

9. Adding Media

We browse the Internet in search of interesting and informative content, which we usually find in the form of plain text. To accompany this plain text, HTML provides ways to embed rich media in the form of images, audio tracks, and videos, as well as to embed content from another web page in the form of an inline frame.

The ability to include images, audio tracks, videos, and inline frames within websites has been around for some time. Browser support for images and inline frames has generally been pretty good. And while the ability to add audio tracks and videos to a website has been around for years, the process has been fairly cumbersome. Fortunately, this process has improved and is much easier with support directly from HTML.

Today, we can freely use images, audio, video, and inline frames knowing that this content is supported across all major browsers.

Adding Images

To add images to a page, we use the `` inline element. The `` element is a self-containing, or empty, element, which means that it doesn't wrap any other content and it exists as a single tag. For the `` element to work, a `src` attribute and value must be included to specify the source of the image. The `src` attribute value is a URL, typically relative to the server where a website is hosted.

In conjunction with the `src` attribute, the `alt` (alternative text) attribute, which describes the contents of an image, should be applied. The `alt` attribute value is picked up by search engines and assistive technologies to help convey the purpose of an image. The `alt` text will be displayed in place of the image if for some reason the image is not available.

```

```

Supported Image Formats

Images come in a variety of different file formats, and each browser may support (or not support) different formats. By and large, the most commonly supported formats online are `gif`, `jpg`, and `png` images. Of these, the most widely used formats today are `jpg` and `png`. The `jpg` format provides quality images with high colour counts while maintaining a decent file size, ideal for faster load times. The `png` format is great for images with transparencies or low colour counts. We most commonly see `jpg` images used for photographs and `png` images used for icons or background patterns.

Sizing Images

It is important to identify the size of an image in order to tell the browser how large the image should be before the page even loads; thus the browser can reserve space for the image and render the page faster. There are a few different ways to size images so that they work well on a page. One option is to use the `width` and `height` attributes directly within the `` tag in HTML.

Additionally, images may be sized using the `width` and `height` properties in CSS. When both the HTML attributes and CSS properties are used, the CSS attributes will take precedence over the HTML attributes.

Specifying either a width or height will cause the other dimension to adjust automatically to maintain the aspect ratio of the image. As an example, if we want an image to be 200 pixels tall but are less specifically concerned about how wide it is, we can set the `height` to 200 pixels, and the width of the image will adjust accordingly. Setting both a `width` and `height` will work also; however, doing so may break the aspect ratio of an image, causing it to appear distorted.

```
img {  
  height: 200px;  
  width: 200px;  
}
```

While using the `width` and `height` attributes directly in HTML provides some semantic value by noting an image's original size, it can be difficult to manage numerous images that all need to be the same size. In this event, it's common practice to use CSS to resize the images.

Positioning Images

We can use a number of different approaches to position images on a web page. By default, images are positioned as inline-level elements; however, their positions may be changed using CSS, specifically the `float`, `display`, and box model properties, including `padding`, `border`, and `margin`.

Inline Positioning Images

The `` element is by default an inline-level element. Adding an image without any styles to a page will position that image within the same line as the content that surrounds it. Additionally, the height of the line in which an image appears will be changed to match the height of the image, which can create large vertical gaps within that line.



```
<p>Gatsby is a black, brown, and white hound mix puppy who loves howling at fire trucks and collecting belly rubs.  Although he spends most of his time sleeping he is also quick to chase any birds who enter his vision.</p>
```

Leaving images untouched in their default positioning isn't too common. More often than not, images are displayed as block-level elements or are floated flush to one side.

Block Positioning Images

Adding the `display` property to an image and setting its value to `block` forces the image to be a block-level element. This makes the image appear on its own line, allowing the surrounding content to be positioned above and below the image.

```
img {  
  display: block;  
}
```

Positioning Images Flush Left or Right

Sometimes displaying an image as `inline` or `block`, or perhaps even `inline-block`, isn't ideal. We may want the image to appear on the left or right side of its containing element, while all of the other content wraps around the image as necessary. To do this, we use the `float` property with a value of either `left` or `right`.

Remembering back to Practical 5, "Positioning Content," we recall that the `float` property was originally intended to position images to the left or right of a containing element. Now we'll use it for that original purpose.

