prints it to the console.



### Setting a CSS property

To set a css property, specify the property followed by a comma separated value. If a number is given for font size, then it is converted to px, probably best to not rely on this functionality though and explicitly provide a string and state that you want the size in pixels.

**$**("p.big").**css**("fontSize", 100);

**$**("p.big").**css**("fontSize", "100px");

If you would like to set multiple properties at once, pass an object instead.

If you would like to use fallback fonts, you must pass a list as the value.

**$**("p").**css**({

      fontSize: "50px", // a Number works too -> 50

      fontFamily: ["Arial", "Consolas", "sans-serif"],

      color: "blue",

      backgroundColor: "yellow",

    });

It is not recommended to use the .css() method as a setter in production code. It is best to keep presentational information out of the javascript code. It is better to keep all of your css styles in your .css file and to instead use javascript to add/remove classes.

### Adding a class with JS

**$**(function () {

**$**("button").**click**(function () {

    // when using addClass(), do not include the "."

**$**("h1").**addClass**("red");

  });

});

### Removing a class

**$**("p").**removeClass**("big");

### toggleClass()

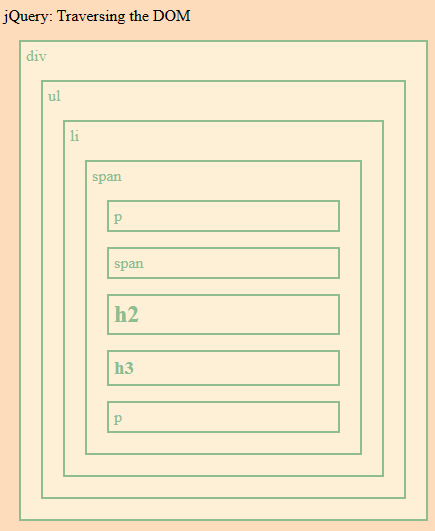
**$**("h2").**toggleClass**("red");

### hasClass()

    console.**log**(**$**("h2").**hasClass**("red")); // true/false

useful with an if statement.

## Traversing the DOM in jQuery



Finding specific html elements based on their relationship to other html elements.

.container \* {

  display: block;

  border: 2px solid DarkSeaGreen;

  color: DarkSeaGreen;

  padding: 5px;

  margin: 15px;

  background: PapayaWhip;

}

body {

background: PeachPuff;

}

The boxes are used to help tell the relationship between parent, child and sibling elements.

The method names are very straight forward, use .parent() to select the parent element, etc.

**$**(function () {

  // selects the parent of the li element

**$**("li").**parent**().**css**({

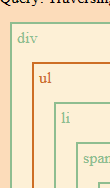
    // keys can be quoted or unquoted

    "color": "chocolate",

    border: "2px solid chocolate"

  })

})



You can call a method multiple times to traverse the tree, .parent().parent() selects the div element.

Remember to be specific when using a selector.

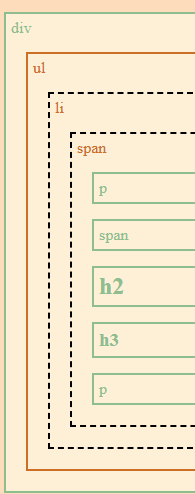
**$**("span").**parent**().**css**({

    // there are multiple spans in the DOM

    "color": "chocolate",

    border: "2px dashed black"

  })



There is also the .parent**s**() method which selects all elements up to the root.

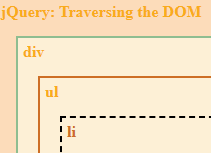
**$**("li").**parents**().**css**({

    // there are multiple spans in the DOM

    color: "orange",

    fontWeight: "bold"

  })



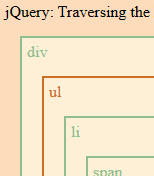
    // limit .parents() with parentsUntil()

**$**("li").**parentsUntil**("div").**css**(

      {color: "chocolate",

      border: "2px solid chocolate"}

      )



The only element between the li and the div is the ul.

## Selecting children

This works the same way as parent but in reverse.

