HTML5 & CSS3

For Beginners

The tactical guide book

HTML5 & CSS - The Tactical Guide Book

You can reach the author at: info@csharpschool.com.

Source code

The source code accompanying this book is shared under the MIT License and can be downloaded here after registering with the site http://www.csharpschool.com using the code **html5** or by emailing the author.

Contents

About the book

Disclaimer - Who is this book for?

It's important to mention that this book is not meant to be a *get-to-know-it-all* book it's more on the practical and tactical side where you will learn as you progress through the examples and build a real website in the process. Because I personally dislike having to read hundreds upon hundreds of pages of irrelevant fluff (filler material) not relevant to the tasks at hand and view it as a disservice to the readers, I will assume that we are of a same mind on this and will therefore only include important information pertinent for the tasks at hand making the book shorter and more condensed saving you time and effort in the process. Don't get me wrong, I will describe the important things in great detail, leaving out only the things that are not directly relevant to your first experience with HTML5 and CSS3. The goal is for you to have created a working HTML5 website upon finishing this book. You can always look into details at a later time when you have a few projects under your belt. ***If you prefer encyclopedic books describing everything in minute detail with short examples and value a book by how many pages it has rather than its content, then this book is NOT for you***.

About the author

Jonas started a company back in 1994 focusing on education in Microsoft Office and the Microsoft operating systems. While still studying at the university in 1995, he wrote his first book about Widows 95 as well as a number of course materials.

In the year 2000, after working as a Microsoft Office developer consultant for a couple of years, he wrote his second book about Visual Basic 6.0.

Between 2000 and 2004 he worked as a Microsoft instructor with two of the largest educational companies in Sweden. First teaching Visual Basic 6.0, and when Visual Basic.NET and C# were released he started teaching these languages as well as the .NET Framework. Teaching classes on all levels for beginner to advanced developers.

From the year 2005, Jonas shifted his career towards consulting once again, working hands on with the languages and framework he taught.

Jonas wrote his third book *C# programming* aimed at beginners to intermediate developers in 2013 and now in 2015 his fourth book *C# for beginners - The Tactical Guide* was published.

1. Introduction To HTML5

Introduction

This course is primarily aimed at developers who are new to HTML5 and CSS3. Even if you already have created a couple of HTML5 web pages you might find the content in this book useful as a refresher. The content is delivered in a no-fluff way to get you up to speed with HTML5 and CSS3 as fast as possible.

The examples in this book are presented using Visual Studio 2015 but they will work with any text editor. The free express version of Visual Studio should do fine when you follow along and implement them yourself.

At the end of this chapter you will have an understanding of the most commonly used HTML5 elements and how to add them to a web page.

Two groups

There are two groups who are responsible for the development of HTML5. The World Wide Web Consortium (W3C) is the main international standards organization for the World Wide Web (www.w3.org) *and* The Web Hypertext Application Technology Working Group, or WHATWG, is a community of people interested in evolving HTML and related technologies (www.whatwg.org).

In previous versions of HTML plug-ins were used to add functionality to a page at the cost of it often breaking in some browsers. With HTML5 plug-ins like Flash are no longer needed because there are HTML5 elements that do the same thing like the **<video>** element.

HTML

HTML stands for Hyper Text Markup Language and it is an angel bracket based structure that allows you to define different parts of a page where some are visible to the user and some are not.

The following example shows how angle brackets are used to define HTML elements which make up the content of a page. As you can see the code is made up of a hierarchical element structure which always begin with an **<html>** opening tag and ends with the **</html>** closing tag. The **<!DOCTYPE html>** element signals to the document parser that HTML5 is being used to build the page; this element has been significantly trimmed down compared with older versions of it.

```
<!DOCTYPE html>
<html>
   <head>
      <title></title>
   </head>
   <body>
      <!--This is a comment-->
      <h1>This is a heading</h1>
      <p>This is a paragraph.</p>
   </body>
</html>
```

The section between the **<head>** element and its closing tag is often used to add meta data about the page, a document title which is displayed in the browser tab and to link in script and CSS files needed in the document.

The **<body>** element and its closing tag defines where the actual document content begins and ends. As you can see there are three pieces of content inside the **<body>** element.

The first line is a comment which won't be rendered to the browser; you can use comments to describe the purpose of a piece of markup or to temporarily comment out a block of markup.

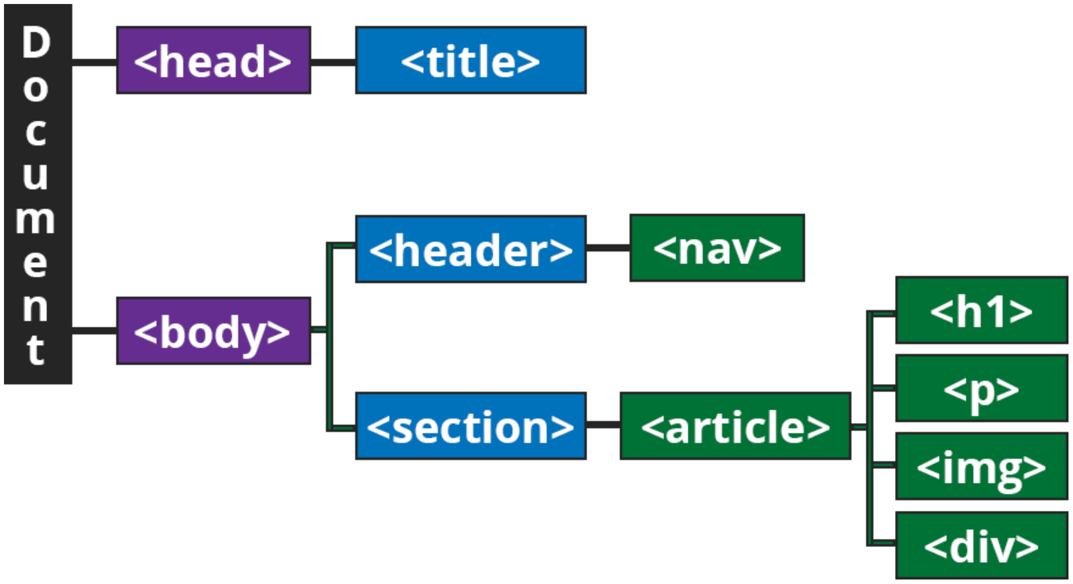
The second line is a heading which will be rendered as text with a larger font size and the third line defines a paragraph, a line of text which is displayed on a new line.

The HTML markup is not case sensitive although most coders use lowercase letters for the element names within the angle brackets.

There are mixed closing rules for elements where some elements, like **<img />** and **<br />**, always are self closing and not allowed to contain any child elements. Other elements, like **<div></div>**, must have an opening and closing tag defining a block where you can place child elements.

The DOM

HTML relies on a Document Object Model (DOM) which basically defines the hierarchy of the elements that make up the page and builds an object graph of the elements at run-time. The graph represent the objects that the user can interact with as well as a set of objects that you as a developer can interact with.



HTML Elements

Before we jump into the wonderful new semantic HTML5 elements it is important to list a few of the older elements that are widely used. You will use some of these later in the book to learn how to position elements on the web page.

Element	Description
<a>	Defines a clickable hyperlink. Can contain other elements in HTML5. Use the **herf** attribute to specify the destination URL.
<address>	Defines contact information for the author/owner of a document. Can contain other elements.

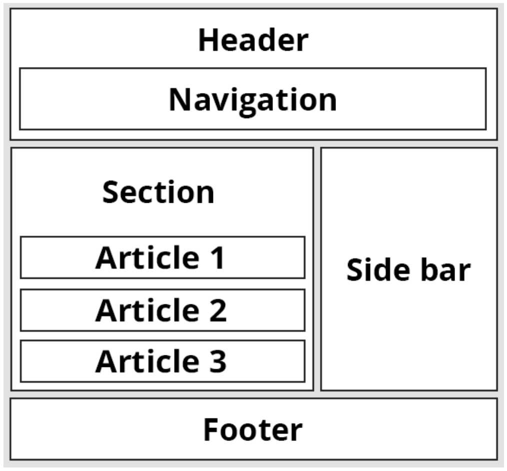
Element	Description
<b>	Defines bold text.
<body>	Defines the document's body.
<br>	Defines a line break.
<button>	Defines a clickable button. Could also be achieved with an **<input>** element.
<dl>, <dd>, <dt>	Defines a description list of value pairs where **<dl>** is the list container, **<dd>** contain the description and **<dt>** contain the term/name.
<div>	Defines a section which begin on a new line in the document.
<form>	Defines an HTML form for user input. Can be used to send data to the server synchronously or asynchronously using Ajax.
<h1> - <h6>	Defines HTML headings.
<head>	Defines information about the document.
<hr>	Displays a horizontal line as a thematic change in the content.
<img>	Displays an image. Use the **src** attribute to assign the image source and **alt** to provide a description of the image (can be used for SEO purposes and is displayed if the image for some reason can't be displayed).
<label>	Defines a label for an input element. Often used when the elements are part of a form.
<li>	Defines a list item in an ordered **<ol>** or unordered **<ul>** list.
<meta>	Defines metadata about the HTML document.
<ol>	Defines an ordered list where the bullets are displayed as a numbered list.
<ul>	Defines an unordered list where the list items are displayed as bullet points.
<p>	Defines a paragraph of text where the text will begin on a new line.

Element	Description
<script>	Defines an area in the HTML document where script code can be declared. You normally don't use this element because script code should be linked in from a script file.
<select>	Defines a drop-down list.
<span>	Defines an in-line section of the document. One common use is to place **<span>** elements in a **<div>** element and let them flow inside the container placing the **<span>** elements side by side.
<strong>	Denotes important text.
<style>	Defines an area in the HTML document where CSS rules can be declared. You normally don't use this element because CSS styling should be placed in a separate file and be linked in to the document.
<table>	Defines a table in the HTML document. Is used in conjunction with the **<thead>**, **<tbody>**, **<tfoot>**, **<td>**, **<th>**, **<tr>** elements.
<u>	Underlines text.

NOTE: *You can read more about HTML elements at http://www.w3schools.com/tags.*

Exercise 1: The basic HTML structure

In this exercise commonly used HTML elements will be used to build a web page. The image below shows the different sections the page should contain.



The page skeleton

Let's start by adding the basic page structure with HTML. When creating a new HTML page in Visual Studio the **<!DOCTYPE>**, **<html>**, **<head>** and **<body>** elements are added by default to give you a starting point.

```
<!DOCTYPE html>
<html>
   <head>
      <title></title>
      <meta charset="utf-8" />
      <!--Add links to external files here-->
   </head>
   <body>
      <!--Add your HTML markup here-->
   </body>
</html>
```

The first section you will add is the **header** section which usually contain a logo and navigation, a **<div>** is usually used when adding sections with HTML because they can contain other elements and begin on a new line. Looking at the code above you can see that the HTML you write goes between the **<body>** and **</body>** tags. To illustrate the purpose of the **<div>** you can add a CSS class name to it which later can be used to style the element.

```
<body>
   <div class="header">
    </div>
</body>
```

The page navigation is a created as a child section to the **header** section. To create this hierarchy you must add a new **<div>** inside the Header **<div>** and decorate it with the a CSS class denoting that it is meant for **navigation**.

```
<body>
   <div class="header">
      <div class="navigation">
      </div>
   </div>
</body>
```

The **section** element lives outside the **header** element and should therefore not be added inside the **header** element but below it. As you can see it contains three **article** sections which you add inside the **section** element.

```
<body>
    <div class="header">
        <div class="navigation">
        </div>
    </div>
    <div class="section">
        <div class="article">
        </div>
        <div class="article">
        </div>
        <div class="article">
        </div>
    </div>
</body>
```

The **side** bar section should be placed to the right of the **section**, this is however not possible to achieve with HTML alone; to achieve this result you have to style the element with CSS. The side bar **<div>** therefore has to be placed either above or below the **section** element in the hierarchical structure.

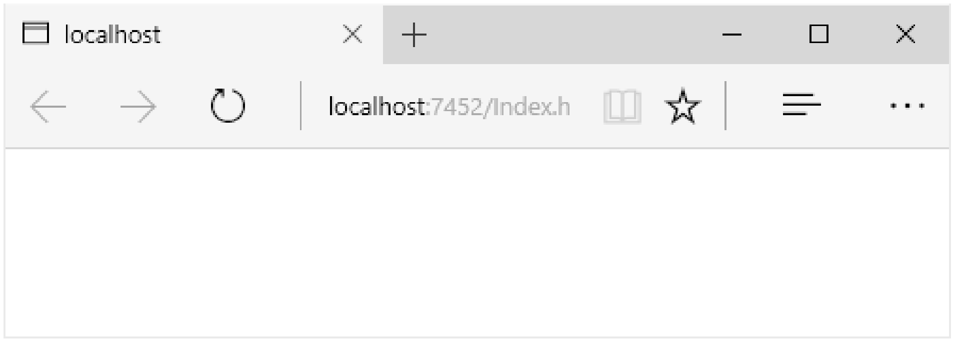
```
<body>
    <div class="header">
        <div class="navigation">
        </div>
    </div>
    <div class="section">
        <div class="article">
        </div>
        <div class="article">
        </div>
        <div class="article">
        </div>
    </div>
    <div class="side-bar">
    </div>
</body>
```

The last section is the **footer** section where you normally find a copyright notice. The finished skeleton structure should look like this:

```
<body>
   <div class="header">
      <div class="navigation">
      </div>
   </div>
   <div class="section">
      <div class="article">
      </div>
      <div class="article">
      </div>
      <div class="article">
      </div>
   </div>
   <div class="side-bar">
   </div>
   <div class="footer">
   </div>
</body>
```

Before you run the application the **Index** page you are working with has to be set as the start page by right clicking on it in the solution explorer and selecting **Set As Start Page** in the menu. If you are using a text editor to implement the code you simply right click on the file, point to **Open with** in the menu and select the browser of your choice.

As you can see nothing is displayed in the browser window when you view the page. The reason is that the elements used so far don't render as content themselves if they are un-styled, they only act as containers for other HTML elements.



In the next section you will alter the HTML markup to HTML5 markup.

The **<div>** element is used to place content that belong together in a container. This gives you a huge benefit when styling the content with CSS. You can now move all the elements in that container by moving the **<div>**. You can also style or place the elements in the **<div>** in respect to it.

HTML5 Elements

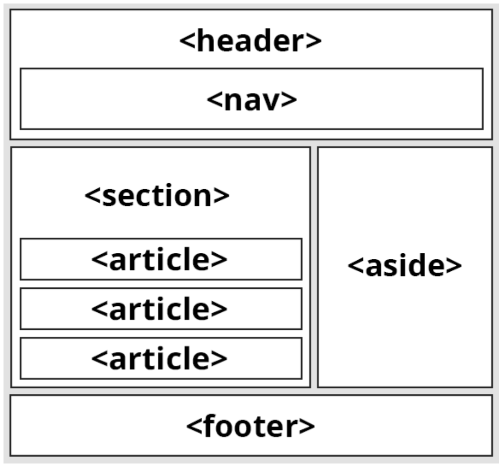
In this section you will learn about the most commonly used HTML5 elements (tags) when building a web page. Some elements like **<div>**, **<span>** and **<a>** have been around for a long time but there are a few new elements which have a more semantic meaning like **<article>**, **<video>** and **<aside>**; most of them behave like a **<div>** and they describe their purpose.

"… *semantics is the subfield that is devoted to the study of meaning, as inherent at the levels of words, phrases, sentences …*" - Wikipedia

How do semantics apply to HTML5? Well, if you look at older elements like **<div>** and **<span>** they don't convey what they are meant to be used for, they could be used as a container for anything such as other elements or text. When semantics is put into the mix the result is elements that actually have a meaning, they are meant to be used for specific purposes. Take the **<video>** element for instance, there is no doubt when to use it; it has a semantic meaning.

So, should you completely disregard the "old" elements and only use the "new"? Of course not. **<div>** and **<span>** elements for instance are still highly relevant when you want to position elements relative to one another.

The image below show where on the page some of the new elements usually are used although they can be used in other scenarios as well. The **<header>** and **<footer>** elements can for instance be used as the page header and footer or as a header and footer inside a **<section>** or **<article>** element. The **<nav>** element is almost always only used for the main navigation area and should not be used for other navigation. The aside can be used on either side of the main content to show loosely related content. The **<article>** element is meant to be used for content that can stand on its own, like an newspaper article or a blog post while the **section** is meant to section the page into content areas. There is a gray area regarding **<article>** and **<section>** elements where you see **<section>** elements in **<article>** elements and vice versa.