Floating an image is a start; however, all other content will align directly against it. To provide spacing around an image, we'll use the margin property. Additionally, we can use the `padding`, `border`, and `background` properties to build a frame for the image, if desired.

```
img {  
  background: #eaeaed;  
  border: 1px solid #9799a7;  
  float: right;  
  margin: 8px 0 0 20px;  
  padding: 4px;  
}
```

When to Use an Image Element vs. a Background Image

There are two primary ways to add images to a web page. One way, as covered here, is to use the `` element within HTML. Another way is to use the `background` or `background-image` property within CSS to assign a background image to an element. Either option will do the job; however, they each have specific use cases.

The `` element within HTML is the preferred option when the image being used holds semantic value and its content is relevant to the content of the page.

The `background` or `background-image` property within CSS is the preferred option when the image being used is part of the design or user interface of the page. As such, it's not directly relevant to the content of the page.

The `` element is quite popular, and when it was originally added to the HTML specification it forever changed the way websites were built.

Exercise 9.1

Now that we know how to add and position images on a page, let's take a look at our Styles Conference website and see where we can add a few images.

1. Let's begin by adding some images to our home page. Specifically, we'll add an image within each of the teaser sections promoting a few of our pages.

Before we jump into the code, though, let's create a new folder named "images" within our "assets" folder. Within the "images" folder we'll add three images: `speakers.jpg`, `schedule.jpg`, and `venue.jpg`. (For reference, these images may be downloaded from BBL.)

Then, inside our `index.html` file, each teaser section has an `<a>` element wrapping both an `<h3>` and an `<h5>` element. Let's move the `<h5>` element above the `<a>` element and replace it with an `` element. The `src` attribute value for each `` element will correspond to the folder structure and filename we set up, and the `alt` attribute value will describe the contents of each image.

The HTML for our first teaser, for the Speakers page, will look like this:

```
<section class="teaser col-1-3">
  <h5>Speakers</h5>
  <a href="speakers.html">
    
    <h3>World-Class Speakers</h3>
  </a>
  <p>Joining us from all around the world are over twenty fantastic speakers, here to share their stories.</p>
</section>
```

Let's continue this pattern for both the Schedule and Venue page teasers, too.

- Now that we've added a few images to our home page, we'll need to clean up their styles a bit and make sure they properly fit into the layout of our page.

Since images are inline-level elements by default, let's change our images within the teaser sections to block-level elements. Let's also set their maximum `width` to `100%` to ensure they don't exceed the width of their respective columns. Changing this `width` value is important as it allows our images to adjust with the width of the columns as necessary.

Lastly, let's round the corners of the images slightly and apply `22` pixels of bottom `margin` to the images, providing a little breathing room.

Add these new styles to our existing home page styles in the "HERO" section (using the teaser class as a qualifying selector for the `` elements), our CSS will look like this:

```
.teaser img {
  border-radius: 5px;
  display: block;
  margin-bottom: 22px;
  max-width: 100%
}
```

- Next up, let's add images of all of the speakers to the Speakers page. We'll begin by placing images of all of the speakers into the images folder.

Within the `speakers.html` file, let's add an `` element within each of the speaker information `<aside>` elements. Let's place each `` element inside the `<div>` element with the class attribute value of `speaker-info`, just above the `` element.

The `src` attribute value of each image will correspond to the "image" folder we set up and the speaker's image filename; the `alt` attribute value will be the speaker's name.

The `<aside>` element for myself, as a speaker, will look like this:

```
<aside class="col-1-3">
  <div class="speaker-info">

    <ul>
      <li><a href="https://twitter.com/dermotk">@dermotk</a></li>
      <li><a href="http://dermotkerr.co.uk/">dermotkerr.co.uk</a></li>
```

```

    </ul>

    </div>
</aside>

```

This same pattern for adding an image should then be applied to **all other speakers**.

4. As we did with the images on our home page, we'll want to apply some styles to the images on the Speakers page.

Let's begin by applying the `border-radius` property with a value of `50%`, turning our images into circles. From there, let's set a fixed `height` of `130` pixels to each image and set them to be vertically aligned to the top of the line they reside within.

With the `height` and vertical alignment in place, let's apply vertical margins to the images. Using a negative `66`-pixel `margin` on the top of the images, we'll pull them slightly out of the `<aside>` element and make them vertically centered with the top `border` of the `<div>` element with a `class` attribute value of `speaker-info`. Then, applying a `22`-pixel `margin` on the bottom of the image provides space between the image and the `` element below it.

When we add these new styles to our existing Speakers page styles (using the `speaker-info` class as a qualifying selector for the `` elements), our CSS will look like this:

```

.speaker-info img {
  border-radius: 50%;
  height: 130px;
  margin: -66px 0 22px 0;
  vertical-align: top;
}

```

5. Since we are using an aggressive negative `margin` on the `` element within the `<div>` element with a class attribute value of `speaker-info`, we need to remove the `padding` on the top of that `<div>` element.