**$**("li").**children**().**css**({ color: "chocolate", border: "2px solid chocolate" });

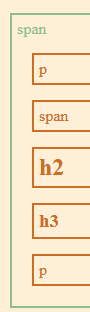


This only traverses a single level down, if there were multiple li elements, all of their first level descendents would be styled.

Again, this time targeting the span elements:

  // only selects direct descendants

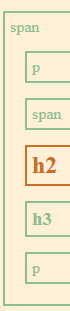
**$**("span").**children**().**css**({ color: "chocolate", border: "2px solid chocolate" });



.children() can take a filter which allows you to select only specific elements:

  // filter

**$**("span").**children**("h2").**css**({ color: "chocolate", border: "2px solid chocolate" });



The .find() method selects all descendants matching a filter value

  // find - selects descendants

**$**("ul").**find**("p").**css**({ color: "chocolate", border: "2px solid chocolate" })

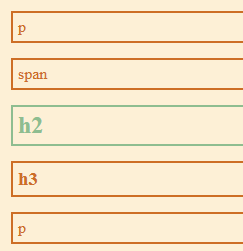
});

If you would like to find all children, use an "\*" as the value

Select siblings()

  // siblings

**$**("h2").**siblings**().**css**({ color: "chocolate", border: "2px solid chocolate" })

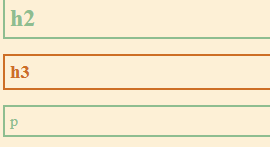
 // h2's siblings are styled

A filter can be provided to the siblings() method to select specific sibling elements.

Select the next sibling:

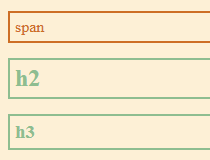
  // next sibling

**$**("h2").**next**().**css**({ color: "chocolate", border: "2px solid chocolate" })



Use .prev() to select the previous sibling:

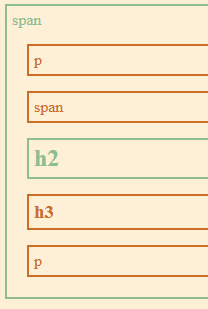
**$**("h2").**prev**().**css**({ color: "chocolate", border: "2px solid chocolate" })



There is also nextAll() and prevAll()

**$**("h2").**nextAll**().**css**({ color: "chocolate", border: "2px solid chocolate" })

**$**("h2").**prevAll**().**css**({ color: "chocolate", border: "2px solid chocolate" })

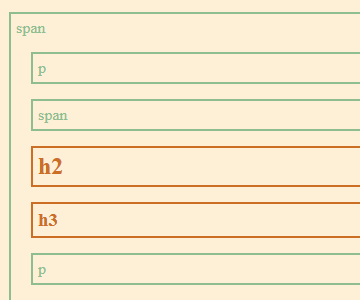


nextUntil() selects the next child elements until a matching element stop condition is met.

 // nextUntil/prevUntil

**$**("span.child").**nextUntil**("p").**css**({ color: "chocolate", border: "2px solid chocolate" })

This starts at the span with a class of child and stops at the h3 element (exclusive)



## AJAX with jQuery

AJAX allows you to exchange data from a server without having to reload the entire page (think react). The jQuery AJAX methods can be used to request or send text, HTML, XML or JSON from a remote server.

Different browsers have different syntax for AJAX implementation, but jquery makes it work for all browsers with a single line of code.

function **loadDoc**() {

  var xhttp = new **XMLHttpRequest**();

  xhttp.**onreadystatechange** = function () {

    if (this.readyState == 4 && this.status === 200) {

      document.**getElementById**("demo").innerHTML = this.responseText;

    }

  };

  xhttp.**open**("GET", "https://cors-anywhere.herokuapp.com/ http://carnes.cc/code/ajax\_example.txt", true);

  xhttp.**send**()

}

What's happening is that we are fetching data from another website.

The url is a little convoluted due to CORS policy.

The cors-anywhere url is used because of a CORS error as well as accessing data from an insecure website (http instead of https). cors-anywhere is a CORS proxy which can be used in development if you're having trouble getting data from a website.

In a real situation, you would want both websites to be secure (https://) and add some specific headers.