Element	Description

<article>	Specifies independent, self-contained content and can contain other elements. It should make sense on its own like a forum post, blog post or a newspaper article. If the article is long you could use the **<section>** element within the **<article>** element to divide the content into sections.
	`<article>`
	`   <h1>What is a UFO?</h1>`
	`   <p>It's an unidentified flying object.</p>`
	`</article>`

<aside>	Defines content that loosely belong to the content but shouldn't be directly part of it. An example could be a short biography about the article author or a list of related links.
	`<p>HTML For Beginners is a book ... </p>`
	`<p>Lorem ipsum dolor sit amet, in non morbi  ... </p>`
	`<p>Est wisi, ut risus duis lacus porttitor, bibendum</p>`
	`<aside>`
	`   <h4>HTML</h4>`
	`   <p>Stands for Hyper Text Markup Language and is the`
	`      primary markup language used to write content on`
	`      the web.</p>`
	`</aside>`

Element	Description
<audio>	Defines audio content. Add multiple sources for different audio types with the **<source>** element and a descriptive text which will be displayed if the browser don't support the **<audio>** element.
	`<audio controls>`
	`   <source src="horse.ogg" type="audio/ogg">`
	`   <source src="horse.mp3" type="audio/mpeg">`
	`   Your browser does not support the audio tag.`
	`</audio>`
<bdi>	This element is useful when embedding user-generated content with an unknown directionality. Can be used to format text displayed in a different direction from the surrounding text, for instance when English and Arabic are mixed.
	`<ul>`
	`   <li>User <bdi>Jane Doe</bdi>: 20 points</li>`
	`   <li>User <bdi>John Doe</bdi>: 30 points</li>`
	`   <li>User <bdi>إيان</bdi>: 60 points</li>`
	`</ul>`
<canvas>	An area where graphics can be drawn on the fly using a script language such as JavaScript. The example code below would draw a red rectangle inside the **<canvas>** element.
	`<canvas id="theCanvas"></canvas>`
	`<script>`
	`   var canvas = document.getElementById("theCanvas");`
	`   var ctx = canvas.getContext("2d");`
	`   ctx.fillStyle = "#FF0000";`
	`   ctx.fillRect(10, 100, 80, 80);`
	`</script>`
<datalist>	Displays a pre-defined list of choices for an **<input>** element providing auto-complete functionality.
	`<input list="browsers">`
	`<datalist id="browsers">`
	`   <option value="Internet Explorer">`
	`   <option value="Firefox">`
	`   <option value="Chrome">`
	`</datalist>`

Element	Description
<details>	Displays data that the user can show and hide by clicking on the text defined by the **<summary>** element. **NOTE:** *That this feature only is implemented in the Opera, Chrome and Safari browsers.* `<details>` `<summary>A book title</summary>` `<p>Synopsis of the book</p>` `<p>Author information</p>` `</details>`
<embed>	Defines a container for an external non-HTML application or interactive content (plug-in). `<embed src="plug-in.swf">`
<figure>, <figcaption>	Can be used to display an image (or other content) with a description. `<figure>` `<img src="mountain.jpg" alt="The Rocky Mountains" width="304" height="228">` `<figcaption>Fig1. - A scenic view of the Rocky Mountains.</figcaption>` `</figure>`
<header>	Defines a header for a document section. **NOTE:** *This element can be used several times in the same document defining headers for different* ***<section>*** *or* ***<article>*** *elements.* `<article>` `<header>` `<h1>Main heading</h1>` `<h3>Sub-heading</h3>` `<p>Some additional header info here</p>` `</header>` `<p>Lorem Ipsum dolor set amet....</p>` `</article>`
<main>	Specifies the main content of a document and can only added once to a document. The content inside the **<main>** element should not be repeated across documents such as sidebars, navigation links, copyright information, site logos, and search forms. **NOTE:** *The* ***<main>*** *element must NOT be a descendant of an* ***<article>****,* ***<aside>****,* ***<footer>****,* ***<header>****, or* ***<nav>*** *element.*

Element	Description
	`<main>` `<h1>Film list</h1>` `<p>A list of highly ranked films on IMDB</p>` `<article>` `<h1>The Godfather</h1>` `<p>Plot description ...</p>` `</article>` `<article>` `<h1>Aliens</h1>` `<p>Plot description</p>` `</article>` `</main>`
<mark>	Marks/highlights the text inside the **<mark>** element. `<p>It's a <mark>lovely</mark> sunny morning.</p>` *Output:* It's a lovely sunny morning.
<meter>	Defines a gauge for a scalar measurement within a known range, or a fractional value. Should be used for scenarios such as displaying disk usage or the relevance of a query result. **Note:** *It should not be used to indicate progress; use the* ***<progress>*** *element for progress bars.* **Note:** *The meter tag is not supported by Internet Explorer or Safari 5 (and earlier versions).* `<meter value="2" min="0" max="10">2 out of 10</meter>` `<!--or-->` `<meter value="0.6">60%</meter>`
<nav>	Is a container for MAJOR navigation links such as the application's menu bar; it is not meant for sidebar links. `<nav>` `<a href="/home/">Home</a> |` `<a href="/about/">About</a> |` `<a href="/contact/">Contact</a>` `</nav>`

Element	Description

<output>	Represent the result of a calculation (usually performed by a script).
	NOTE: *Is not supported by Internet Explorer.*
	`<form`
	`oninput="x.value=parseInt(a.value)+parseInt(b.value)">`
	`  0<input type="range" id="a" value="50">100`
	`  +<input type="number" id="b" value="50">`
	`  =<output name="x" for="a b"></output>`
	`</form>`

<progress>	Represent how far a task has progressed. `<progress value="22" max="100"></progress>`

<section>	Defines sections of an HTML document, such as chapters, headers, footers, or any other sections of the document. It can also be nested in other elements, such as **<section>** and **<article>**.
	`<section>`
	`  <h1>Section header</h1>`
	`  <p>Section content</p>`
	`</section>`

<time>	Denotes that the content should be regarded as a human-readable time/date.
	NOTE: *This element can also be used to encode dates and times in a machine-readable way so that user agents can offer to add birthday reminders or scheduled events to the user's calendar, and search engines can produce smarter search results.*
	`<!--human-readable date/time-->`
	`<p>The store open at <time>10:00</time>.</p>`
	`<!--machine-readable date/time-->`
	`<p>Family gathering on <time datetime="2015-12-25`
	`20:00"> Christmas day</time>.</p>`

<video>	Defines a video player for streaming a video or movie clip. Use **<source>** elements to specify the video sources.
	`<video width="320" height="240" controls>`
	`  <source src="movie.mp4" type="video/mp4">`
	`  <source src="movie.ogg" type="video/ogg">`
	`  Your browser does not support the video tag.`
	`</video>`

Element	Description
<wbr>	Specifies where it would be ok to add a line-break in a text. In certain texts, such as legal documents, line-breaks should be at certain places to conform to the norm. Adding **<wbr>** elements will tell the browser where line-breaks are allowed to be added. `<p>To learn AJAX, you must be familiar with the XML<wbr>HttpRequest Object.</p>`

NOTE: *Since there is no definite directive about whether an **<article>** should be placed in a **<section>** or the other way around you are free to implement it as the situation demands.*

Exercise 2: The basic HTML5 structure

In this exercise you will alter the HTML elements used in the previous exercise to semantic HTML5 elements. You will see that the markup will be much cleaner and easier to understand.

The **<div>** with the **header** class can be replaced with the **<header>** element.

```
<body>
    <header></header>
</body>
```

The **<div>** with the **navigation** class can be replaced with the **<nav>** element.

```
<body>
    <header>
        <nav></nav>
    </header>
</body>
```

The **<div>** with the **section** class can be replaced with the **<section>** element.

```
<body>
    <header>
        <nav></nav>
    </header>
    <section></section>
</body>
```

The **<div>** elements with the **article** class can be replaced with the **<article>** element.

```
<body>
    <header>
        <nav></nav>
    </header>
    <section>
        <article></article>
        <article></article>
        <article></article>
    </section>
</body>
```

The **<div>** with the **side-bar** class can be replaced with the **<aside>** element.

```
<body>
    <header>
        <nav></nav>
    </header>
    <section>
        <article></article>
        <article></article>
        <article></article>
    </section>
    <aside></aside>
</body>
```

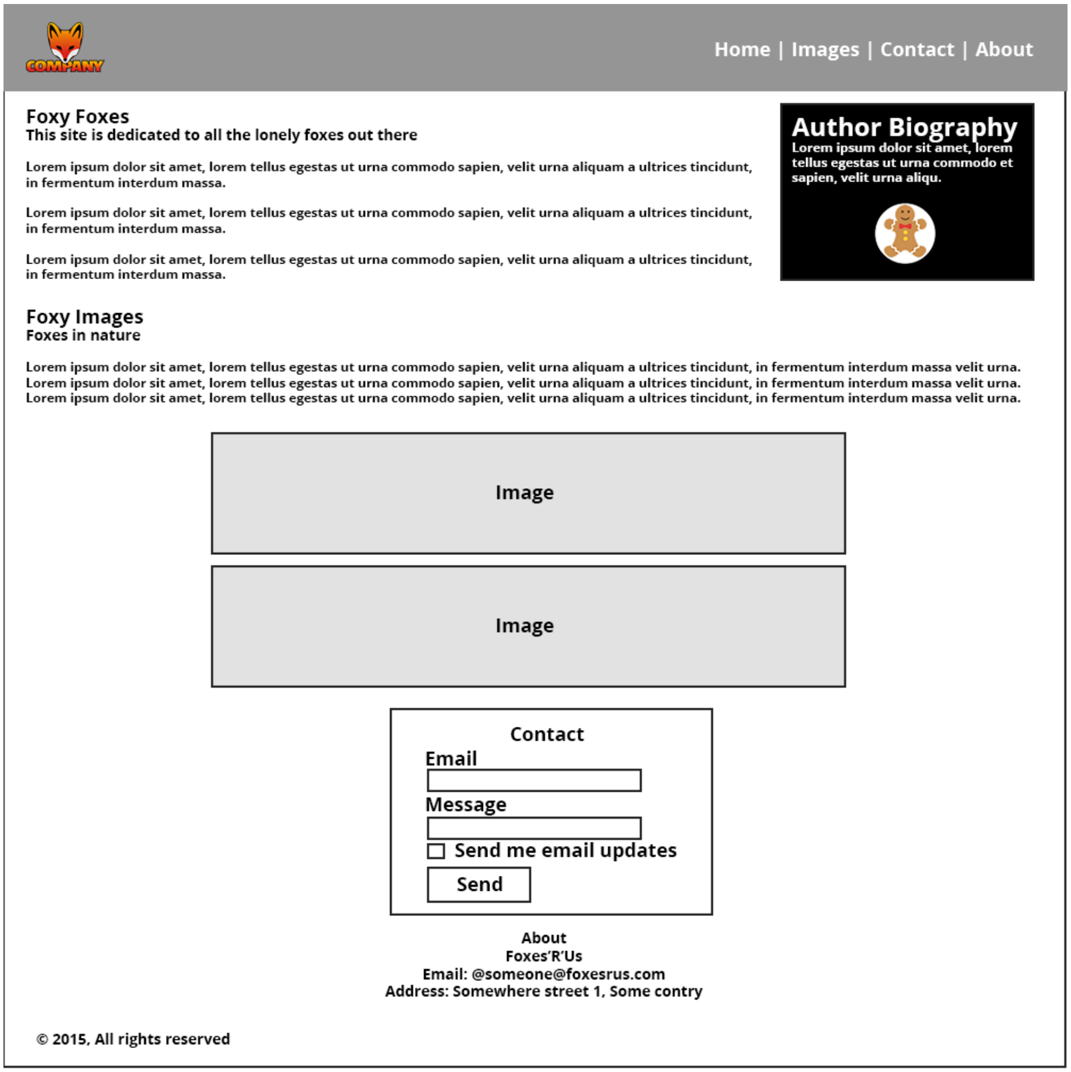
The **<div>** with the **footer** class can be replaced with the **<footer>** element.

```
<body>
    <header>
        <nav></nav>
    </header>
    <section>
        <article></article>
        <article></article>
        <article></article>
    </section>
    <aside></aside>
    <footer></footer>
</body>
```

Exercise 3: Building the Foxy Foxes web page

In this exercise you use semantic HTML5 elements to create the basic structure of the Foxy Foxes web page. It will not look pretty with just the HTML but in the next chapter you will use CSS to style the web page.

The image below illustrate the layout end result of the web page.



Home | Images | Contact | About

Foxy Foxes
This site is dedicated to all the lonely foxes out there

Lorem ipsum dolor sit amet, lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum massa.

Lorem ipsum dolor sit amet, lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum massa.

Lorem ipsum dolor sit amet, lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum massa.

Author Biography
Lorem ipsum dolor sit amet, lorem tellus egestas ut urna commodo et sapien, velit urna aliqu.

Foxy Images
Foxes in nature

Lorem ipsum dolor sit amet, lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum massa velit urna.
Lorem ipsum dolor sit amet, lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum massa velit urna.
Lorem ipsum dolor sit amet, lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum massa velit urna.

Image

Image

Contact
Email
Message
☐ Send me email updates
Send

About
Foxes'R'Us
Email: @someone@foxesrus.com
Address: Somewhere street 1, Some contry

© 2015, All rights reserved

Use *Lorem Ipsum* text from a site like http://www.blindtextgenerator.com/lorem-ipsum to populate the paragraphs.

The web page consist of a header with a logo and navigation links in the form of anchor tags (**<a>**) in an unordered list (**<ul>**) where the links navigate to the *Home*, *Images*, *Contact*, *About* and *Footer* sections.

Below the header are a number of sections dedicated to different areas of the web page. The first section represent the *Home* area with an **<article>** element containing headings and a riveting description of the web page and an **<aside>** element describing the site author.

The second section represent the *Images* area and it contains some interesting text about foxes followed by a couple of **<articles>** with images and their descriptions.

The third section represent the *Contact* area with an input form where the user can submit data; it should contain a textbox, a text area, a checkbox, a submit button and labels describing the elements. The form will not be connected to a web server or a data-base.

The fourth section represent the *About* area and should contain contact information, such as the company name, an email and a physical address.

The fifth section represent the *Footer* area where a copyright notice should be provided.

All images should be placed in a new folder called **Content**.

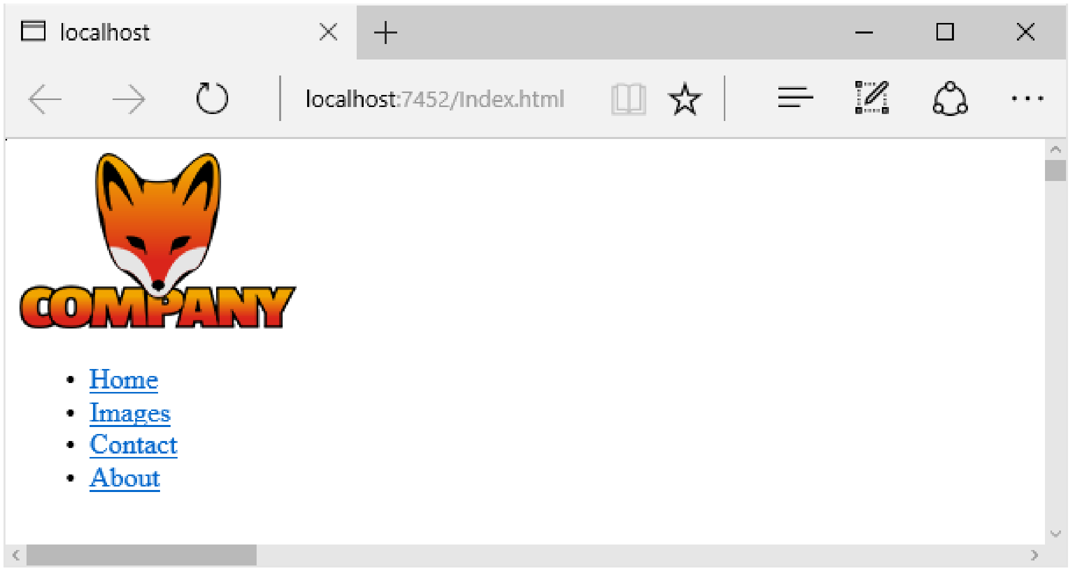
Creating the solution

1. Open Visual Studio and select **File-New-Project** in the menu.
2. Select **ASP.NET Web Application** in the template list, give the project a suitable name and click the **OK** button.
3. Select the **Empty** template (don't make any other choices in the dialog) and click the **OK** button.
4. Right click on the project name and select **Add-HTML Page**.
5. Name the page **Index** and click the **OK** button.

6. Right click on the **Index.html** file in the solution explorer and select **Set As Start Page**; this will automatically display the page when the application is started.
7. Right click on the project name in the Solution Explorer and select **Add-New Folder** to add a folder called **Content**. This folder will be used to store images and CSS files later on.

Adding the header area

Add a **<header>** element to the **<body>** element. The header should contain a logotype image with a height of 100px and a navigation area with an unordered list. The list should contain four links representing the main sections of the page *Home*, *Images*, *Contact* and *About*. The links don't have to navigate anywhere right now, you will fix the links at the end of the exercise. No link has to be provided for the *Footer* section.



1. Add a **<header>** element inside the **<body>** element in the **Index.html** document.
2. Open Windows Explorer and drag the logotype image to the **Content** folder to add it to the project. You can use any image with the dimensions 100x100px.
3. Drag the image from the Solution Explorer and place it inside the **<header>** element.

4. Add a **<nav>** element inside the **<header>** element below the image you just added.
5. Add an unordered list (**<ul>**) with four list items (**<li>**) and add link elements (**<a>**) with the descriptions "*Home*", "*Images*", "*Contact*" and "*About*" to the list items.
6. Assign **#home**, **#images**, **#contact** and **#about** to their respective **href** attributes. This will enable navigation when the other sections have been added.
   ```
   <li><a href="#home">Home</a></li>
   ```
7. Save the **Index.html** document.

The markup for the **header**:

```
<body>
    <header>
        <img src="Content/fox-logo 100px.png" />
        <nav>
            <ul>
                <li><a href="#home">Home</a></li>
                <li><a href="#images">Images</a></li>
                <li><a href="#contact">Contact</a></li>
                <li><a href="#about">About</a></li>
            </ul>
        </nav>
    </header>
</body>
```

Adding the home section

Add a **<section>** element to the **<header>** element with an id of **home** (it should match the **href** of the first **<a>** tag without the hash tag).