Previously we were using the `padding` property with a value of `22px 0`, thus placing `22` pixels of `padding` on the top and bottom and `0` pixels of `padding` on the left and right of the `<div>` element. Let's swap this property and value out for the `padding-bottom` property, as that's the only `padding` we need to identify, and use a value of `22` pixels.

The new `speaker-info` class rule set looks like this:

```

.speaker-info {

```

```
border: 1px solid #dfe2e5;
border-radius: 5px;
margin-top: 88px;
padding-bottom: 22px;
text-align: center;
}
```

Now both our home and Speaker pages are looking pretty sharp.

SCEIS
CONFERENCE

September 24–26th — Magee


HOME SPEAKERS SCHEDULE VENUE REGISTER

Dedicated to educating the best

Every year the brightest students descend on Magee to discuss the latest technologies. Join us this September!

REGISTER NOW


SPEAKERS



World-Class Speakers

Joining us from all around the world are over twenty fantastic speakers, here to share their stories.


SCHEDULE



Three Inspiring Days

Enjoy three inspiring and action-packed days of talks, gatherings, and all-round good times.

VENUE

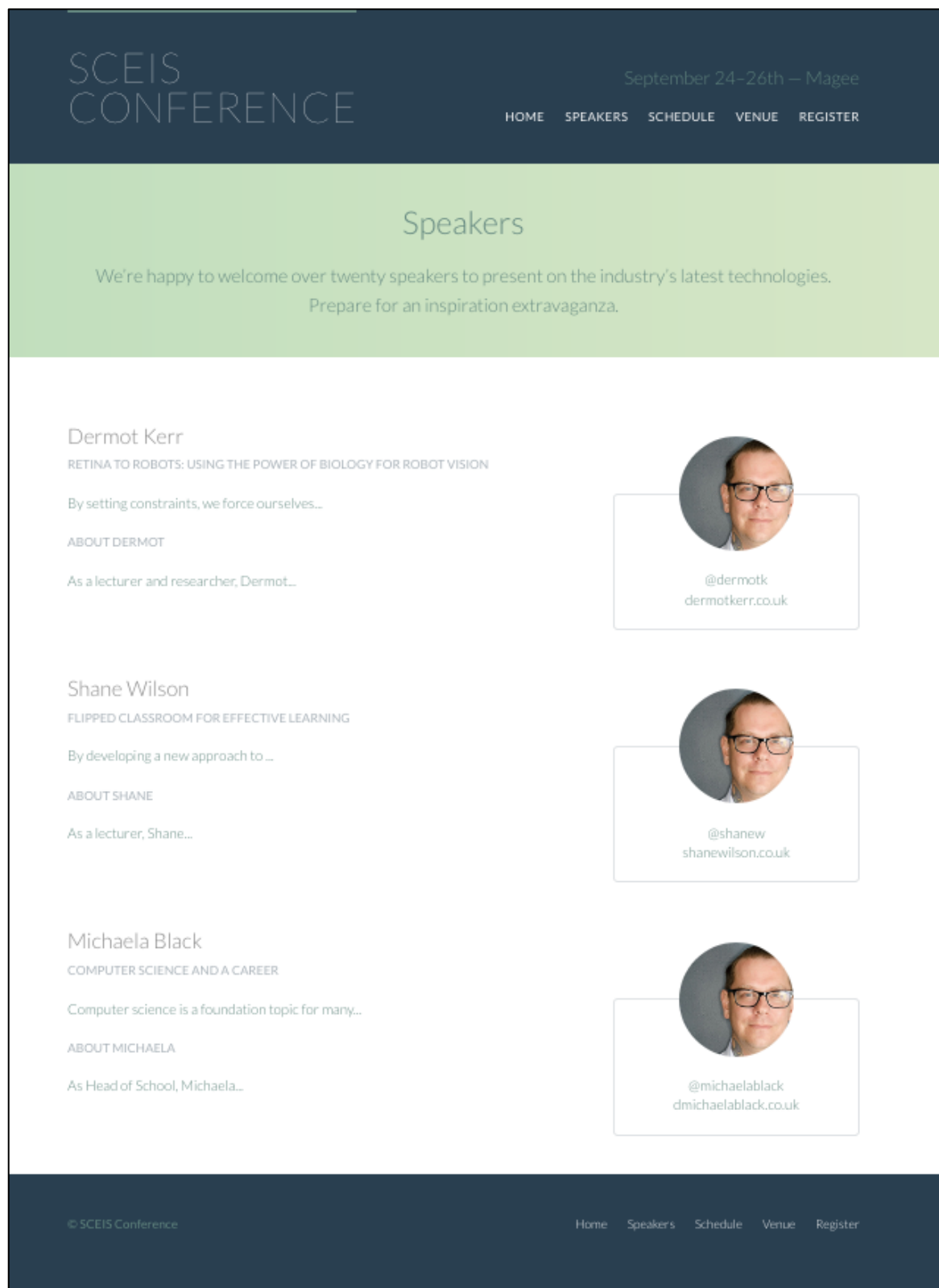


Ulster University, Magee

Beautiful city centre campus location with modern state of the art facilities.

©SCEIS Conference

Home Speakers Schedule Venue Register

Fig 9.1. SCEIS Conference home page after adding images to each section that teases another page**Fig 9.2. SCEIS Conference Speakers page after adding images for each of the speakers**

Adding Audio

HTML5 provides a quick and easy way to add audio files¹ to a website by way of the `<audio>` element². As with the `` element, the `<audio>` element accepts a source URL specified by the `src` attribute. Unlike the `` element, though, the `<audio>` element requires both opening and closing tags, which we'll discuss soon.

```
<audio src="jazz.ogg"></audio>
```

Audio Attributes

Several other attributes may accompany the `src` attribute on the `<audio>` element; the most popular include `autoplay`, `controls`, `loop`, and `preload`.

The `autoplay`, `controls`, and `loop` attributes are all Boolean attributes. As Boolean attributes, they don't require a stated value. Instead, when each is present on the `<audio>` element its value will be set to true, and the `<audio>` element will behave accordingly.

By default, the `<audio>` element isn't displayed on a page. If the `autoplay` Boolean attribute is present on the `<audio>` element, nothing will appear on the page, but the audio file will automatically play upon loading.

```
<audio src="jazz.ogg" autoplay></audio>
```

To display the `<audio>` element on a page, the `controls` Boolean attribute is necessary. When it's applied to the `<audio>` element, the `controls` Boolean attribute will display a browser's default audio controls, including play and pause, seek, and volume controls.

```
<audio src="jazz.ogg" controls></audio>
```

