# Week 5 – jQuery and the DOM

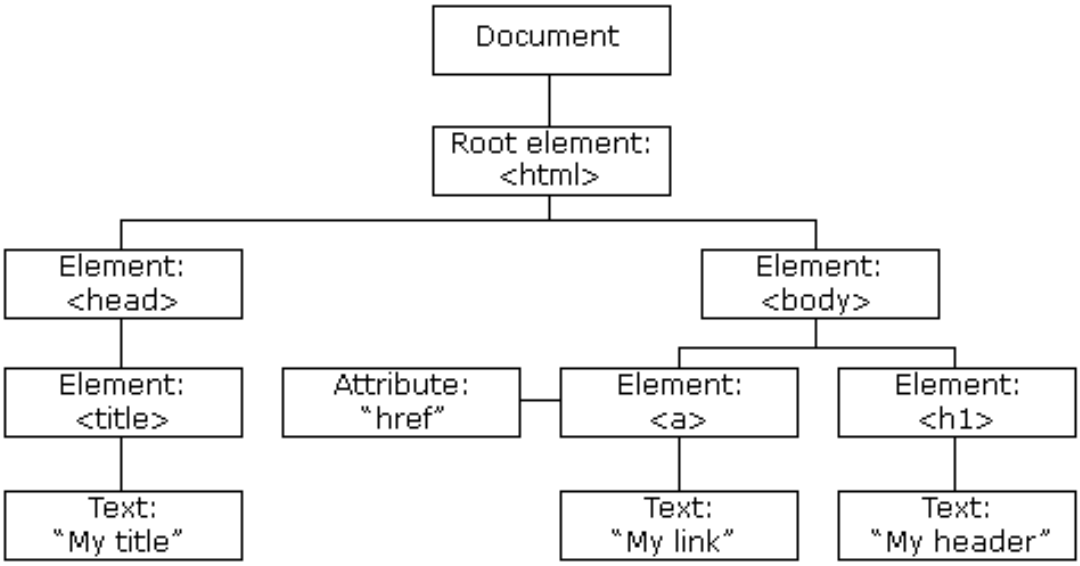
## Introduction

Last week we looked at how jQuery syntax is constructed to target selectors. We also looked at how to use the inbuilt effects, `animate()` and `css()`, and how to trigger them from events such as `click()` or `mouseenter()`.

This week we'll be looking at some specific jQuery commands that help us navigate the Document Object Model. These commands, however, are only beneficial if we have good HTML structure.

## The Document Object Model

It's useful for us to understand an HTML document as a tree:




As we know, HTML tags are frequently nested inside of one another. For example:

```
<section>
   <h1>A Heading</h1>
   <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut
   mattis mi a nulla eleifend vitae sagittis <span>libero</span>
   imperdiet. Vivamus consequat risus sit amet tortor laoreet vitae
   mattis enim ornare.</p>
</section>
```

In the above code, we have a `<h1>` and `<p>` nested inside of a `<section>`. Additionally, there is a `<span>` nested inside of the `<p>`.

jQuery provides us with some tools for describing the relationship between elements:

| Command | Description |
|---|---|
| `.next()` | Refers to the next HTML element. |
| `.prev()` | Refers to the previous HTML element. |
| `.find('selector')` | Allows us to refer to a nested element. |
| `$(this)` | Refers to the particular instance of selector that is being currently interacted with. |

These tools are incredibly handy because they allow us to get around a lot of unnecessary coding.

## $(this)

Let's start by explaining how `$(this)` works. Let's say we have an HTML document with several `<h1>` tags. When we hover over an `<h1>` we want its colour to turn red (`#FF0000`).

If we were to code the following, then whenever we placed the mouse over *any* of the `<h1>` tags, they would *all* turn red, but we only want the current instance to change colour:

```
$('h1').mouseenter(function() {
   $('h1').css({
      color: '#ff0000'
   });
});
```

We could create a series of IDs to give a unique identifier to each one. Whilst this would work, it would result in a lot of code because we'd need to code the event for each individual `<h1>`:

| HTML | jQuery |
|---|---|
| `<h1 id="first">Heading</h1>`<br><br>`<h1 id="second">Heading</h1>`<br><br>`<h1 id="third">Heading</h1>` | `$('h1#first').mouseenter(function() {`<br>`  $('h1#first').css({`<br>`  color: '#ff0000'`<br>`  });`<br>`});`<br>`$('h1#second).mouseenter(function() {`<br>`  $('h1#second).css({`<br>`  color: '#ff0000'`<br>`  });`<br>`});`<br>`$('h1#third).mouseenter(function() {`<br>`  $('h1#third).css({`<br>`  color: '#ff0000'`<br>`  });`<br>`});` |