Add an **<article>** element in the **<section>** element and place an **<h1>** element with the text "*Foxy Foxes*" and an **<h2>** element with the text "*This site is dedicated to all the lonely foxes out there*" inside it. Also add a couple of paragraphs (**<p>**) with *lorem ipsum* text (roughly 200 characters each) below the **<h2>** element.

Add an **<aside>** element below the **<article>** element inside the **<section>** element which contain a **<header>** element containing an **<h1>** element with the text "*Author Biography*", a **<section>** containing a paragraph of *lorem ipsum* text and a 96x96px image below the inner **<section>**. In the next chapter the **<aside>** will be styled with CSS to be displayed to the right of the text in the **<article>** element.

1. Add a **<section>** element below the **<header>** element with the id **home**.
2. Add an **<article>** element inside the **<section>** element.
3. Add an **<h1>** with the text "*Foxy Foxes*" inside the **<article>** element.
4. And an **<h2>** element with the text "*This site is dedicated to all the lonely foxes out there*" below the **<h1>** element.
5. Add a few paragraphs (**<p>**) elements with a lot of *lorem ipsum* text (roughly 200 characters each). The idea is to add some filler text that fills the page vertically.
6. Add an **<aside>** element below the **<article>** element.
7. Add a **<header>** element inside the **<aside>** element.
8. Add an **<h1>** with the text "*Author Biography*" inside the **<header>** element.
9. Add a **<section>** element below the **<header>** element.
10. Add a paragraph of *lorem ipsum* text inside the **<section>** element.
11. If you haven't already added a 96x96px image to the **Content** folder then do so before continuing.
12. And the 96x96px image below the **<section>** element.

The markup for the **home** section are:

```
<section id="home">
   <article>
      <h1>Foxy Foxes</h1>
      <h2>This site is dedicated to all the lonely foxes out there</h2>
      <p>Lorem ipsum dolor sit amet ...</p>
      <p>Arcu vel condimentum risus facilisis ...</p>
      <p>Mauris accumsan, non magna vitae feugiat ipsum ...</p>
      <p>Hendrerit taciti bibendum ipsum etiam amet ...</p>
   </article>
   <aside>
      <header>
         <h1>Author biography</h1>
      </header>
      <section>
         <p>Lorem ipsum dolor sit amet, lorem tellus egestas ...</p>
      </section>
      <img src="Content/gingerbreadman-96px.png" />
   </aside>
```

```
</section>
```

Adding the images section

Add a **<section>** element below the previous **<section>** element with an id of **images** (it should match the **href** of the second **<a>** tag without the hash tag).

Add an **<h1>** element with the text "*Foxy Images*" and an **<h2>** element with the text "F*oxes in nature*" below the **<h1>** element. Add a couple of paragraphs (**<p>**) with *loerm ipsum* text below the **<h2>** element.

Add three **<article>** elements below the paragraphs inside the **<section>** element. They should contain an image and a paragraph with the texts "*Sleeping fox*", "*Resting fox*", "*Howling fox*" respectively.

1. Add a **<section>** element below the previous **<section>** element with the id **images**.
2. Add an **<h1>** with the text "*Foxy Images*" inside the **<section>** element.
3. And an **<h2>** element with the text "*Foxes in nature*" below the **<h1>** element.
4. Add a few paragraphs (**<p>**) elements with *lorem ipsum* text. The idea is to add some filler text that fills the page vertically.
5. Add an **<article>** element below the last **<p>** element.
6. If you haven't already added three fox images to the **Content** folder then do so before continuing.
7. Add an **<img>** element inside the **<article>** element.
8. Add a paragraph with the text "*Sleeping Fox*" below the image inside the **<article>** element.
9. Repeat 5-7 twice and change the text to "*Resting fox*" and "*Howling fox*".

The markup for the **images** section are:

```
<section id="images">
   <h1>Foxy Images</h1>
   <h2>Foxes in nature</h2>
   <p>Lorem ipsum dolor sit amet ...</p>
   <p>Arcu vel condimentum risus facilisis ...</p>
   <p>Mauris accumsan, non magna vitae feugiat ipsum ...</p>
   <p>Hendrerit taciti bibendum ipsum etiam amet ...</p>
   <article>
```

```
        <img src="Content/red-fox-01.jpg" />
        <p>Sleeping Fox</p>
    </article>
    <article>
        <img src="Content/red-fox-02.jpg" />
        <p>Resting Fox</p>
    </article>
    <article>
        <img src="Content/red-fox-03.jpg" />
        <p>Howling Fox</p>
    </article>
</section>
```

Adding the contact section

Add a **<section>** element below the previous **<section>** element with an id of **content** (it should match the **href** of the third **<a>** tag without the hash tag).

Add an **<article>** element to the **<section>** element and an **<h1>** element with the text "*Contact*" to the **<article>** element.

Add a **<form>** element below the **<h1>** element. Add **input** element for an email address which has auto focus, a **<textarea>** element, a checkbox **input** element and a submit button **input** element. Add labels for the three first controls with the texts "*Email*", "*Message*", "*Send me email updates*" respectively. Separate the controls with line breaks.

1. Add a **<section>** element below the **<section>** element with the id **contact**.
2. Add an **<article>** element to the **<section>** element.
3. Add an **<h1>** with the text "*Contact*" inside the **<article>** element.
4. Add a **<form>** element below the **<h1>** element.
5. Add a **<label>** element inside the **<form>** element and assign "*email*" to its **for** attribute, this will associate it with the **<input>** element with the id *email* which you will add later.
6. Add a line break (**<br />**).
7. Add an **<input>** element below the label with its **type** attribute set to **email**. Give it an **id** of **email**. Add the **autofocus** attribute to give the control focus.
8. Add a line break.

9. Add a **<label>** element and assign "*message*" to its **for** attribute and add the text "Message" as its content.
10. Add a line break.
11. Add an **<textarea>** element below the label and give it an **id** of **message**.
12. Add a line break.
13. Add an **<input>** element with its **type** attribute set to **checkbox**. Give it an **id** of **updates**. Add the **checked** attribute to ensure that it is checked by default.
14. Add a **<label>** element and assign "*updates*" to its **for** attribute. By doing this the user can click on the label to check/uncheck the checkbox. Add the text "*Send me email updates*" as its content.
15. Add a line break.
16. Add an **<input>** element with its **type** attribute set to **submit** and its **value** attribute set to **Send** (the text displayed on the button).

The markup for the **contact** section are:

```
<section id="contact">
   <article>
      <h1>Contact</h1>
      <form>
         <label for="email">Email</label>
         <br />
         <input type="email" autofocus id="email" />
         <br/>
         <label for="message">Message</label>
         <br />
         <textarea id="message" ></textarea>
         <br />
         <input type="checkbox" checked id="updates" />
         <label for="updates">Send me email updates</label>
         <br />
         <input type="submit" value="Send"/>
      </form>
   </article>
</section>
```

Adding the about section

Add a **<section>** element below the previous **<section>** element with an id of **about** (it should match the **href** of the last **<a>** tag without the hash tag).

Add an **<article>** element to the **<section>** element and an **<h1>** element with the text "*About*" to the **<article>** element.

Add an **<address>** element below the **<h1>** element with three paragraphs containing "*Foxes'R'Us*", "*Email: someone@foxesrus.com*", "*Address: Somewhere street 1, Some country*" respectively.

1. Add a **<section>** element below the **<section>** element with the id **about**.
2. Add an **<article>** element to the **<section>** element.
3. Add an **<h1>** element with the text "*About*" inside the **<article>** element.
4. Add an **<address>** element below the **<h1>** element.
5. Add three **<p>** element inside the **<address>** element with the texts "*Foxes'R'Us*", "*Email: someone@foxesrus.com*", "*Address: Somewhere street 1, Some country*" respectively.

The markup for the **about** section are:

```
<section id="about">
   <article>
      <h1>About</h1>
      <address>
         <p>Foxes'R'Us</p>
         <p>Email: someone@foxesrus.com</p>
         <p>Address: Somewhere street 1, Some contry</p>
      </address>
   </article>
</section>
```

Adding the footer

Add a **<footer>** element below the previous **<section>** element.

Add the text "© *2015, All rights reserved*" to the **<footer>** element.

1. Add a **<footer>** element below the **<section>** element.
2. Add the copyright sign (**&copy;**) and the text "*2015, All rights reserved*" to the **<footer>** element.

The markup for the **footer**:

```
<footer>&copy; 2015, All rights reserved</footer>
```

2. Introduction to CSS3

Introduction

This chapter is all about CSS3 and how you can style the elements that make up your web site. This is meant as an introduction, not a comprehensive list of all the CSS styles and HTML elements available for you to use. We will have a brief discussion about the elements you will use when completing the exercises in this book, it will be a great starting point for you when you go out and explore more on your own once you have mastered the basics. You will use CSS throughout the modules in the book to style the elements as you create more complex user controls that sometimes are made up of several components.

You use CSS to control formatting, presentation and layout of a web page by using settings like colors, layouts and fonts. Although you can do styling directly on the HTML elements in the HTML markup it is not recommended that you do so unless you really have to. The preferred approach is to create one or more **.css** files that are linked to individual pages or a master page such as the **_Layout.cshtml** file which renders the separate views in a MVC application.

There are several reasons why you want to use **.css** files instead of inline styling or styling with the **<style>** element in separate pages; it makes it much easier to read the HTML markup, easier to maintain the CSS and HTML markup and a it enables reuse. By separating out the CSS it can be applied to any element in any page without duplication of the CSS and thus achieving consistency across the web site.

Technologies used in this chapter

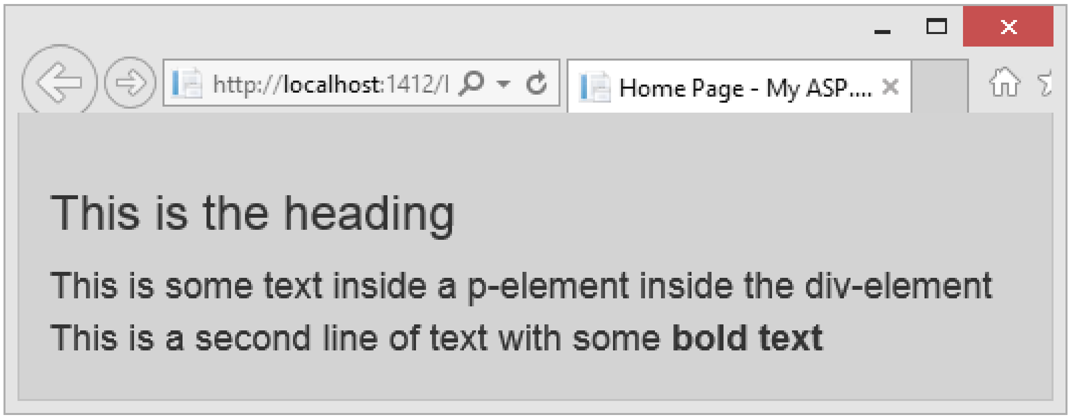
- **HTML 5** - HTML elements are used when illustrating how CSS works.
- **CSS** - Used to style HTML elements.
- **F12 Web Developer Tool** - a browser tool for investigating the Document Object Model (DOM).

Below is a short example of inline styling. Imagine that the page contain hundreds of lines of HTML markup where inline styling is applied, it would get ugly very fast.

```
<body style="background-color:lightgray;">
   <h3>This is the heading</h3>
   <div>
      <p style="font-size:large">This is some text inside a paragraph
         element inside the div-element
         <br/>
         This is a second line of text with some <strong>bold
         text</strong>
      </p>
   </div>
</body>
```

Instead of using inline styling a **<style>** block can be used to define the layout, this would at least keep the CSS separate from the HTML markup.

```
<style type="text/css">
   body { background-color:lightgray; }
</style>
```



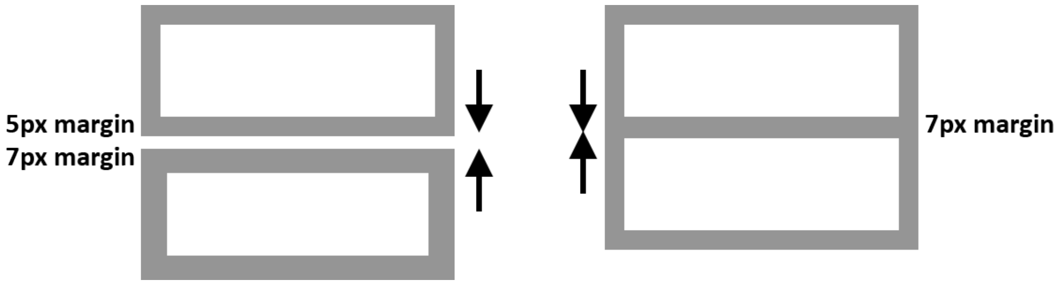
This is the heading

This is some text inside a p-element inside the div-element

This is a second line of text with some **bold text**

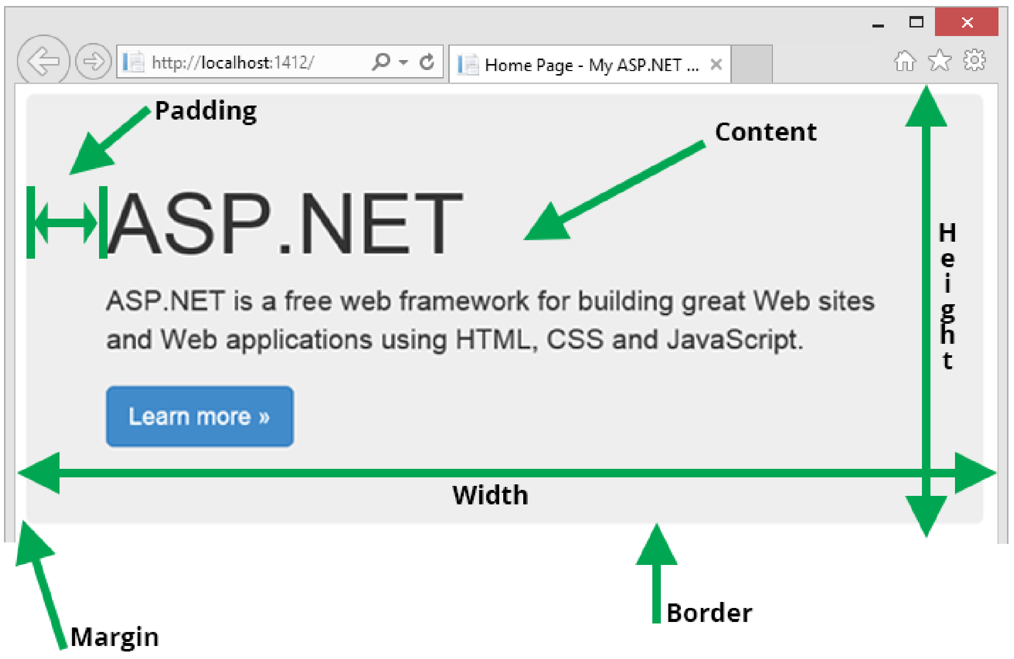
CSS Box Model

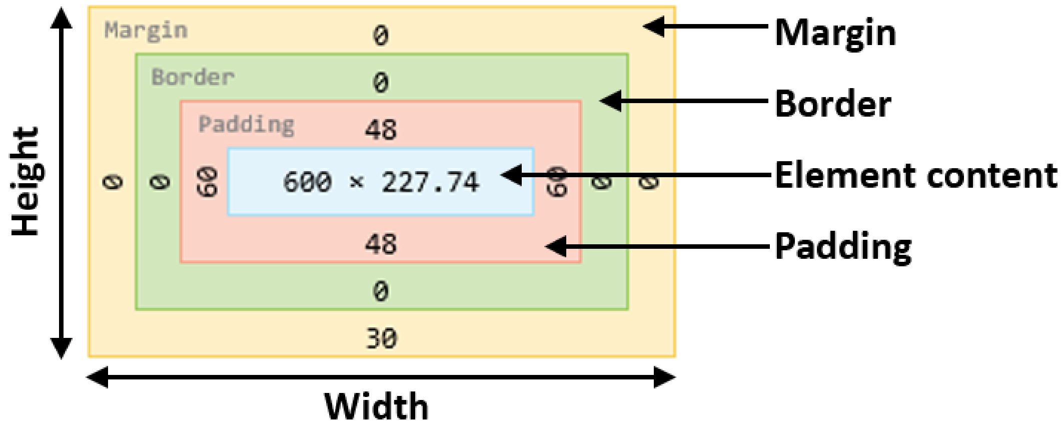
In the CSS box model every element is represented by a square which has a width and a height as well as padding (space used to position the content inside the element), margin (space added to the outside of the element) and a border. When two vertical margins meet they overlap leaving only the maximum margin between the two elements, this does not apply to horizontal margins.



When specifying a width of an element you are specifying the *width of the content area* of that element, any border, margin or padding will add to that width. As an example, if you specify an element's width as 100px and the left and right margin as 5px then the total width of the element would be 110px.

In the example below the CSS box model is displayed for a **<div>** with the Bootstrap **Jumbotron** class applied (you'll learn more about Bootstrap in an upcoming chapter).


http://localhost:1412/
Home Page - My ASP.NET ...
Padding
Content
ASP.NET
ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.
Learn more »
Height
Width
Margin
Border



The Display and Visibility Properties

The **display** property determines how an element should be displayed on screen and is usually set to **block** (placed on a new row), **inline** (placed as an element on the same line. Elements move to a new line when not enough room is available to fit them all on a single line) or **none** (not displayed and does not leave a space where the element should have appeared). You cannot assign widths to inline elements they will only expand to make room for the content, border and padding. If you want to set the width of an inline element its **display** property has to be set to **inline-block**.