When present on the `<audio>` element, the `loop` Boolean attribute will cause an audio file to repeat continually, from beginning to end.

Lastly, the `preload` attribute for the `<audio>` element helps identify what, if any, information about the audio file should be loaded before the clip is played. It accepts three values: `none`, `auto`, and `metadata`. The `none` value won't preload any information about an audio file, while the `auto` value will preload all information about an audio file. The `metadata` value sits in between the `none` and `auto` values, as it will preload any available metadata information about an audio file, such as the clip's length, but not all information.

¹ https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Using_HTML5_audio_and_video

² <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio>

When the `preload` attribute isn't present on the `<audio>` element, all information about an audio file is loaded, as if the value was set to `auto`. For this reason, using the `preload` attribute with a value of `metadata` or `none` is a good idea when an audio file is not essential to a page. It'll help to conserve bandwidth and allow pages to load faster.

Audio Fallbacks & Multiple Sources

At the moment, different browsers support different audio file formats, the three most popular of which are `ogg`, `mp3`, and `wav`. For the best browser support we'll need to use a handful of audio fallbacks, which will be included inside an `<audio>` element's opening and closing tags.

To begin, we'll remove the `src` attribute from the `<audio>` element. Instead, we'll use the `<source>` element, with a `src` attribute, nested inside the `<audio>` element to define a new source.

Using a `<source>` element and `src` attribute for each file format, we can list one audio file format after the other. We'll use the `type` attribute to quickly help the browser identify which audio types are available. When a browser recognises an audio file format it will load that file and ignore all the others.

Because it was introduced in HTML5, some browsers may not support the `<audio>` element. In this case, we can provide a link to download the audio file after any `<source>` elements within the `<audio>` element.

```
<audio controls>
  <source src="jazz.ogg" type="audio/ogg">
  <source src="jazz.mp3" type="audio/mpeg">
  <source src="jazz.wav" type="audio/wav">
  Please <a href="jazz.mp3" download>download</a> the audio file.
</audio>
```

To review the previous code, the `<audio>` element includes the `controls` Boolean attribute to ensure the audio player is displayed within browsers that support the element. The `<audio>` element does not include a `src` attribute and instead wraps three different `<source>` elements. Each `<source>` element includes a `src` attribute that references a different audio file format and a `type` attribute to identify the format of the audio file. As a last fallback, if a browser doesn't recognize any of the audio file formats, the anchor link to download the element will be displayed.