Luckily, jQuery provides us with a simple way around this using `$(this)`.

| HTML | jQuery |
|---|---|
| `<h1 id="first">Heading</h1>`<br><br>`<h1 id="second">Heading</h1>`<br><br>`<h1 id="third">Heading</h1>` | `$('h1').mouseenter(function() {`<br>`  $(this).css({`<br>`  color: '#ff0000'`<br>`  });`<br>`});` |

In this example, we can see that the mouseenter event and ensuing function is attached to the `<h1>` selector, meaning that placing the mouse over *any* `<h1>` will trigger the function. The function, however, will only affect the *specific* `<h1>` that currently has the mouse over it rather than every single one, and that is what `$(this)` refers to – the *current instance* that is being interacted with.

Clever huh? Using `$(this)` the browser knows which specific instance is being affected. Using `.next()` and `.prev()` we can then affect elements that precede or follow the current instance … *and this is where we need good HTML structure*.

## Let's Build an Accordion

You can see an example of the accordion at work on the MMCC2041 page. The code that makes it work is *incredibly* simple, but its efficacy is entirely dependent on good HTML structure.

What we have here is a structure like this:

```
<h3>Heading 1</h3>
<p class="hiddenContent">Lorem ipsum dolor sit amet … interdum augue
feugiat.</p>

<h3>Heading 2</h3>
<p class="hiddenContent"> Donec ante risus, iaculis sed faucibus … ac
turpis egestas.</p>
```

There's some very simple styling applied to `<h3>` tag and the `.hiddenContent` class (which is applied to the `<p>` tags):

```
h3 {
  border: 1px solid black;
  padding: 10px;
  background-color: black;
  color: white;
  margin: 1px 0 0 0;
  cursor: pointer;
}
.hiddenContent {
  display: none;
  margin-left: 10px;
}
```

So, when the page loads, any element with the `.hiddenContent` class is hidden.

The jQuery code is also very simple. It relies heavily on good HTML structure. We bind a click event to the <h3> and then trigger the slides:

```
$('h3').click(function() {
   $('.hiddenContent').slideUp(300);
   $(this).next().slideDown();
});
```

Let's break this down into steps:

1. When any <h3> tag is clicked it triggers the function.
2. The first thing to happen is that every element with the `.hiddenContent` class is told to slideUp. This ensures that only one block of hidden content can be displayed at any given time.
3. jQuery then determines which particular instance of `<h3>` has been clicked and tells the *next* element in the HTML structure to slideDown. If you look back at the HTML code on the previous page, you'll see that the *next* element along from the `<h3>` is always a `<p class="hiddenContent"> </p>`.

Too easy, and an effective way of a) condensing a lot of content and b) focusing your user's attention on the particular bit of content they've selected to view.

## When to use .find()

jQuery's `.prev()` and `.next()` are pretty straightforward to use, but they only refer to the previous and next HTML element, they don't help when an element is nested inside of another. For example:

```
<section>
   <h1>A Heading</h1>
   <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut
   mattis mi a nulla eleifend vitae sagittis <span>libero</span>
   imperdiet. Vivamus consequat risus sit amet tortor laoreet vitae
   mattis enim ornare.</p>
   <p>Phasellus eget augue ac arcu tincidunt mattis sit amet ac
   turpis. Pellentesque congue risus a ipsum dapibus sed interdum
   ligula tincidunt.</p>
</section>
```

If we wanted to affect the `<span>` which is nested in the `<p>` when the `<h1>` is clicked, then `.next()` won't help us. Why? Well, the next element from the `<h1>` is the `<p>`. It is possible to chain more than one instance of `.next()` and `.prev()` together to traverse more than one element. For example:

```
$('h1').click(function() {
   $(this).next().next().css({ color: '#FF0000' });
});
```

But in the HTML example above, using `.next().next()` takes us to the second `<p>` tag not the `<span>`. This is where we need to use `.find('selector')` to locate the selector we want to target.

The following code shows us how to target a nested selector – in this case the  inside the <p>. **This is important because it relates to the second coding test you have to complete.**

```
$('h1').click(function() {
   $(this).next().find('span').css({ color: '#FF0000' });
});
```

| Code | Explanation |
|---|---|
| `$(this)` | The particular <h3> that is clicked. |
| `.next()` | The next HTML element in the structure. In this case that's the <p> tag. |
| `.find('span')` | This locates the  tag. |

So, as you can see, we can use `.prev()`, `.next()`, `.find()` and any combination thereof to traverse the HTML structure relative to the position of whatever element is treated as `$(this)`.