The **visibility** property shows or hides an element. A hidden element leaves an empty space where the element should have been displayed if visible.

Property	Description
display:block	Elements are displayed on new lines (default for many element types)
display:inline	Places elements on the same line. Element widths cannot be assigned.
display:none	The element is completely removed from the page.
display:inline-block	Elements are displayed on the same line and widths can be assigned.
visibility:hidden	Hides an element and leaves an empty space where the element should have appeared if visible.

Property values

There are many ways to specify values for CSS properties because there are a lot of things you have to specify values for like borders, background colors, font sizes and many others. You can use keywords to specify some of them like **thin**, **thick** and **large** but these keywords cannot be used everywhere. **thin** for example can be used to define the width of a border but not the size of a font and **large** can be used to define a font size but not a border width.

You can use physical measurements such as centimeter (cm), millimeter (mm), inches (in), picas (pc) where one pc equals 12pt and points (pt) which often are used to define font sizes when writing text in a word processor, where one point is 1/72 of an inch.

Another popular measurement is pixels (px) where 1px is equal to 1/96 of an inch.

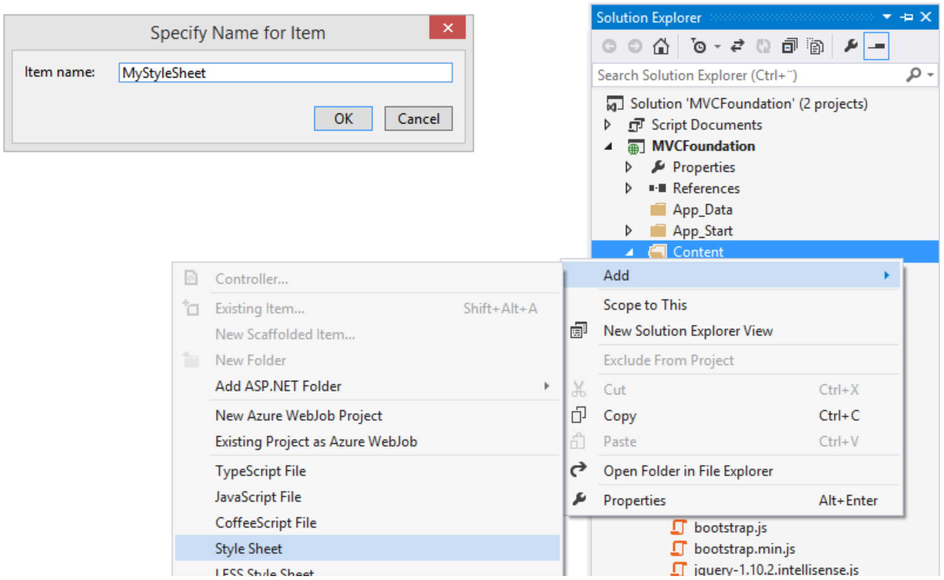
Some prefer to use relative measurements when designing a web site using percent (%) or em where 1em is the current font size of the element and 2em is twice that size. Relative measurements like these lends themselves well when designing for mobile devices.

There are also some CSS functions available such as **rgb** for assigning a color value with decimal numbers for each of the three colors red, green and blue, and **url** for specifying a background image.

You will see some of these in action throughout this chapter and in the rest of the book.

How to add a CSS style sheet

1. Right click on the **Content** folder in Visual Studio and select **Add-Style sheet** in the context menu.
2. Give the style sheet a name in the dialog box and click **OK**.


Specify Name for Item

Item name: MyStyleSheet

3. Open the **Index.html** document and add a reference to the **.css** file. You can do this by dragging it from the solution explorer to the **<head>** element.
4. Now you can add the desired styles to the file and reference them from the HTML markup to style the page elements.

Adding styles to the style sheet

Now that you have created a style sheet and referenced it from the **Index.html** document all styles added to this style sheet will be available throughout the web page.

if you want to remove the ugly inline styling from the HTML markup all you need to do is to move it into the style sheet and reference the desired elements.

Move the styles into the *MyStyleSheet.css* file. When adding a style to a style sheet you place the name-value pairs defining the style rules inside curly braces after the element name.

```css
body { background-color:lightgray; }
p { font-size:large; }
```

The CSS specification is maintained by the World Wide Web consortium (W3C), to keep up with the changes to CSS you can read all about it at www.w3.org/Style/css. It is important to follow the news on this page because there is varying support for CSS among the browsers.

You can find out CSS browser compatibility at:
www.quirksmode.org/compatibility.html.

Style rules

A rule consist of a selector and a block containing properties which are *name-value* pairs, you can find a listing of all selectors and properties on the W3C web site (www.w3schools.com/css). The name and the value of a property is separated by a colon (:) and ends with a semicolon (;). The following example would make the font size larger and the text red for all paragraph (**<p>**) elements.

```
p {
   font-size:large;
   color:#ff0000;
}
```

Selectors

So far you have seen the **body** and **p** selectors, in this section you will learn about some other powerful selectors.

Simple selectors

The previously mentioned **body** and **p** selectors are known as simple selectors that target element types, meaning that the style rules is applied to all element of that type. Other simple selectors you can use are **input**, **button**, **div**, **ul** (unordered list), **li** (list item), **span** (groups inline-elements), **form**, **table**, **tr** (table row), **th** (table header), **td** (table data), **i** (part of a text that is different from the surrounding text), **h1-h6** (headers), **a** (link) and many more.

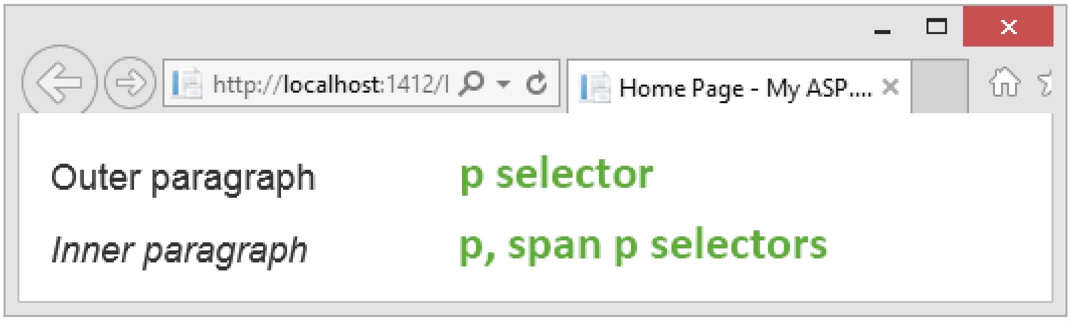
It is possible to stack multiple selectors on the same row targeting them all by separating them with commas (,). It is also possible to target elements within another element by separating them with a space, you can do this for many levels if you have elements nested within elements.

An example could be that you want to target the paragraph **<p>** that is defined inside the inner **<span>** but not the one outside the **<span>** in the following HTML markup.

```
<p>Outer paragraph</p>
<span>
    <p>Inner paragraph</p>
</span>
```

Let's apply the following rules and see what happens. If all goes according to plan the text in the inner paragraph should be displayed italicized and both paragraphs should be displayed with a larger font size.

```
p { font-size: large; }
span p { font-style:italic; }
```

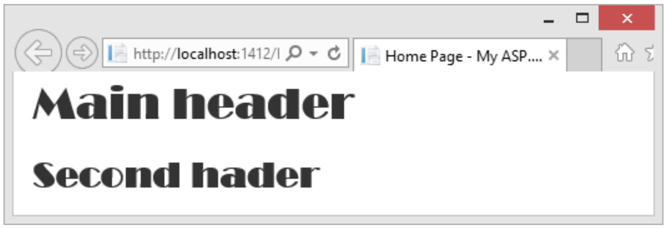

http://localhost:1412/l
Home Page - My ASP....

Outer paragraph

Inner paragraph

The following example targets all headings and make their text bold and change their font family (note that this particular font is not recommended for web pages because it is very hard to read).

```
h1, h2, h3, h4, h5, h6 {
    font-weight:bold;
    font-family: Broadway;
}
```


http://localhost:1412/l
Home Page - My ASP....

Main header

Second hader

Here's a list of simple selectors you will work with in this chapter.

```
a { }
body, h1, h2, h3, h4, h5, h6 { }
button { }
i { }
ul { }
li { }
div { }
span { }
head { }
body { }
form { }
table tr, table th, table td { }
input { }
```

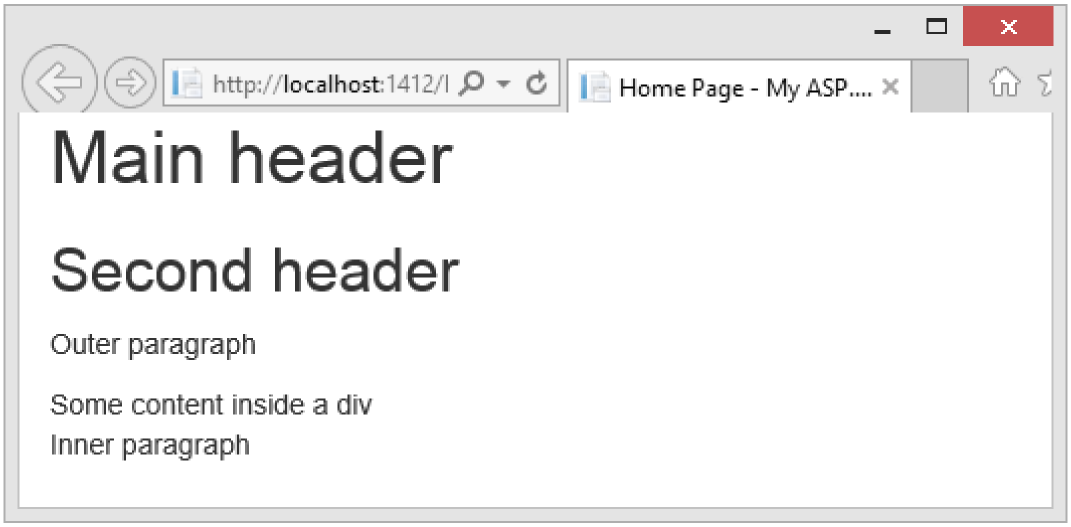
Other selectors

There are other selectors that are frequently used when styling a web page. Instead of targeting elements of a certain type you might want to target an element by its unique id using the #-sign, class names which can be used on multiple elements using a dot (.) or square brackets ([]) to target elements with a certain attribute or attribute value. You can create your own attributes by prefixing the name with **data-**, these attributes will be ignored by the browser but can be used in CSS and JavaScript.

Selector	Description
#id-name	Selects a specific HTML element by its id.
.class-name	Selects all HTML elements decorated with a specific class.
[attribute-name]	Selects all HTML elements decorated with the specified attribute.

The HTML markup for this example has no inline styles and the style sheet has been cleared so that only the default Bootstrap styles are applied.

```
<h1>Main header</h1>
<h2>Second header</h2>

<p>Outer paragraph</p>
<div>Some content inside a div</div>
<div>
    <p>Inner paragraph</p>
</div>
```



Main header

Second header

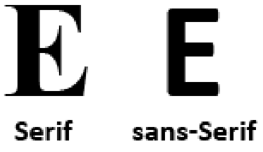
Outer paragraph

Some content inside a div
Inner paragraph

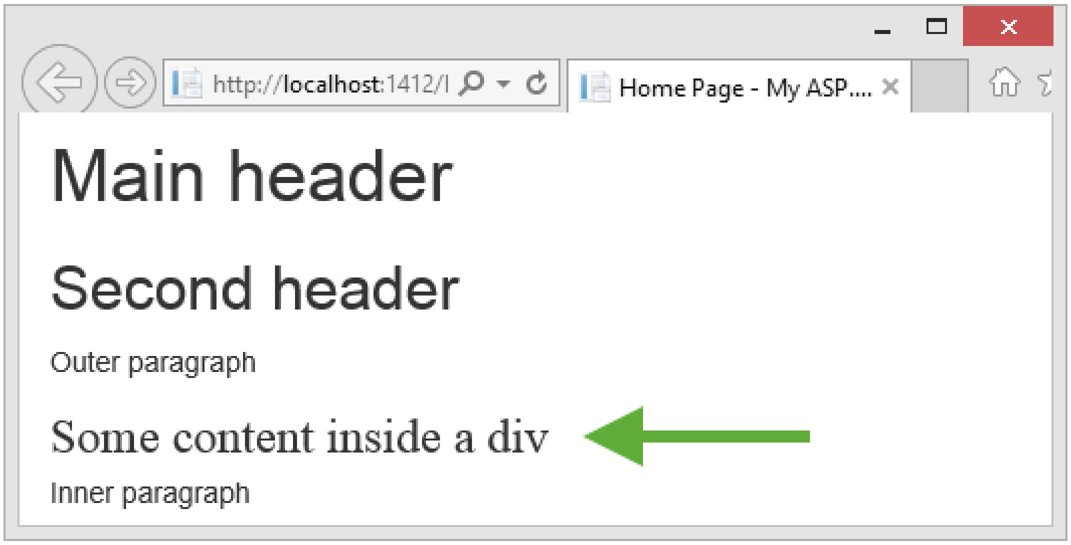
Let's change the styling using an id-selector on the first **<div>**. To do this you have to add a unique id to the element, the id *some-content* is used in this example.

```
<div id="some-content">Some content inside a div</div>
```

Use the **#**-symbol before the name in the style sheet to target this specific **<div>**. Let's make the text stand out more by changing its font to **Time New Roman** which is a *serif* font. *Serif* fonts have small embellishments added to them which their counterpart *sans-serif* does not have. In this example more than one font has been specified; if **Times New Roman** isn't available in the operating system, the last font is the generic **serif** family which can be any serif font.



```
#some-content {
    font-family:'Times New Roman', Times, serif;
    font-size:x-large;
}
```



Main header

Second header

Outer paragraph

Some content inside a div

Inner paragraph

Let's add the class name *paragraph* to the first **<p>** element and the second **<div>** and the attribute *data-heading* to the two headings, then let's add style rules for them in the CSS file. With the added class, attribute and the previously added id on the **<div>** element the HTML markup will look like this.

```
<h1 data-heading>Main header</h1>
<h2 data-heading>Second header</h2>

<p class="paragraph">Outer paragraph</p>
<div id="some-content">Some content inside a div</div>
<div class="paragraph">
    <p>Inner paragraph</p>
</div>
```

When adding a color to a rule in a style sheet a color bar with the most recently used colors in the project will automatically be displayed for you to choose from, if you can't find the color you are looking for there is a more advanced color picker you can open by clicking on the small button with the two arrows at the end of the color bar.

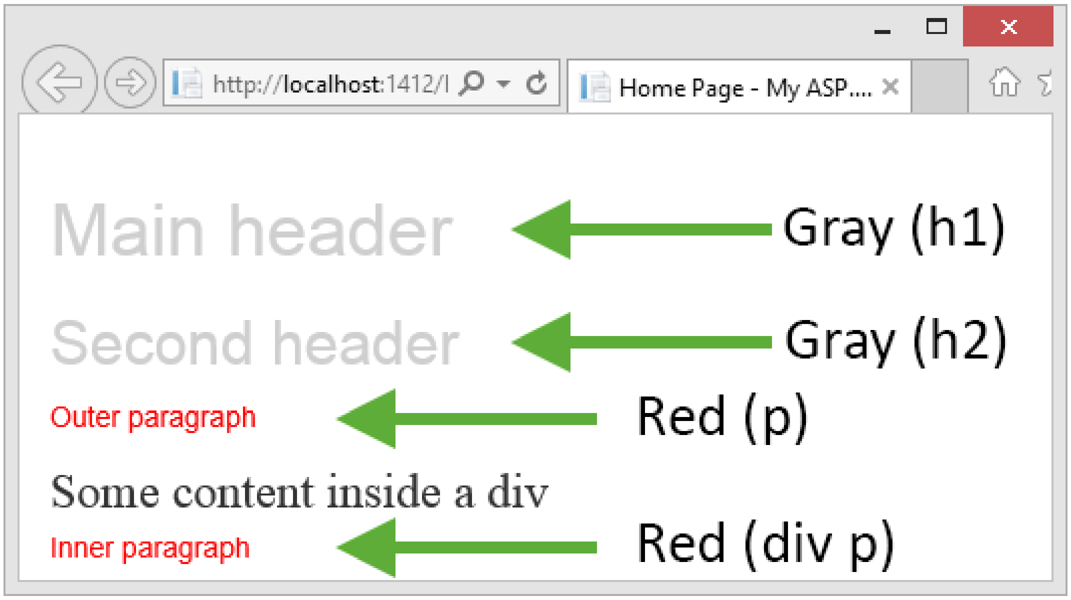
A color can be assigned to a style rule in 3 ways, with the **#**-sign followed by the hexadecimal RGB color values, specifying the color by name or by calling the CSS **rgb** method passing in decimal color values for red, green and blue.

```
color:#cfcfcf;
color:lightgray;
color:rgb(207,207,207);
```

A light gray color (**#cfcfcf**) will give the headings a more dimmed look and a red color added to the elements using the *paragraph* rule will make them stand out from the rest of the text.

```
.paragraph { color:red; }

[data-heading] { color:#cfcfcf; }
```

You might be surprised that the text in the innermost paragraph residing in the **<div>** element turns red despite not having the *paragraph* rule defined. The reason is that the **<p>** element inherits the color from its parent which is the **<div>** with the *paragraph* class.