In addition to the `<audio>` element, HTML5 also introduced the `<video>` element, which shares quite a few similarities with the `<audio>` element.

Adding Video

Adding video in HTML5³ is very similar to adding audio. We use the `<video>` element in place of the `<audio>` element. All of the same attributes (`src`, `autoplay`, `controls`, `loop`, and `preload`) and fallbacks apply here, too.

With the `<audio>` element, if the `controls` Boolean attribute isn't specified the audio clip isn't displayed. With videos, if the `controls` Boolean attribute is not specified the video will display. However, it is fairly difficult to view unless the `autoplay` Boolean attribute is also applied. In general, the best practice here is to include the `controls` Boolean attribute unless there is a good reason not to allow users to start, stop, or replay the video.

Since videos take up space on the page, it doesn't hurt to specify their dimensions, which is most commonly done with `width` and `height` properties in CSS. This helps ensure that the video isn't too large and stays within the implied layout of a page. Additionally, specifying a size, as with images, helps the browser render videos faster and allows it to allocate the proper space needed for the video to be displayed.

```
<video src="earth.ogv" controls></video>
```

Customizing Audio & Video Controls

By default, the `<audio>` and `<video>` element controls are determined by each browser independently. Depending on the design of a website, more authority over the look and feel of the media player may be needed. If this is the case, a customised player can be built, but it will require a little JavaScript to work.

Additionally, if a customised player uses the `` element as a control the value of the `alt` attribute should explicitly state that the image is a control and requires the proper interaction to work.

Poster Attribute

One additional attribute available for the `<video>` element is the `poster` attribute. The `poster` attribute allows us to specify an image, in the form of a URL, to be shown before a video is played. The example below uses a screen capture from the video as the poster for the Earth video.

```
<video src="earth.ogv" controls poster="earth-video-screenshot.jpg"></video>
```

³ <http://dev.opera.com/articles/introduction-html5-video/>

Video Fallbacks

As with the `<audio>` element, video fallbacks are also necessary. The same markup format, with multiple `<source>` elements for each file type and a plain text fallback, also applies within the `<video>` element.

```
<video controls>
  <source src="earth.ogv" type="video/ogg">
  <source src="earth.mp4" type="video/mp4">
  Please <a href="earth.mp4" download>download</a> the video.
</video>
```

One additional fallback option that could be used in place of a plain text fallback is to use a YouTube⁴ or Vimeo⁵ embedded video. These video hosting websites allow us to upload our videos, provide a standard video player, and enable us to embed our videos onto a page using an inline frame.

HTML5 Audio & Video File Formats

Browser support for the `<audio>` and `<video>` elements varies, as do the file formats required with these elements. Each browser has its own preferred⁶ audio and video file formats.

There are a few tools that help to convert an audio⁷ or video⁸ file into different formats, and a quick search will provide an abundance of options.

Adding Inline Frames

Another way to add content to a page is to embed another HTML page within the current page. This is done using an inline frame, or `<iframe>` element. The `<iframe>` element accepts the URL of another HTML page within the `src` attribute value; this causes the content from the embedded HTML page to be displayed on the current page. The value of the `src` attribute may be a URL relative to the page the `<iframe>` element appears on or an absolute URL for an entirely external page.

⁴ <https://www.youtube.com/>

⁵ <https://vimeo.com/>

⁶ https://developer.mozilla.org/en-US/docs/HTML/Supported_media_formats

⁷ <http://media.io/>

⁸ <http://www.mirovideoconverter.com/>

Many pages use the `<iframe>` element to embed media onto a page from an external website such as Google Maps, YouTube, and others.

```
<iframe src="https://www.google.com/maps/embed?..."></iframe>
```

The `<iframe>` element has a few default styles, including an inset `border` and a `width` and `height`. These styles can be adjusted using the `frameborder`, `width`, and `height` HTML attributes or by using the `border`, `width`, and `height` CSS properties.

Pages referenced within the `src` attribute of an `<iframe>` element play by their own rules, as they do not inherit any styles or behaviours from the page they are referenced on. Any styles applied to a page that includes an `<iframe>` element will not be inherited by the page referenced within the `<iframe>` element. Additionally, links within the page referenced within the `<iframe>` element will open inside that frame, leaving the page that contains the `<iframe>` element unchanged.