The .load() method replaces all of the js code required to access a remote website and etch data.

function **loadDoc**() {

**$**("#demo").**load**(

    "https://cors-anywhere.herokuapp.com/http://carnes.cc/code/ajax\_example.txt"

  );

}

Just place the url within the load() method and you run AJAX with this code.

With load, we can also add a callback function which runs after the site loads/fails to load.

function **loadDoc**() {

  // load + optional callback

**$**("#demo").**load**(

    "https://cors-anywhere.herokuapp.com/http://carnes.cc/code/ajax\_example.txt",

    // optional callback function

    function (responseTxt, statusTxt, xhr) {

      if (statusTxt == "success") {

**alert**("External content loaded successfully!");

        // console.log(responseTxt);

        // responseTxt = Subscribe to the freeCodeCamp YouTube channel!

      }

      if (statusTxt == "error") {

**alert**("Error: " + xhr.status + ": " + xhr.statusText);

      }

    }

  );

}

// too many requests, but code should be correct now.

**$.get("url")** , just like the load() method, will get data from the main server. The main difference is that load() is going to load the data directly into an element while get is not necessarily going to load that data into an element.

  // $get() similar to load but will not load data into an element

  $.**get**("https://cors-anywhere.herokuapp.com/http://carnes.cc/code/ajax\_example.txt",

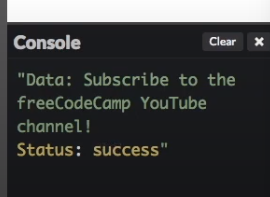
    function (data, status) {

      console.**log**("Data: " + data + "\nStatus: " + status);

    })

The callback function is responsible for what happens to the text; in this case, were are simply printing it to the console.

// error 429 too many requests



    // $.post() is similar, but you are passing data to a server instead of asking for data

    // takes an extra parameter as an {object} to send data

    $.**post**("https://cors-anywhere.herokuapp.com/http://carnes.cc/code/ajax\_example.txt",

    {

      name: "SpongeBob SquarePants",

      city: "Bikini Bottom"

    },

    // handle any server response

      function (data, status) {

        console.**log**("Data: " + data + "\nStatus: " + status);

      })

## Comparing jQuery and vanilla JS

Originally vanilla JS was a lot more complicated and jQuery simplified a lot of working with DOM elements.

// jQuery way

let $elem = **$**(".someClass");

$elem.**remove**();

$elem.**prepend**($otherElement);

$elem.**before**($otherElement);

$elem.**after**($otherElement);

$elem.**replaceWith**($otherElement);

$elem.**closest**("div") // parent

// vanilla JavaScript

let elem = document.**querySelector**(".someClass");

elem.**remove**();

elem.**prepend**(otherElement);

elem.**before**(otherElement);

elem.**after**(otherElement);

elem.**replaceWith**(otherElement);

elem.**closest**("div") // parent

All of these are pretty much the same aside from the element selector.

Where js and jQuery start to differ would be effects/built in styling.

$elem.**fadeIn**();

// see style.css for initial element state

elem.style.display = "block";

**requestAnimationFrame**(() => (elem.style.opacity = 1));

/\* style.css

fadeIn for vanilla js \*/

.thingy {

  display: none;

  opacity: 0;

  transition: 0.8s;

}

// using web animations API

elem.**animate**([

  {

    transform: "translateY(-1000px) scaleY(2.5) scaleX(2.5)",

    transformOrigin: "50% 0",

    filter: 'blur(40px)',

    opacity: 0

  },

  {

    transform: "translateY(0) scaleY(1) scaleX(1)",

    transformOrigin: "50% 50%",

    filter: 'blur(0)',

    opacity: 1

  },

])

The web animations API is more powerful than jquery.

// AJAX

$.**ajax**('https://some.url', {

**success**: (data) => {/\* do something with data \*/}

})

// fetch API for vanilla js

**fetch**("https://some.url")

.**then**(response => response.**json**())

.**then**(data => {

  // do something with data

})

polyfill can be used to handle browser compatibility

add this script

  <script src="https://polyfill.io/v3/polyfill.min.js"></script>