More complex selectors

You can run into scenarios where you need to target elements within elements of simple types or a certain sub-type like the **input** selector which can represent a button, input field, a checkbox or a radio button. The more commonly used complex selectors for these scenarios are described in the upcoming sections.

Elements separated by space

In a rule where the elements are separated by a space the right element has to reside inside the left element. The same rule applies when a class (.) or a specific id (#) is involved.

- div p
  Targets all **<p>** elements located inside **<div>** elements.
- .class1 .class2
  Targets all elements decorated with the class **class2** located inside an element decorated with the class **class1**.
- #id-name .class1
  Targets all elements decorated with the class **class1** located inside an element with the id **id-name**.

In the following example all text inside **<p>** tags that reside inside a **<div>** will be colored blue.

```
div p { color:blue; }
```

```
<div>
    <p>Outer paragraph</p>
    <div>
        <ul>
            <li><p>Inner paragraph</p></li>
        </ul>
    </div>
</div>
```

Elements separated by a greater than sign (>)

For elements separated by a greater than sign (>) the right element has to be a direct descendant of the left element. The same rule applies when a class (.) or a specific id (#) is involved.

- div > p
  Targets **<p>** elements located directly inside a **<div>** (parent). The **<p>** element (child) cannot be nested inside other element in the **<div>**.
- .class1 > .class2
  Targets elements decorated with the class **class2** located directly inside an element decorated with the class **class1**.
- #id-name > .class1
  Targets elements decorated with the class **class1** located directly inside an element with the specific id **id-name**.

In the following example all the **<p>** (child) elements has to reside directly inside a **<div>** (parent). The **<p>** element (child) cannot be nested inside of other elements inside the **<div>**. The first **<p>** element ("Outer paragraph") will turn blue but not the nested one because it is not a direct descendant of a **<div>** element.

```
div > p { color:blue; }
```

```
<div>
    <p>Outer paragraph</p>
    <div>
        <ul>
            <li><p>Inner paragraph</p></li>
        </ul>
    </div>
</div>
```

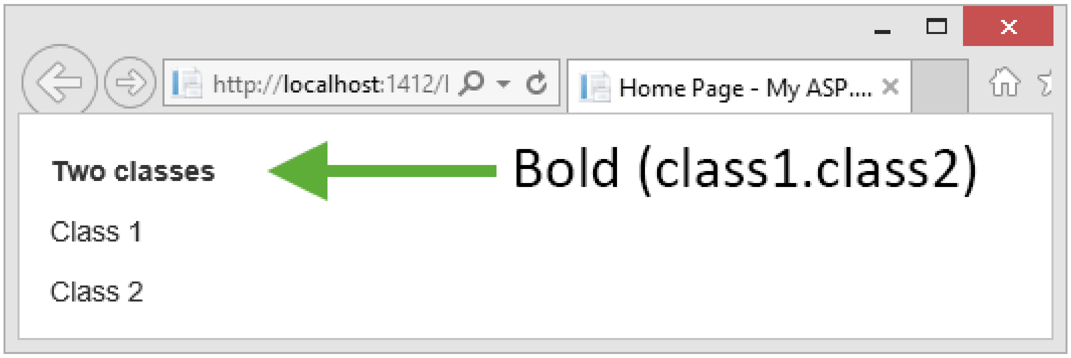
Element selectors directly following each other

Element selectors without any space or other character in between, for instance two class names back to back, requires the target element to have all involved selectors present for the rule to be applied.

- .class1.class2
  The targeted elements has to be decorated with both **class1** and **class2**.
- [attribute-name].class1
  The targeted elements have to be decorated with the attribute **attribute-name** and the class **class1**.

In the following example elements decorated with the classes **class1** and **class2** will have bold text.

```
.class1.class2 { font-weight:bold; }

<p class="class1 class2">Two classes</p>
<p class="class1">Class 1</p>
<p class="class2">Class 2</p>
```

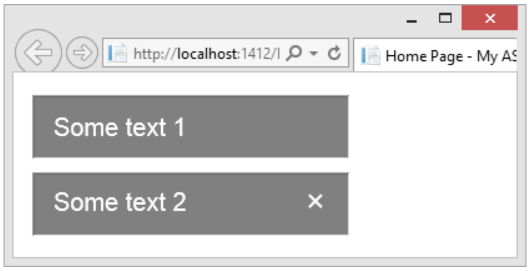


Elements of a certain type

Elements can be selected based on element type. An **input** element can for instance be a button, a textbox, a submit button, a checkbox or a radio button. To target a specific type of **input** element you need to specify the desired type inside square brackets, **input[type="text"]** for textboxes, **input[type="checkbox"]** for checkboxes, **input[type= "submit"]** for a submit button, **input[type="button"]** for buttons and **input[type= "radio"]** for radio buttons.

The following example displays all textboxes on separate rows (**display:block;**) and gives them a bottom margin of 10px. Space is added inside the textboxes (padding), the font size is 20px, the background color is changed to gray and the text color to white.

```
input[type="text"] {
    display:block;
    margin-bottom:10px;
    padding:10px 15px 15px 15px;
    font-size:20px;
    background-color:gray;
    color:white;
}

<input type="text">
<input type="text">
```

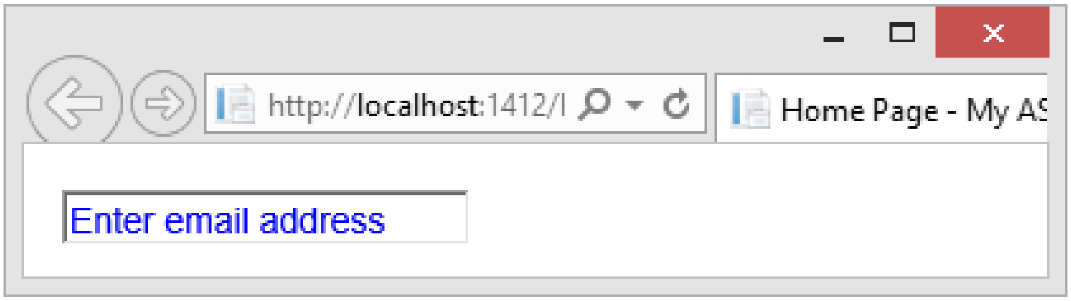


Target elements by attribute

Elements can be selected based on the attributes they are decorated with like the *placeholder* text inside a textbox.

The following example would change the *placeholder* text (the text describing what should be entered into the textbox) from black to blue.

```
input[placeholder] { color:blue; }

<input type="text" placeholder="Enter email address"/>
```



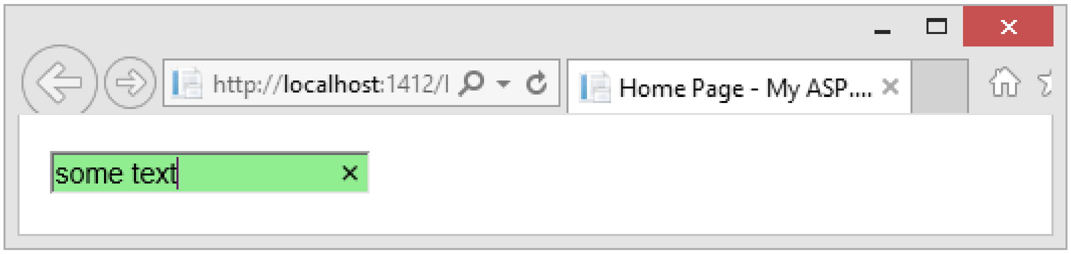
Pseudo classes

Pseudo class selectors are not explicitly assigned to an element but are available as a part of it, among them are **hover**, **visited** and **focus**. The **hover** class is activated when a user hovers with the mouse pointer over an element, the **visited** class is assigned when a user has clicked on a URL link created with an **<a>** element and the **focus** class is activated when a control is the currently used control (is in focus).

Add a colon (:) followed by the name of the desired pseudo class at the end of a CSS selector to target that specific pseudo class. The **focus** pseudo class can for instance be used to change the background color of a textbox when a user clicks in it to enter a value.

```
input[type=text]:focus
{
   background-color:lightgreen;
}
```

```
<input type="text"/>
```



Cascading

CSS rules are applied in a cascading manner to the HTML elements meaning that the most specific rule wins when conflicting rules are resolved.

The following example has three conflicting rules. The first states that the background color of paragraphs (**<p>**) residing in a **<div>** will be green, the second rule states that paragraphs (**<p>**) should have blue backgrounds and the third rule states that all elements should have a red background. The question is which color will be applied to the two paragraphs (**<p>**) inside the **<div>**?

```
div > p { background-color: green; }
p { background-color: blue; }
* { background-color: red; }

<div>
   <p>This is a paragraph</p>
   <p>This is a second paragraph</p>
</div>
```

Because the **div > p** rule is more specific than the other two rules the paragraphs background color will be changed to green.

Style rules can be applied from a variety of sources. You as a developer can provide linked in style sheets, script sections directly in the HTML markup or through the style attribute directly on elements. But styles can also be provided from a user-defined style sheet assigned through the browser settings and from the default styles provided by the browser itself. So which rules will win? The developer styles always weigh heavier than user and browser settings.

One way a rule in a user style sheet can have more importance than the same rule assigned from a style sheet linked to the web application is by using the **!important** keyword in the user style sheet. You as a developer can use this keyword to force an override of a rule from another linked in style sheet.

If you look at the previous example and make one small alteration to the CSS you can force the paragraphs to have a blue background by applying the **!important** keyword.

```
p { background-color: blue !important; }
```

But what happens if the same rule is assigned more than once by the developer in one or more style sheets?

```
p { background-color: blue; }

p { background-color: green; }
```

In this scenario the last rule would win and the background would be green because they have the same weight and the last change to the property will be applied.

If two different style sheets have the same rule defined the order in which the **.css** files have been linked to the web site will be significant because then the rule in the style sheet that is linked in last will win.

Specificity

Each style is given a rating by the browser to determine which rule will be applied, the higher specificity rating the more important the rule is. You can think of the ranking system as having three parts A, B and C where A has a higher ranking than B and B has a higher ranking than C.

A = The number of id selectors present in the rule.
B = The number of class and attribute selectors present in the rule.
C = The number of type selectors present in the rule.

```
*                /* A=0, B=0, C=0 ---   0 */
p                /* A=0, B=0, C=1 ---   1 */
div p            /* A=0, B=0, C=2 ---   2 */
p.paragraph      /* A=0, B=1, C=1 ---  11 */
#content         /* A=1, B=0, C=0 --- 100 */
```

In the image above the **#content** selector will have the highest rating with a rating of 100 and the **p.paragraph** element and class selector will have a rating of 11, all the way down to the ***** selector which has the lowest rating of 0. There is however one rule that trumps all other rules and that is a rule specified as an inline style using the **style** attribute on an element.

```
<p style="background-color:purple;">This is a paragraph</p>
```

A winning rule only replaces the conflicting CSS properties not the entire rule content. If lesser ranked rules contain CSS properties which not are part of the winning rule then those properties will still be in effect after the rules have been evaluated.

Important: Only CSS properties that are in conflict with other properties will be replaced.

If you go back to the example from the previous section you can now see why the paragraphs had a green background after the rules had been applied. The **div > p** rule only applies to paragraphs who are immediate children of **<div>** elements, for other paragraphs the second rule would win and the background would be blue.

```
div > p { background-color: green; }    /*A=0, B=0, C=2 ---   2*/
p { background-color: blue; }           /*A=0, B=0, C=1 ---   1*/
* { background-color: red; }            /*A=0, B=0, C=0 ---   0*/
```

Inheritance

Certain CSS properties are inherited from the parent to its children. If you for instance set the **color** property using the **body** selector all elements inside the **<body>** element will be affected and get the same text color. This is only true for some CSS properties like the text color which you probably only want to assign once and maybe override for specific elements.

In this example all paragraphs except the one with the **content** id will have yellow text and the paragraph with the **content** id will have green text.

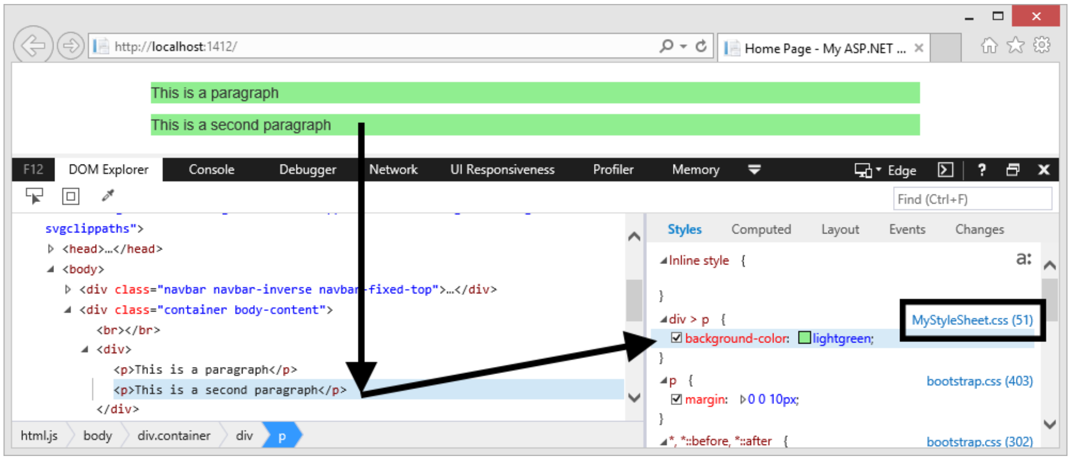
```
body { color:yellow; }
#content { color:green; }

<body>
   <p id="content">This is a paragraph</p>
   <p>This is a second paragraph</p>
   <p>This is a third paragraph</p>
</body>
```

Developer Tools (F12)

One of the most important tools when designing a web page is the Developer Tools which you can open by pressing the **F12** key in the browser. This tool allows you to inspect the HTML elements and their associated styles, it can be invaluable in situations where you are unsure why a style hasn't been applied the way you thought it would be.

In this section Internet Explorer is used to illustrate the **F12** tool, other browsers have similar functionality which you easily can learn by yourself once you have seen how it works in IE.

This example inspects the second paragraph in the Document Object Model (DOM) tree view which shows that the **background-color** property has been applied through *MyStylesheet.css*.

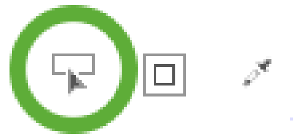


```
svgclippaths">
 <head>…</head>
 <body>
   <div class="navbar navbar-inverse navbar-fixed-top">…</div>
   <div class="container body-content">
     <br></br>
     <div>
       <p>This is a paragraph</p>
       <p>This is a second paragraph</p>
     </div>
```

```
Inline style {

}
div > p {                                MyStyleSheet.css (51)
  ☑ background-color: 🟩lightgreen;
}
p {                                      bootstrap.css (403)
  ☑ margin: 0 0 10px;
}
*, *::before, *::after {                 bootstrap.css (302)
```

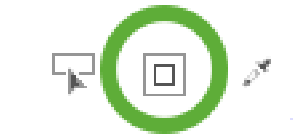
Inspecting an element by clicking it in the web page

To inspect an element on the web page using Internet Explorer you click the left most button in the toolbar and then the element on the web page that you want to inspect.



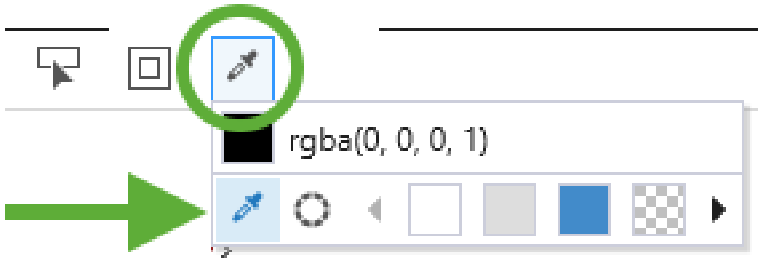
Inspecting an element by selecting it in the tree view

To inspect an element by hovering over or clicking an element in the tree view you click the middle button in the toolbar and then click the element you want to inspect in the tree view.



Inspecting the color of an element

To inspect an element's color you click the third button in the toolbar and then the eyedropper button. Finally you click on an element on the web page and copy the color as an **rgba** function from the textbox and paste it into a style sheet.



Change style rules directly in the browser

When designing a page you might want to see what the result would be if you changed one or more CSS rules. You can temporarily change CSS rules directly in the browser by using the **F12** tool and simply add, delete or change rules in the CSS section of the tool window. You can make changes by clicking or right clicking on a CSS property or inside the curly braces of a rule.

Change HTML directly in the browser

When you are designing a page you might want to see what the result would be if you changed one or more HTML elements. You can temporarily change elements directly in the browser by using the **F12** tool and simply add, delete or change elements in the HTML tree of the tool window. You can make changes by clicking or right clicking on an element.

CSS Reset

A CSS reset style sheet is a collection of CSS rules and properties that can be used to reduce browser inconsistencies for default line heights, margins and font sizes of HTML elements. All the major browsers applies a default style sheet to the HTML content being displayed in the view port. This can pose a problem in that the content by default will be rendered differently in every browser and the outcome of the styles you add can be affected and display the content differently among the browsers.