Exercise 9.2

Inline frames provide a great way to add dynamic content to a page. Let's give this a shot by updating our Venue page with some maps.

1. Before adding any maps or inline frames, let's first prepare our Venue page for a two-column grid. Below the leading section of the page we'll add a `<section>` element with the `class` attribute value of `row` to identify a new section of the page, and we'll include some general styles, such as a white `background` and some vertical `padding`.

Directly inside this `<section>` element let's add a `<div>` element with the `class` attribute value of `grid`. The class of `grid` centres our content on the page and prepares for the one-third and two-thirds columns to follow.

Now the main content section of our `venue.html` file should look like this:

```
<!-- Main content -->
<section class="row-alt">
  <div class="lead-container">
    <h1>Venue</h1>
    <p>Located in the historic Ulster University Magee campus.</p>
  </div>
</section>

<section class="row">
  <div class="grid">
    ...
```

```
</div>
</section>
```

2. Within the `<div>` element with the class attribute value of `grid` we'll have two new sections, one for the conference venue and one for the conference hotel. Let's add two new `<section>` elements and give each of these `<section>` elements a unique class that corresponds to its content. We'll use these classes to add margins to the bottom of each section.

Our HTML should now look like this:

```
<!-- Main content -->

<section class="row-alt">
  <div class="lead container">
    <h1>Venue</h1>
    <p>Located in the historic Ulster University Magee campus.</p>
  </div>
</section>

<section class="row">
  <div class="grid">

    <section class="venue-theatre">
      ...
    </section>

    <section class="venue-hotel">
      ...
    </section>

  </div>
</section>
```

3. Now that we have a few classes to work with, let's create a new section within our `main.css` file for Venue page styles. We'll add a 66-pixel `margin` to the bottom of the `<section>` element with the class attribute value of `venue-theatre` to insert some space between it and the `<section>` element below it.

Then, we'll add a 22-pixel [margin](#) to the bottom of the `<section>` element with the class attribute value of `venue-hotel` to provide some space between it and the `<footer>` element below it.

The new venue section within the `main.css` file looks like the following:

```
/*
=====
Venue
=====
*/

.venue-theatre {
  margin-bottom: 66px;
}

.venue-hotel {
  margin-bottom: 22px;
}
```

The `<section>` element with the class attribute value of `venue-hotel` has a smaller bottom margin than the `<section>` element with the class attribute value of `venue-theatre` because it sits next to the [padding](#) from the bottom of the `<section>` element with the class attribute of `row`. Adding that [margin](#) and [padding](#) together gives us the same value as the bottom [margin](#) on the `<section>` element with the class attribute value of `venue-theatre`.

- Now it's time to create the two columns within each of the new `<section>` elements. We'll start by adding a `<div>` element with a [class](#) attribute value of `col-1-3` to establish a one-third column. After it we'll add an `<iframe>` element with a [class](#) attribute value of `col-2-3` to establish a two-thirds column.

Keeping in mind that the column classes make both the `<div>` and `<iframe>` elements inline-block elements, we need to remove the empty space that will appear between them. To do so we'll open an HTML comment directly after the closing `<div>` tag, and we'll close the HTML comment immediately before the opening `<iframe>` tag.

In all, our HTML for the columns looks like this:

```
<section class="row">
  <div class="grid">

    <section class="venue-theatre">
      <div class="col-1-3"></div><!--
```

```

--><iframe class="col-2-3"></iframe>
</section>

<section class="venue-hotel">
  <div class="col-1-3"></div><!--
  --><iframe class="col-2-3"></iframe>
</section>

</div>
</section>

```

5. Within each of the `<div>` elements with a `class` attribute value of `col-1-3` let's add the venue's name within an `<h2>` element, followed by two `<p>` elements. In the first `<p>` element let's include the venue's address, and in the second `<p>` element let's include the venue's website (within an anchor link) and phone number.

Within each of the paragraphs, let's use the line-break element, `
`, to place breaks within the address and in between the website and phone number.

For the `<section>` element with the `class` attribute value of `venue-theatre`, the HTML looks like this:

```

<section class="venue-theatre">
  <div class="col-1-3">
    <h2>Magee Theatre</h2>
    <p>Nothland Road <br> BT48 7JL</p>
    <p><a href="http://www.ulster.ac.uk/">ulster.ac.uk</a> <br> 028 712345678</p>
  </div><!--
  --><iframe class="col-2-3"></iframe>
</section>

```

The same pattern shown here for the theatre should also be applied to the hotel (using, of course, the proper address, website, and phone number).

