**The DOM Tree**

jQuery makes it easy to target HTML elements by tag name, class, and id. We can also dynamically target a single element in a given class using by accessing an event's .currentTarget attribute. In this lesson, we'll go even further. You will learn how to target elements based on their position relative to other elements.

Before we begin, let's review the Document Object Model or DOM. The DOM is a tree of objects based on the HTML document that is created by the browser when it loads a page. Every element in this page exists on some branch of the tree, with elements above it, and possibly next to or below it.

<div class='parent'> <div class='child' id='sibling1'> <p class='grandchild'></p> </div> <div class='child' id='sibling2'></div> <div class='child' id='sibling3'></div> </div>

Elements inside other elements are considered descendants. We use family words to refer to these relationships. For instance, an outer <div> would be considered the parent of any <div> element inside it (those with class 'child' above). Any other <div>s inside of 'parent' on the same level as 'child' are considered siblings of each other. A <p> in any 'child' would itself be a child of the 'child'element that contains it and a grandchild of 'parent'.

# Children

According to the DOM tree, the outermost element is the parent of all elements inside of it. Therefore, the HTML elements inside of the outer element are children. The jQuery .children() method allows us to target these elements.

<div class='parent' id='holder'> <div>Child <span>1</span></div> <div>Child <span>2</span></div> <div>Child <span>3</span></div> </div>

const $kids = $('#holder').children(); $kids.on('click', event => { $(event.currentTarget).css('border', '1px solid black'); });

In the example above, the $kids variable refers to all children of the element with id 'holder'(the divs inside '#holder'). The .on() method adds the click event handler to these $kids. When one of them is clicked, jQuery will add a border around it that is 1px wide and solid black.

It is important to note that only the direct descendants of '#holder' are considered children. Any elements inside the child elements of '#holder', such as the <span> elements, will not be targeted by the .children() method.

# Parent & Siblings

In addition to the .children() method we covered in the last exercise, there are two methods you can use to select the parent and siblings of an element.

$('.choice').on('click', event => { $(event.currentTarget).parent().hide(); });

In the example above, the .parent() method targets the parent element of '.choice'elements and removes it from the DOM.

$('.choice').on('click', event => { $(this).siblings().removeClass('selected'); $(event.currentTarget).addClass('selected'); });

In the code above, the .siblings() method targets elements adjacent to the clicked '.choice' and removes the 'selected' class from any previously clicked '.choice's. Then the 'selected' class is added only to the clicked '.choice'.

# Closest

To select an element close to the current element, we can use jQuery's .closest()method.

The .closest() method will travel up the DOM tree to find a specified selector closest to it. Its syntax looks like:

$('.example-class-one').closest('.another-class');

In the example above:

* The .closest() method is called on '.example-class' elements.
* The method then targets the element selected by the .closest() method with a class of '.another-class'.

<div class='.another-class'> <p class='.example-class-one'>1</p> </div> <div class='.another-class'> <p class='.example-class-two'>2</p> </div>

Given this HTML, the jQuery above would select the <div> element that wraps the paragraph with a value of 1, because it is the closest element, up the DOM tree, with the class .another-class.

# Next

Sometimes you don't want to target all the siblings of an element — you just want to target the next one. That's where the aptly-named .next() method comes in!

The code below is HTML for a menu. The list of food types is hidden, <ol style='display:none'>.

<div class='heading'>MENU</div> <ol style='display: none'> <li>Appetizers</li> <li>Entrees</li> <li>Salads</li> <li>Sides</li> <li>Desserts</li> </ol>

Since the div and <ol> exist on the same level of the DOM, they are siblings. Since there are no elements between them, the <ol> is the next sibling of '.heading'. We can add an event handler to the div element and use the .next() method to show and hide the <ol>using the .toggle() method.

const $heading = $('.heading'); $heading.on('click', () => { $(event.currentTarget).next().toggle(); });

In the example above, the .on() method attaches the click event handler to $heading. Then the callback function will toggle the class of the $heading's next sibling, the ol element.

It's important to note that jQuery also has a method called [.prev()](https://api.jquery.com/prev/) that can look at the previous sibling.

# find

Sometimes we want to target an element that lives inside another, but we don't want to use the .children() method, since that might target more elements than we need. That's where the .find() method comes in. This method finds and targets singular or multiple elements that are descendants of an element. Unlike the .children() method, it traverses all descendants of the specified element, not just the first level down.

const $items = $('.myList').find('li');

The .find() method takes a parameter that specifies how to filter results. This parameter is just like anything you might use to select a jQuery object, ('#id', '.class', tag, etc.). .find() will return all descendants that match the passed in selector. In the example above, the .find() method will return all <li> child elements inside the '.myList' element.

**Review: Traversing the DOM**

Understanding how elements relate to each other in the DOM makes it easy to efficiently select elements. We have covered several methods in this lesson including:

* .children() to target an element's child elements.
* .siblings() to target elements adjacent to an element.
* .parent() to target an element's parent.
* .closest() travels up the DOM tree from the current element to target the closest element with a given selector.
* .next() to target the element immediately following the selected element.
* .prev() to target the element that is immediately preceding the selected element.
* .find() to target descendant elements by some selector, ie- class, id, tag etc.

In addition to these methods, there are even more, including .prevUntil(), .nextUntil() and others. To get an idea, check out [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript) reference for jQuery Traversing.