To solve this problem you can begin with a clean slate by adding a CSS reset style sheet that is loaded before any other styles are added. You can download a commonly used reset at http://meyerweb.com/eric/tools/css/reset/ or search for "*CSS reset*" in you favorite search engine. Below is the CSS reset from the page mentioned earlier.

```
html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6,
p, blockquote, pre, a, abbr, acronym, address, big, cite, code, del,
dfn, em, img, ins, kbd, q, s, samp, small, strike, strong, sub, sup, tt,
var, b, u, i, center, dl, dt, dd, ol, ul, li, fieldset, form, label,
legend, table, caption, tbody, tfoot, thead, tr, th, td, article, aside,
canvas, details, embed, figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary, time, mark, audio, video
{
   margin: 0;
   padding: 0;
   border: 0;
   font-size: 100%;
   font: inherit;
   vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure, footer, header, hgroup,
menu, nav, section
{
   display: block;
}
body
{
   line-height: 1;
}
ol, ul
{
   list-style: none;
}
blockquote, q
{
   quotes: none;
}
blockquote:before, blockquote:after, q:before, q:after
{
   content: '';
   content: none;
}
table
{
   border-collapse: collapse;
   border-spacing: 0;
}
```

Exercise 4: Styling the Foxy Foxes web page

In this exercise you will use CSS to style the HTML5 elements in the Index.html document for the Foxy Foxes web page.

Tip: You can press **Ctrl+E+D** to organize the content in a CSS style sheet.

Adding a CSS file

When styling an application you should strive to keep the CSS styling in separate files which you link into the web site where needed. Add a CSS file called **site.css** and add a link to it in the **Index.html** file.

1. Open the Foxy Foxes project.
2. Right click on the **Content** folder and select **Add-Style Sheet**.
3. Name it **site.css** and click the **OK** button.
4. Open the **Index.html** page.
5. Drag the **site.css** file from the Solution Explorer and drop it below the **<meta>** tag in the **Index.html** file or add it manually by writing the HTML **<link>** element yourself.
   ```
   <link href="Content/site.css" rel="stylesheet" />
   ```

Changing the font family

Using a clean readable uniform font for the web site gives it a more professional look. There are many free fonts you can use, one popular font family is Google's *Open Sans* which can be referenced from https://www.google.com/fonts.

1. Browse to https://www.google.com/fonts.
2. Click on the **Quick-Use** button for the *Open Sans* font family.
3. Select **Normal 400** in the list. The more versions of the font you select the greater the impact will be on the load time.
4. Copy the HTML link in the gray area below the font list.
   ```
   <link href='http://fonts.googleapis.com/css?family=Open+Sans'
   rel='stylesheet' type='text/css'>
   ```
5. Open the **Index.html** page in the Foxy Foxes project.
6. Paste in the link you copied below the **<meta>** tag inside the **<head>** element.
7. Open the **site.css** file and locate the **<body>** selector block.

8. Change the font family to **Open Sans** and use the generic **Sans Serif** font family as the fallback font family. This will change the font family for the whole site because it is inherited by the child elements.

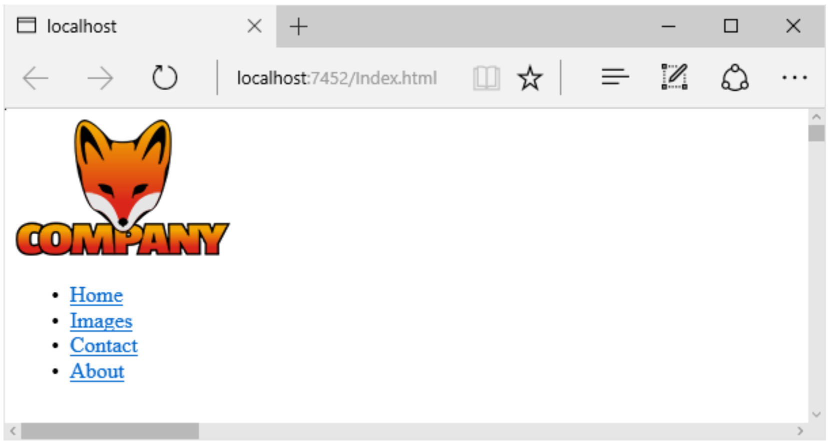
```
body {
    font-family: 'Open Sans', sans-serif;
}
```

Styling the navigation area

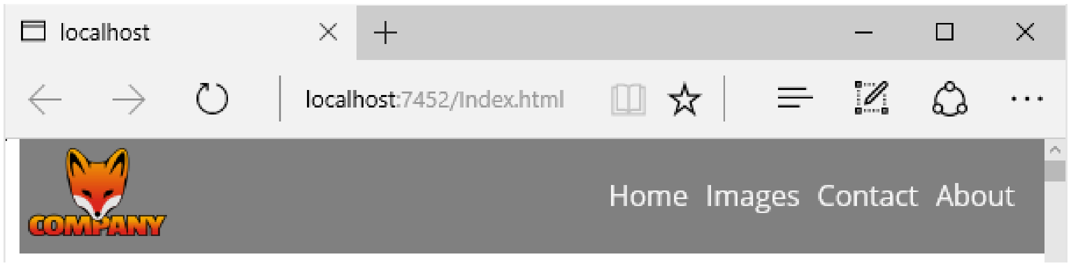
The way the navigation area is styled by default leaves a lot to be desired, let's change the layout to a menu bar fixed to the top of the web page so that the content scrolls beneath it and the menu stays on-screen at all times.

Give it a dark gray background color (#808080) and display it across the whole page width. Give it a padding of 5px to give the content a little breathing room and keep it from being displayed flush to the sides.

The images below show the navigation area with and without styling.

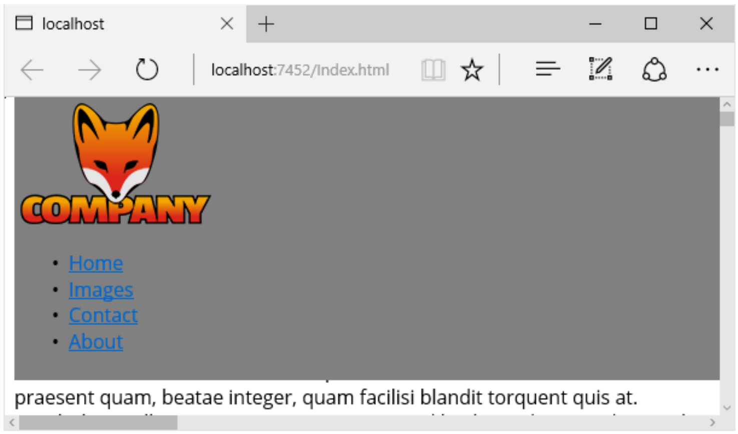


- Home
- Images
- Contact
- About



Styling the background and positioning the navigation area

You will have to add a selector to the **site.css** file that targets the **<header>** element inside the **<body>** element to style the navigation area.



praesent quam, beatae integer, quam facilisi blandit torquent quis at.

1. Open the **site.css** file.
2. Add a selector which target the **<header>** element inside the **<body>** element.
   ```
   body > header { }
   ```
3. Add a CSS property that changes the background color to a dark gray.
   ```
   background-color: #808080;
   ```
4. Make the header area stretch over the whole page width.
   ```
   width: 100%;
   ```
5. Add a 5px padding to all sides of the navigation area.
   ```
   padding: 5px;
   ```
6. Position the navigation area and fix it to the top of the page.
   ```
   top: 0;
   position: fixed;
   ```
7. View the web page in the browser by right clicking on the **Index.html** page in the Solution Explorer and select **View in Browser**.

NOTE: *You don't have to close the browser when updating the HTML markup or the CSS styling. When you want to view your changes you simply save the files with **Ctrl+Shift+S** and refresh the web page in the browser.*

The complete **header** styling:

```
body > header {
    background-color: #808080;
    width: 100%;
    padding: 5px;
    top: 0;
    position: fixed;
}
```

Resizing the image

The logotype is too large for the navigation area and needs to be resized to 50px high. To style the image you have to add a selector to the **site.css** file that targets the **<img>** element inside the **<header>** element.

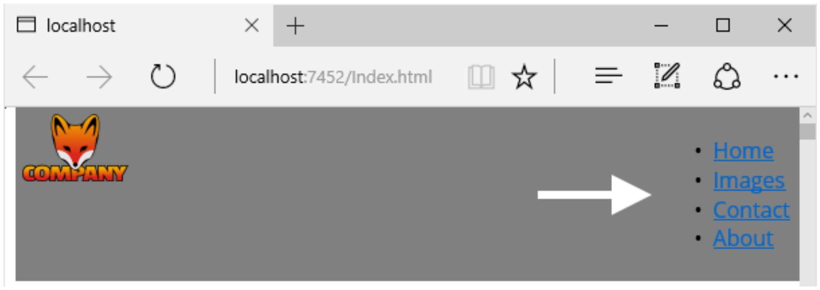
1. Open the **site.css** file.
2. Add a selector which target images inside the **<header>**.
   `body > header img { }`
3. Resize the image to be 50px high.
   `height: 50px;`
4. Save the files and switch to the browser. Refresh the page to view the changes.

The complete **image** styling:

```
body > header img {
    height: 50px;
}
```

Positioning the menu links to the right side

You need to reposition the **<nav>** element to the right side of the **<header>** element by floating it to the right and giving it a 20px right margin.



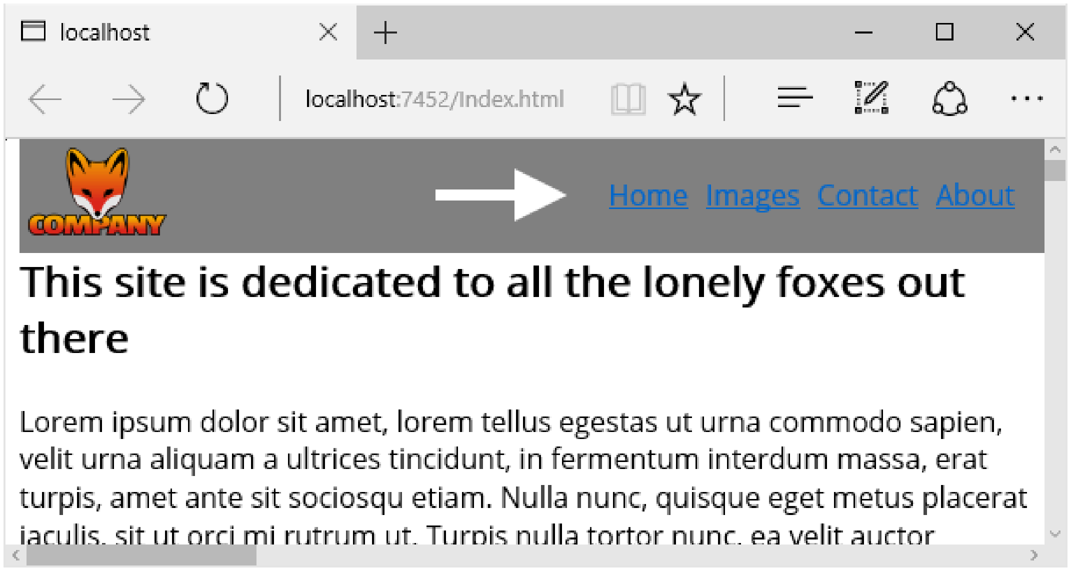
1. Open the **site.css** file.
2. Add a selector which target the **<nav>** element inside the **<header>**.
   ```
   body > header nav { }
   ```
3. Position the **<nav>** element containing the links to the right side of the **<header>** element with the **float** property.
   ```
   float: right;
   ```
4. Add a 20px right margin to the **<nav>** element.
   ```
   margin-right: 20px;
   ```

The complete **nav** styling:

```
body > header nav {
    float: right;
    margin-right: 20px;
}
```

Positioning the menu links within the <nav> element

You need to reposition the link list items within the **<nav>** element to its left side by floating them to the left and giving them a 10px right margin to add some space between the links. You also want to remove the bullets from the **<li>** elements to make them look more like menu items using the **line-style** property.



This site is dedicated to all the lonely foxes out there

Lorem ipsum dolor sit amet, lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum massa, erat turpis, amet ante sit sociosqu etiam. Nulla nunc, quisque eget metus placerat iaculis, sit ut orci mi rutrum ut. Turpis nulla tortor nunc, ea velit auctor

1. Open the **site.css** file.
2. Add a selector which target the **<li>** elements inside the **<nav>**.
   ```
   body > header nav li { }
   ```
3. Position the **<li>** elements to the left side of the **<nav>** element using the **float** property.
   ```
   float: left;
   ```
4. Add a right margin to add space between the **<li>** elements.
   ```
   margin-right: 10px;
   ```
5. Remove the bullets from the **<li>** elements to make them look more like menu items.
   ```
   list-style: none;
   ```

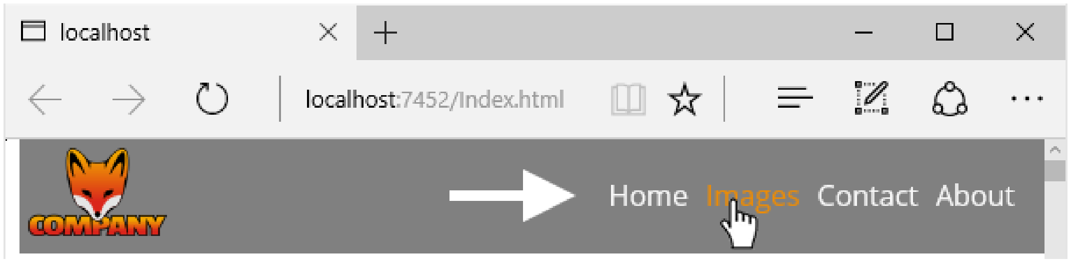
The complete **list item** styling:

```
body > header nav li {
    float: left;
    margin-right: 10px;
    list-style: none;
}
```

Removing the link underline and changing the link color

To make the links look like menu options you could remove their underlining and change the link text to a white color. Change the link text color to orange when the mouse hovers over a link. You can achieve this by targeting the anchor tags (**<a>**) inside the list items (**<li>**) and the use the **:hover** pseudo class to change the color when hovering.

Tip: You can use the **F12** tool to copy a color from an image or any web page element. There is an eyedropper tool in Internet explorer's **F12** tool which copies a color from an element.



1. Open the **site.css** file.
2. Add a selector which target the **<a>** elements inside the **<li>**.
   ```
   body > header nav li a { }
   ```
3. Change the link text color to white.
   ```
   color: #f7f7f7;
   ```
4. Remove the links underlining.
   ```
   text-decoration: none;
   ```
5. Add a selector which target the **:hover** pseudo class on the **<a>** elements.
   ```
   body > header nav li a:hover { }
   ```
6. Change the color to one of the orange color tones of the fox's head. You can use the **F12** tool to copy the color.
   ```
   color: rgba(235, 139, 6, 1);
   ```

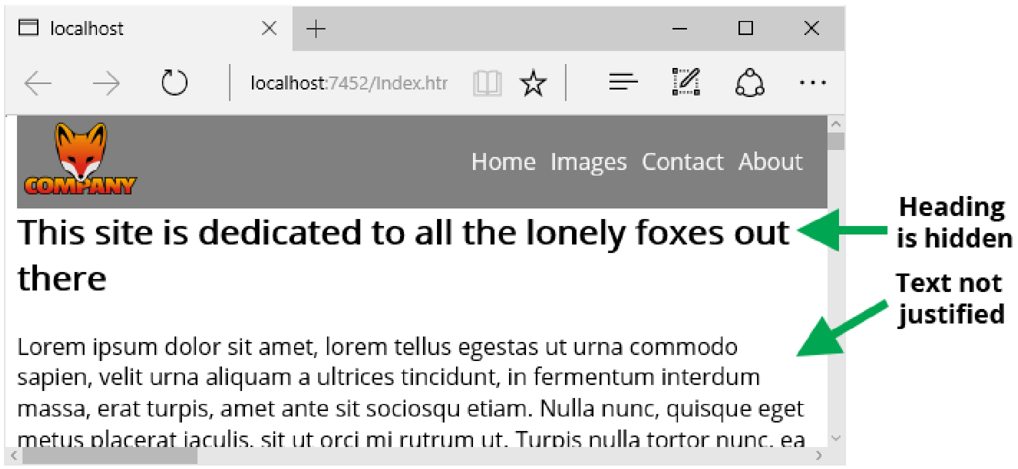
The complete **anchor tag** styling:

```
body > header nav li a {
    color: #f7f7f7;
    text-decoration: none;
}

    body > header nav li a:hover {
        color: rgba(235, 139, 6, 1);
    }
```

Adding padding to and justify the text in the sections

As you might have noticed the first heading is hidden by the navigation bar when the web page is loaded, to fix this you can add 50px padding to the top of the sections which are immediate children to the **<body>** element. It would also look nice if the text in the **<section>** element paragraphs are justified.

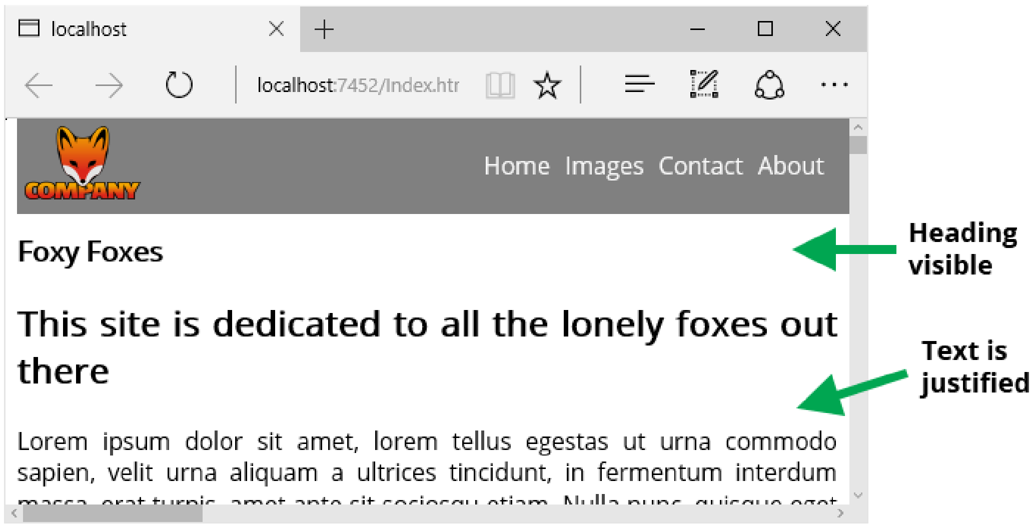
This is what the heading and text look like before the CSS is applied.