6. We can search for these addresses in Google Maps⁹. Once we locate an address and create a customized map, we have the ability to embed that map into our page. Following the instructions on Google Maps for how to share and embed a map will provide us with the HTML for an `<iframe>` element.

⁹ <https://www.google.com/maps/>

Let's copy the HTML—`<iframe>` element, `src` attribute, and all—onto our page where our existing `<iframe>` element resides. We'll do this for each location, using two different `<iframe>` elements.

In copying over the `<iframe>` element from Google Maps we need to make sure we preserve the `class` attribute and value, `col-2-3`, from our existing `<iframe>` element. We also need to be careful not to harm the HTML comment that closes directly before our opening `<iframe>` tag.

Looking directly at the `<section>` element with the `class` attribute value of `venue-theatre` again, the HTML looks like this:

```
<section class="venue-theatre">
  <div class="col-1-3">
    <h2>Magee Theatre</h2>
    <p>Nothland Road <br> BT48 7JL</p>
    <p><a href="http://www.ulster.ac.uk/">ulster.ac.uk</a> <br> 028 712345678</p>
  </div><!--

--><iframe class="col-2-3"
src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d2288.1098604015365!2d-
7.325799383897053!3d55.006239256787374!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x485fe22a
6037f471%3A0x5308c53140e88647!2sUlster+University+Magee+Campus!5e0!3m2!1sen!2suk!4v154176544175
9">
</iframe>
</section>
```

7. Lastly, we'll want to make sure that both `<iframe>` elements that reference Google Maps share the same height. To do this, we'll create a new class, `venue-map`, and apply it to each of the `<iframe>` elements alongside the existing `col-2-3` class attribute value.

The HTML for the `<section>` element with the `class` attribute value of `venue-theatre` now looks like this:

```
--><iframe class="venue-map col-2-3"
src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d2288.1098604015365!2d-
7.325799383897053!3d55.006239256787374!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x485fe22a
```

```
6037f471%3A0x5308c53140e88647!2sUlster+University+Magee+Campus!5e0!3m2!1sen!2suk!4v154176544175
9">
</iframe>
</section>
```

Once the `venue-map` class is applied to each `<iframe>` element, let's create the `venue-map` class rule set within our `main.css` file. It includes the `height` property with a value of `264` pixels.

The `venue-map` class rule set looks like this:

```
.venue-map {
  height: 264px;
}
```

We now have a Venue page, complete with maps for the different locations of our conference.

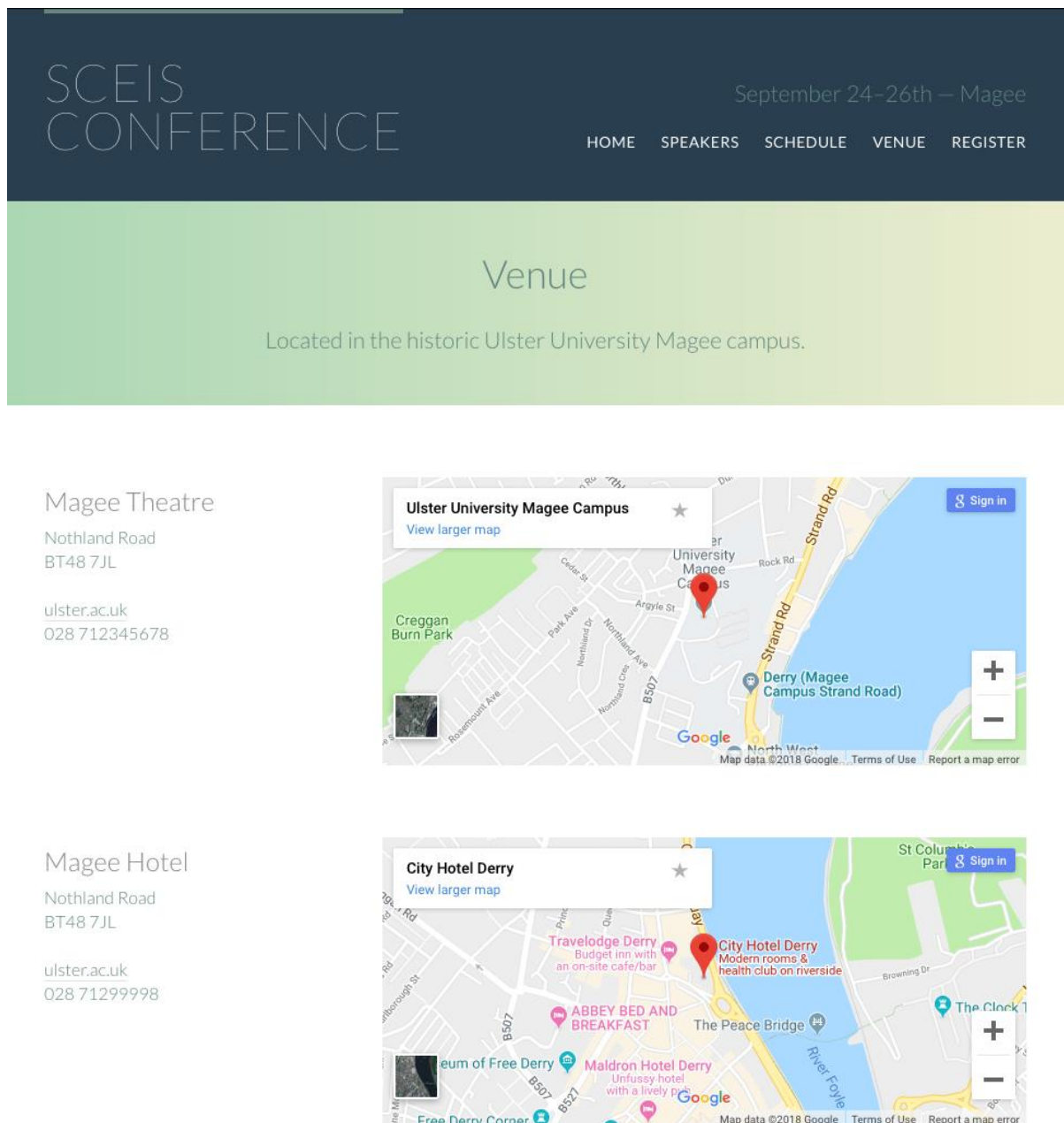


Fig 9.4 Our SCEIS Conference Venue page, which now includes inline frames

Semantically Identifying Figures & Captions

With HTML5 also came the introduction of the `<figure>` and `<figcaption>` elements. These elements¹⁰ were created to semantically mark up self-contained content or media, commonly with a caption. Before HTML5 this was frequently done using an ordered list. While an ordered list worked, the markup was not semantically correct.

¹⁰ <http://html5doctor.com/the-figure-figcaption-elements/>

Figure

The `<figure>` block-level element is used to identify and wrap self-contained content, often in the form of media. It may surround images, audio clips, videos, blocks of code, diagrams, illustrations, or other self-contained media. More than one item of self-contained content, such as multiple images or videos, may be contained within the `<figure>` element at a time. If the `<figure>` element is moved from the main portion of a page to another location (for example, the bottom of the page), it should not disrupt the content or legibility of the page.

```
<figure>
  
</figure>
```

Figure Caption

To add a caption or legend to the `<figure>` element, the `<figcaption>` element is used. The `<figcaption>` may appear at the top of, bottom of, or anywhere within the `<figure>` element; however, it may only appear once. When it's used, the `<figcaption>` element will serve as the caption for all content within the `<figure>` element.

Additionally, the `<figcaption>` element may replace an `` element's `alt` attribute if the content of the `<figcaption>` element provides a useful description of the visual content of the image.

```
<figure>
  
  <figcaption>A beautiful black, brown, and white hound dog.</figcaption>
</figure>
```

Not all forms of media need to be included within a `<figure>` element or include a `<figcaption>` element; only those that are self-contained and belong together as a group.

Summary

Alongside text, media is one of the largest parts of the web. Use of images, audio, and video has only grown over recent years, and it isn't likely to slow down. Now we know how to incorporate these forms of media into our designs and how we can use them to enrich the content on our websites.

Within this lesson we covered the following:

- The best ways to add images, audio clips, videos, and inline frames to a page
- Different ways to position images in different situations

- How to provide audio and video fallbacks for older browsers
- Common attributes available to audio clips and videos
- The semantic way to mark up self-contained content, including media

We're coming into the homestretch of learning HTML and CSS, with only a few more components left to introduce.

Resources & Links

Using HTML5 audio and video via Mozilla Developer Network

https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Using_HTML5_audio_and_video

Audio HTML5 Element via Mozilla Developer Network

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio>

Introduction to HTML5 Video via Dev.Opera

<http://dev.opera.com/articles/introduction-html5-video/>

The figure & figcaption elements via HTML5 Doctor

<http://html5doctor.com/the-figure-figcaption-elements/>