This site is dedicated to all the lonely foxes out there

Lorem ipsum dolor sit amet, lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum massa, erat turpis, amet ante sit sociosqu etiam. Nulla nunc, quisque eget metus placerat iaculis, sit ut orci mi rutrum ut. Turpis nulla tortor nunc, ea

This is what the heading and text look like after the CSS has been applied.



Foxy Foxes

This site is dedicated to all the lonely foxes out there

Lorem ipsum dolor sit amet, lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum massa, erat turpis, amet ante sit sociosqu etiam. Nulla nunc, quisque eget

1. Open the **site.css** file.
2. Add a selector which target the **<section>** elements that are immediate children to the **<body>** element.
   ```
   body > section { }
   ```
3. Add 50px top padding.
   ```
   padding-top: 50px;
   ```

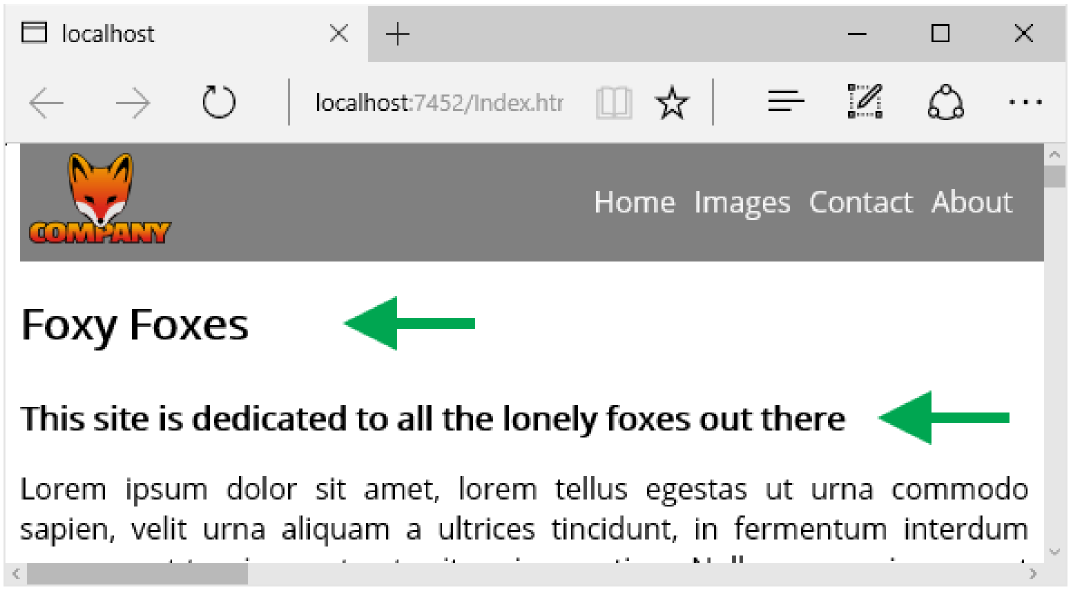
4. Justify the text.
   ```
   text-align: justify;
   ```

The complete **section** element styling:

```
body > section {
    padding-top: 50px;
    text-align: justify;
}
```

Changing the heading font size

As you can see in the previous image the **<h1>** and **<h2>** headings have the wrong size. To fix this you will have to add two CSS selectors targeting all **<h1>** and **<h2>** headings and assign 24px to the **<h1>** heading and 18px to the **<h2>** heading.



Foxy Foxes

This site is dedicated to all the lonely foxes out there

Lorem ipsum dolor sit amet, lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum

1. Open the **site.css** file.
2. Add a selector which target the **<h1>** all elements.
   ```
   h1 { }
   ```
3. Change the font size to 24px.
   ```
   font-size: 24px;
   ```
4. Add a selector which target the **<h2>** all elements.
   ```
   h2 { }
   ```

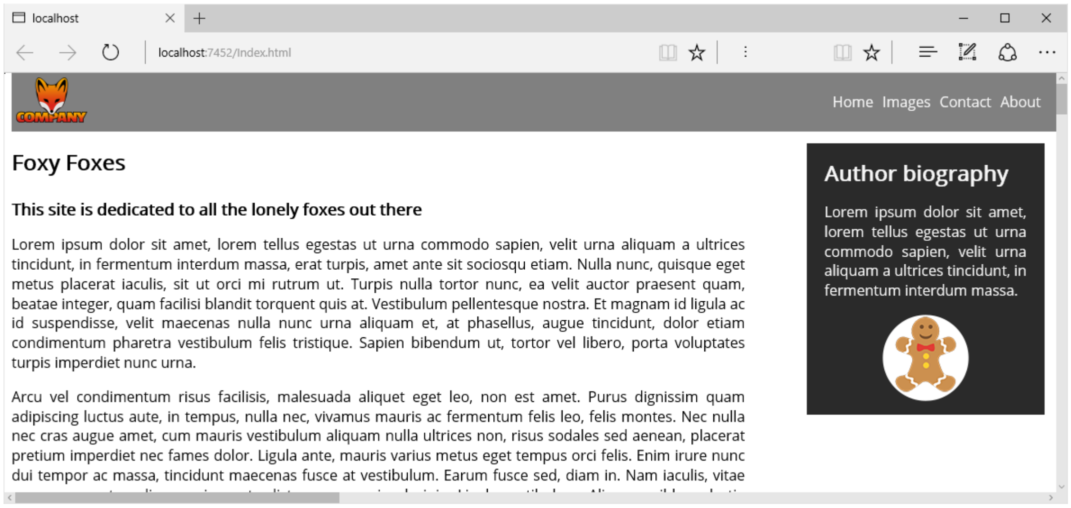
5. Change the font size to 18px.
   ```
   font-size: 18px;
   ```

The complete **heading** styles:

```
h1 {
    font-size: 24px;
}

h2 {
    font-size: 18px;
}
```

Styling the home section

To style the *home* section you will have to float the **<article>** element to the left to make room for the **<aside>** element. The **<aside>** element will have to be floated right to be displayed on the right side of the **<article>** element (see image below).



Foxy Foxes

This site is dedicated to all the lonely foxes out there

Lorem ipsum dolor sit amet, lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum massa, erat turpis, amet ante sit sociosqu etiam. Nulla nunc, quisque eget metus placerat iaculis, sit ut orci mi rutrum ut. Turpis nulla tortor nunc, ea velit auctor praesent quam, beatae integer, quam facilisi blandit torquent quis at. Vestibulum pellentesque nostra. Et magnam id ligula ac id suspendisse, velit maecenas nulla nunc urna aliquam et, at phasellus, augue tincidunt, dolor etiam condimentum pharetra vestibulum felis tristique. Sapien bibendum ut, tortor vel libero, porta voluptates turpis imperdiet nunc urna.

Arcu vel condimentum risus facilisis, malesuada aliquet eget leo, non est amet. Purus dignissim quam adipiscing luctus aute, in tempus, nulla nec, vivamus mauris ac fermentum felis leo, felis montes. Nec nulla nec cras augue amet, cum mauris vestibulum aliquam nulla ultrices non, risus sodales sed aenean, placerat pretium imperdiet nec fames dolor. Ligula ante, mauris varius metus eget tempus orci felis. Enim irure nunc dui tempor ac massa, tincidunt maecenas fusce at vestibulum. Earum fusce sed, diam in. Nam iaculis, vitae

Author biography

Lorem ipsum dolor sit amet, lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum massa.

Scale the **<article>** element to 85% of its original width and the **<aside>** element to 10% of its original width. Also give the **<aside>** element 20px padding to the left and right sides, 20px top margin, 5px right margin, a dark gray background color (#f7f7f7). To position the image inside the **<aside>** element it must be displayed as a **block**, have **auto** left and right margins and a bottom margin of 15px.

To stop the next **<section>** element to float with the content of the *home* section you have to clear the floats on both sides.

Styling the article in the home section

1. Open the **site.css** file.
2. Add a selector which target the **<article>** element in the *home* **<section>**.
   ```
   #home article { }
   ```
3. Scale the article to 85% of its normal width.
   ```
   width: 85%;
   ```
4. Float the content to the left. You need to do this in order to position the **<aside>** element to the right.
   ```
   float: left;
   ```

The complete **article** styling:

```
#home article {
   width: 85%;
   float: left;
}
```

Styling the aside in the home section

1. Open the **site.css** file.
2. Add a selector which target the **<aside>** element in the *home* **<section>**.
   ```
   #home aside { }
   ```
3. Scale the aside to 10% of its normal width.
   ```
   width: 10%;
   ```
4. Float the **<aside>** element to the right.
   ```
   float: right;
   ```
5. Add 20px padding to the left and right sides, 20px top margin and 5px right margin.
   ```
   padding: 0px 20px;
   margin-top: 20px;
   margin-right: 5px;
   ```
6. Add a dark gray background color and a white text color.
   ```
   background-color: #2a2a2a;
   ```

```
      color: #f7f7f7;
```

The complete **aside** styling:

```
#home aside {
   width: 10%;
   float: right;
   padding: 0px 20px;
   margin-top: 20px;
   margin-right: 5px;
   background-color: #2a2a2a;
   color: #f7f7f7;
}
```

Styling the image in the aside element

1. Open the **site.css** file.
2. Add a selector which target **<img>** elements in the **<aside>** element.
   ```
   #home article img { }
   ```
3. Change the image's **display** property to **block** to be able to position it correctly.
   ```
   display: block;
   ```
4. Add automatic left and right margins to position the image horizontally and give it a bottom margin of 15px.
   ```
   margin-left: auto;
   margin-right: auto;
   margin-bottom: 15px;
   ```
5. Save the changes and refresh the web page. Note that the content in the *images* **<section>** element is floating to the right of the **<aside>** content.
6. Open the **site.css** file.
7. Add a selector which target *images* **<section>** element.
   ```
   #images { }
   ```
8. Stop the floating by clearing it on both sides of the element.
   ```
   clear: both;
   ```
9. Save the changes and refresh the web page. Note that the content in the *images* **<section>** element now are below the *home* **<section>**.

The complete **images** styling:

```
#images {
    clear: both;
}
```

Styling the images section

To display the images without needing to scroll the page they should be scaled to 70% of their original width. The images and their descriptive text should be centered on the screen horizontally.

To achieve this you need to target the **<article>** and **<img>** elements in the *images* **<section>**.

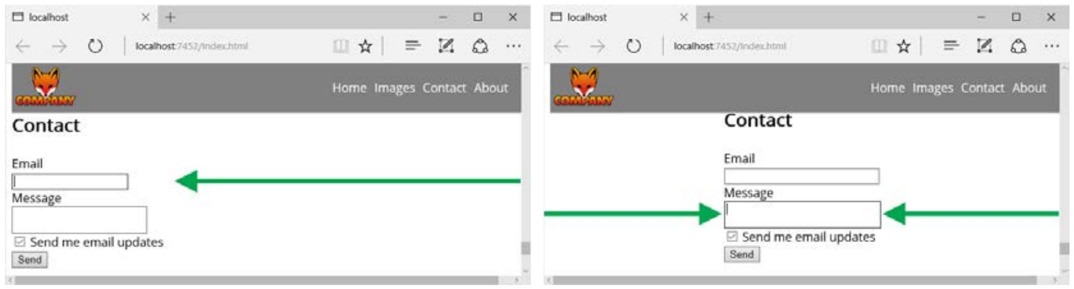
1. Open the **site.css** file.
2. Add a selector which target the **<article>** element in the *images* **<section>**.
   ```
   #images article { }
   ```
3. Center the text; this will center the images as well.
   ```
   text-align: center;
   ```
4. Add a selector which target the **<img>** elements in the *images* **<section>**.
   ```
   #images img { }
   ```
5. Set the image width to 70% to avoid scrolling.
   ```
   max-width: 70%;
   ```
6. Save the changes and refresh the web page. Make sure that the images and their descriptions are centered.

The complete **<article>** and **<img>** styling:

```
#images article {
    text-align: center;
}

#images img {
    max-width: 70%;
}
```

Styling the contact section

To display the form in the *contact* section centered over the page the **<section>** element has to have a fixed width and auto margins. You can use a 205px width since the text elements will be 200px wide.



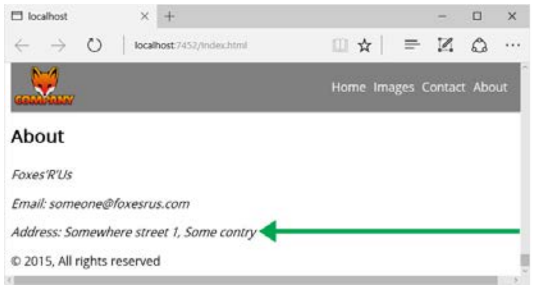
1. Open the **site.css** file.
2. Add a selector which target the *contact* **<section>**.
```
#contact { }
```
3. Assign auto margins and a width of 205px to the element.
```
margin:auto;
width:205px;
```
4. Add a selector which target the email **<input>** and **<textarea>** elements in the *contact* **<section>**.
```
#contact input[type='email'], #contact textarea { }
```
5. Set the element widths to 200px.
```
width: 200px;
```
6. Save the changes and refresh the web page. Make sure that the form is centered.

The complete **contact** section styling:

```
#contact {
    margin:auto;
    width:205px;
}
    #contact input[type='email'], #contact textarea {
        width: 200px;
    }
```

Styling the about section

To display the about information as centered text you have to change its text alignment to **centered**.



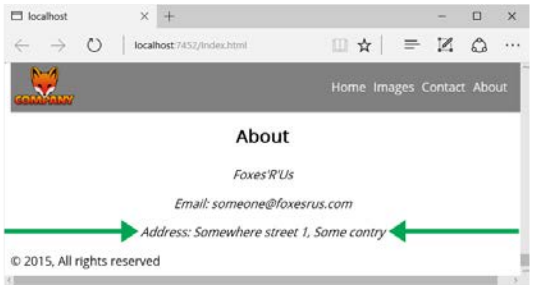
About

Foxes'R'Us

Email: someone@foxesrus.com

Address: Somewhere street 1, Some contry

© 2015, All rights reserved



About

Foxes'R'Us

Email: someone@foxesrus.com

Address: Somewhere street 1, Some contry

© 2015, All rights reserved

1. Open the **site.css** file.
2. Add a selector which target the *contact* **<section>**.
   ```
   #about { }
   ```
3. Change the text alignment to center the text.
   ```
   text-align: center;
   ```
4. Save the changes and refresh the web page. Make sure that the about section has centered text.

The complete **about** section styling:

```
#about {
    text-align: center;
}
```

3. Introduction To Bootstrap

Introduction

Bootstrap is a free open-source frontend toolkit originally developed by Twitter for designing and building web applications using HTML, CSS and JavaScript. It has become very popular with web developers and designers because of its flexibility and ease of use. By using Bootstrap you can have a well designed and fabulous looking web site in no time because it contains the foundation you need for buttons, labels, badges, tables, layout, typography, forms, widgets like picture carousels and much more.

Another benefit is that the web pages will look great on any device because Bootstrap is designed with mobile first in mind. The layout is determined by the resolution of the device displaying the web pages. This saves you the time having to fix a design for mobile devices once the design for the main web site is finished.

Being built on open standards such as HTML, CSS and JavaScript it can be used with any platform, server technology and editor.

A prerequisite for using Bootstrap is that you have basic skills in HTML, CSS and Java-Script, don't worry you don't have to be an expert on those topics when you begin using Bootstrap you will learn the necessary skills as you progress through this book.

Bootstrap is a framework containing well factored single responsibility CSS classes which means that each class has a single purpose. Instead of creating a traditional large and bulky CSS class which styles an entire component, such as a table, Bootstrap gives you many small classes all with a specific purpose, from which you can pick-and-choose when styling your components.

Apart from CSS Bootstrap also comes with a JavaScript library for the more complex components and widgets such as picture carousels. This library is only required if you use components and widgets that depend on JavaScript.

When creating an MVC Web Application in Visual Studio 2015 a default Bootstrap theme is already included in the project providing you with styled web pages from the get go. If you don't like the default theme you can easily download another for free at sites like *www.bootswatch.com* or *www.bootstrapzero.com*. There is also a plethora of other free and paid sites where you can find really nice themes. It can be a real time saver to have a finished theme and tweak it a little to make it perfect for your web site.

To get started you only need to learn a few core Bootstrap classes and once you have mastered those the rest will be easy to figure out.

Bootstrap provides CSS classes for layout and styling controls, JavaScript functions for widget functionality and over 250 glyphs optimized in a few image files. A glyph is a symbol described as a scalable font which can be incorporated into text and components displayed on the web page.

In a web project the files are located in three different folders. The JavaScript files are located in the **Scripts** folder, the CSS style sheets are located in the **Content** folder and the glyphs are located in the **fonts** folder. The JavaScript and CSS files have two versions ending with *.js* or *.css* and *.min.js* or *.min.css*. The *.min* files are **minified** versions of the original files made smaller by removing unnecessary characters, spaces and comments. Using minified versions will make the page load faster and create a better user experience.

Important: *It is also important to note that the Bootstrap JavaScript files are dependent on the JQuery library which is installed by default in MVC projects.*

The *bootstrap.js* file handles JavaScript for widgets and components which rely on code to work properly.

Technologies used in this chapter

- **HTML 5** - HTML elements are used when illustrating how Bootstrap works.
- **Bootstrap** - Used to style HTML elements and the web site.

Installing Bootstrap

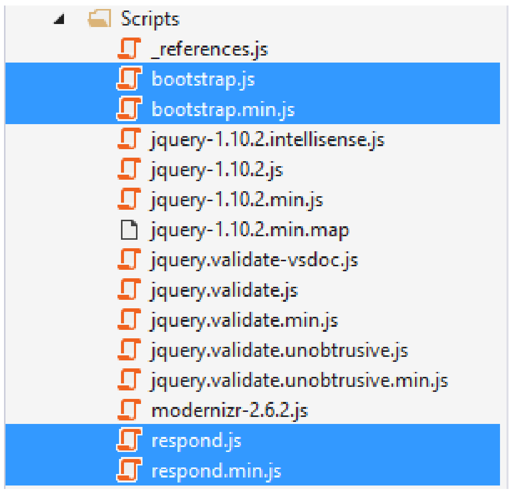
When beginning a web project with an empty project template in Visual Studio you have to add the Bootstrap and JQuery libraries to it manually.

1. Right click on the project name in the Solution Explorer and select **Manage NuGet Packages**.
2. Type "*Bootstrap*" into the search field.
3. Click on the **Install** button for the **Bootstrap** package.

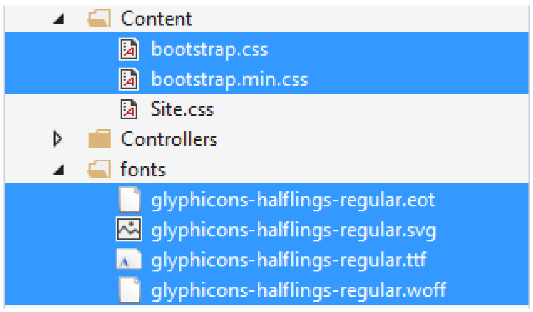
This will add JQuery, the Bootstrap CSS and JavaScript files as well as a library of Glyphicons which are symbols in the form of scalable fonts. You can use Glyphicons instead of image icons which generally takes longer to load because they are larger.

Scripts & CSS

Respond.js is worth mentioning here because it is a *polyfill* which provide alternate solutions for browsers that don't support specific features.



```
Scripts
    _references.js
    bootstrap.js
    bootstrap.min.js
    jquery-1.10.2.intellisense.js
    jquery-1.10.2.js
    jquery-1.10.2.min.js
    jquery-1.10.2.min.map
    jquery.validate-vsdoc.js
    jquery.validate.js
    jquery.validate.min.js
    jquery.validate.unobtrusive.js
    jquery.validate.unobtrusive.min.js
    modernizr-2.6.2.js
    respond.js
    respond.min.js
```

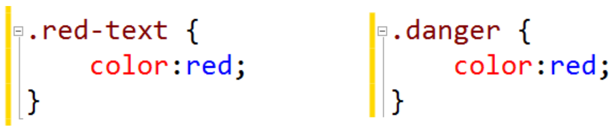


```
Content
    bootstrap.css
    bootstrap.min.css
    Site.css
Controllers
fonts
    glyphicons-halflings-regular.eot
    glyphicons-halflings-regular.svg
    glyphicons-halflings-regular.ttf
    glyphicons-halflings-regular.woff
```

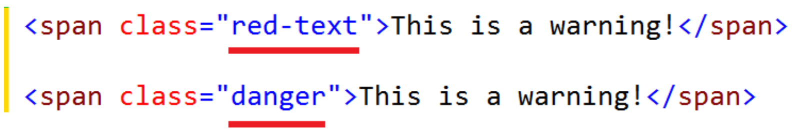
The CSS classes implemented by Bootstrap have meaningful descriptive names and does not contain any implementation detail. An example of a very specific (and perhaps not so well thought out) CSS class name for displaying an error text is **red-text**. The name does not say in what context it should be used or why it is named the way it is, is it because it denotes an error or simply that you enjoy the color red?

By using a name like that you have essentially created a class which only can be used to display text using a red foreground color, if you wanted to change the color (for error messages) at a later time you would have to change the name of the class and all places in the code where the name is used throughout the entire application.

Instead Bootstrap provides classes for displaying text for different scenarios or contexts such as **success** (green), **warning** (orange), **danger** (red) and **info** (light blue). Note that the class names don't contain any specifics about the default color associated with them. This strategy is deliberate because it makes it very easy to override the behavior of a class without changing its name. If you want to change the text color for warning messages in your application you simply override that name and change the color in a CSS file which instantly changes the color in the entire application.



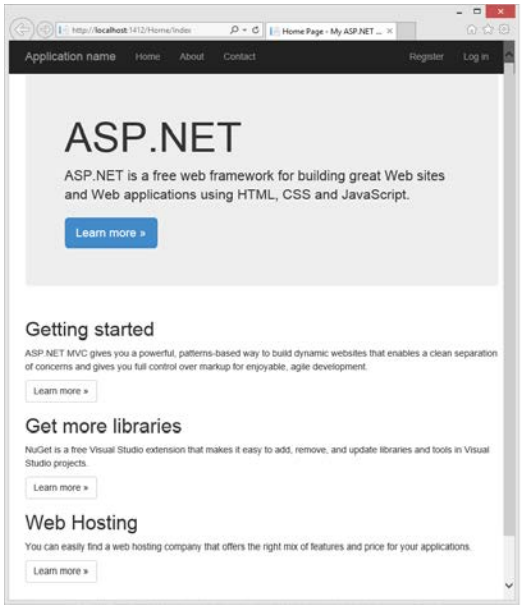
```
.red-text {                  .danger {
    color:red;                   color:red;
}                            }
```



```
<span class="red-text">This is a warning!</span>

<span class="danger">This is a warning!</span>
```

Another example of a poorly chosen CSS class name is **left-side** for a content area which probably will appear to the left side on the page, if we are to believe the class name. But that is not a given, the CSS class could be changed to display the area on the right side of the page but still have the old name and thus creating confusion. A better name would be **side-bar** which don't specify exactly where the area will appear only that it is to be displayed to one of the sides of the page, left or right.

To give you an example of what Bootstrap really do with the web page layout of the web pages out-of-the-box I have commented out the *bootstrap.css* reference which is loaded when the web application starts. Below you can see the page displayed with and without Bootstrap present in an MVC application. Without Bootstrap the page get a retro feel taking you back to the early days of internet.


http://localhost:1412/Home/Index | Home Page - My ASP.NET ...

Application name Home About Contact Register Log in

ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Learn more »

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

Learn more »

Get more libraries

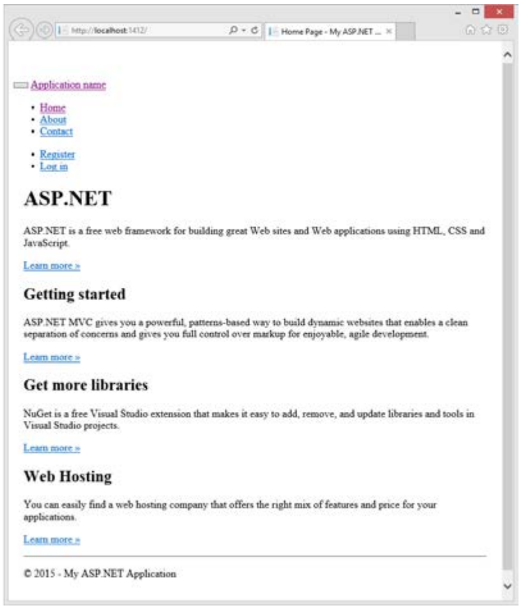
NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Learn more »

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

Learn more »


http://localhost:1412/ | Home Page - My ASP.NET ...

Application name

- Home
- About
- Contact

- Register
- Log in

ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Learn more »

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

Learn more »

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Learn more »

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

Learn more »

© 2015 - My ASP.NET Application

Typography

Looking at the image above you can clearly see that the page not only is better looking but the font family has changed as well to a cleaner and more readable sans-serif. If you are not happy with the default Bootstrap font you can always override it in the *site.css* file and specify a new font family. If you want to change it for every rendered element in the **body** segment of the page you could add a different font family to the **body** rule overriding the font family defined by Bootstrap.

```
body {
    font-family:'Franklin Gothic Medium', 'Arial Narrow', Arial,
    sans-serifArial;
}
```

Glyph Icons

Instead of using images for icons, which can add to the page load time, you can opt to use glyph icons which are defined as scalable fonts. There are over 250 glyphs available in Bootstrap alone but there are other providers as well.

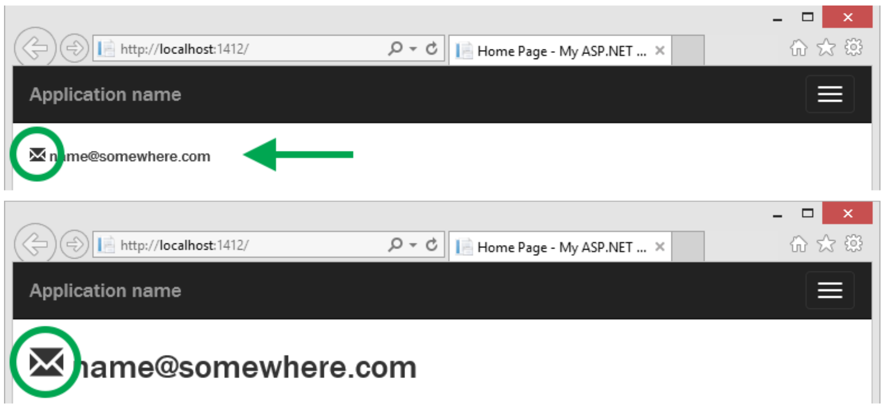
You can easily add a glyph to a control or text to make it stand out more and describe the purpose of the control with an illustration. You can see all the available glyphs, CSS classes and widgets provided by Bootstrap at *http://getbootstrap.com/components/*; the site also contain basic examples on how to use them.

Let's say you want to display an envelope icon to the left of an email address. Instead of uploading an image of an envelope with the solution you can add a glyph to the paragraph.

The one rule you must adhere to when using glyphs is that they have to be added as classes to an empty **<span>** element. When using most Bootstrap classes you have to specify a main class describing the control or its main usage (in this case **glyphicon**) as well as a defining class (in this case **glyphicon-envelope**). The **glyphicon** class specifies that a glyph is being added and the **glyphicon-envelope** class which glyph to display.

```
<p>
    <span class="glyphicon glyphicon-envelope" aria-hidden="true"></span>
    name@somewhere.com
</p>
```

As you can see in the image below the glyph scales really well when changing font size.



Font Awesome

There are other glyph providers that you might want to take a look at. One is *Font Awesome* which is widely used in the industry, it can be downloaded from *GitHub* at *http://fontawesome.github.io/Font-Awesome/.*

Important: *It can be tricky to get Font Awesome to work with applications uploaded to Azure.*

Grid Layout and Responsive Design

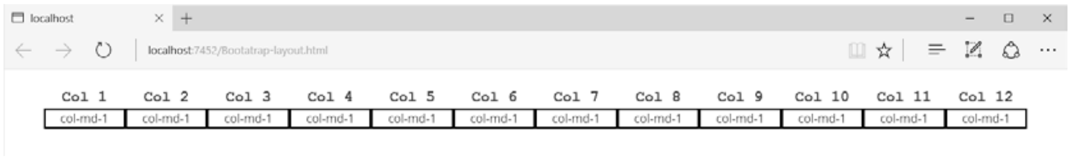
Grid layouts are not a new invention for web pages they have been around since the Sumerians first started taking down inventory on clay tablets. One of the features of using a grid layout is that it gives structure. It helps organize the data in rows and columns and to enforce margins. It is also easier for the user to take in the information since our eyes have been conditioned to move from left to right or right to left when reading information on a page.

Grids are used to create order. We love them so much that we often place grids inside of grids. This type of layout lends itself for browsers since every component on a web page is made up of rectangular elements. No matter how you style an element the browser will still consider it a four sided element.

The Bootstrap grid layout helps you by providing a container which centers the content on the page and structures the content into rows and columns. You can have as many rows as you need and up to 12 columns per row. It is also possible to nest a grid inside of a cell; the nested grid can then have up to 12 columns taking up the full space of the cell it is nested within.

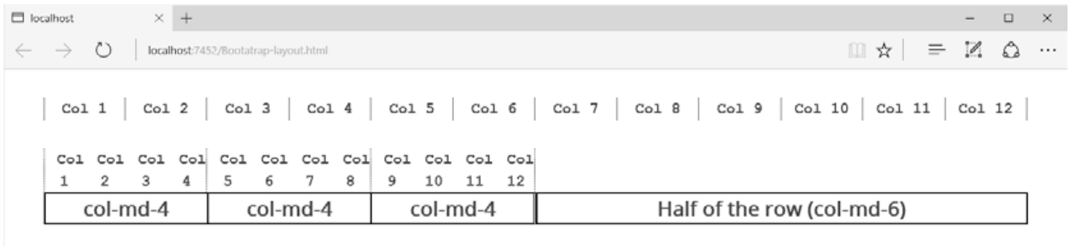
If not all columns of a row are used an empty area representing the remaining unused columns will be displayed.

The content flows within each column moving a cell's overflowing content to a new row within that element.


localhost
localhost:7452/Bootstrap-layout.html
Col 1 Col 2 Col 3 Col 4 Col 5 Col 6 Col 7 Col 8 Col 9 Col 10 Col 11 Col 12
col-md-1 col-md-1 col-md-1 col-md-1 col-md-1 col-md-1 col-md-1 col-md-1 col-md-1 col-md-1 col-md-1 col-md-1

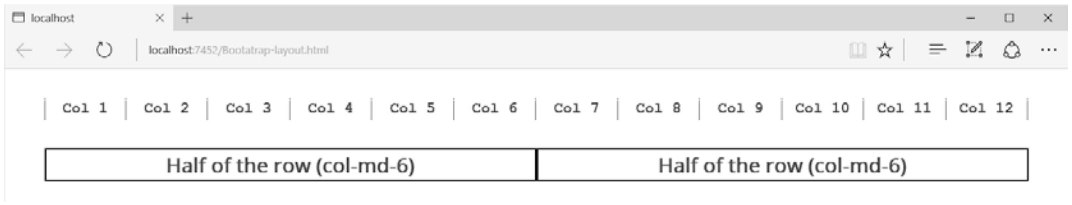
```
<div class="row">
    <div class="col-md-1">col-md-1</div>
    <div class="col-md-1">col-md-1</div>
    @*... 10 more columns*@
</div>
```

The second row in the grid below is divided into two equally sized columns each 6 units wide where the left column is divided into 3 equally sized columns each 4 units wide. This is accomplished by nesting a second row divided into 3 columns inside the outer column effectively dividing the available 50% of the outer row into 3 columns taking up 33% each. Remember that the total number of columns per row must not exceed 12. The outer row consists of two columns each 6 units wide where the first column is divided into 3 columns each 4 units wide which adds up to 12 columns.


localhost
localhost:7452/Bootstrap-layout.html
Col 1 Col 2 Col 3 Col 4 Col 5 Col 6 Col 7 Col 8 Col 9 Col 10 Col 11 Col 12
Col 1 Col 2 Col 3 Col 4 Col 5 Col 6 Col 7 Col 8 Col 9 Col 10 Col 11 Col 12
col-md-4 col-md-4 col-md-4 Half of the row (col-md-6)

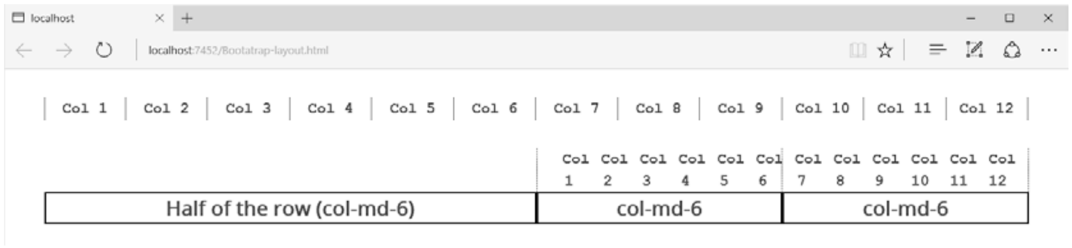
```
<div class="row" >
     <div class="col-md-6">
         <div class="row">
             <div class="col-md-4">col-md-4</div>
             <div class="col-md-4">col-md-4</div>
             <div class="col-md-4">col-md-4</div>
         </div>
     </div>
     <div class="col-md-6">Half of the row (col-md-6)</div>
</div>
```

The next row is divided into two columns each 6 units wide.


localhost
localhost:7452/Bootatrap-layout.html
Col 1
Col 2
Col 3
Col 4
Col 5
Col 6
Col 7
Col 8
Col 9
Col 10
Col 11
Col 12
Half of the row (col-md-6)
Half of the row (col-md-6)

```
<div class="row">
    <div class="col-md-6">Half of the row (col-md-6) </div>
    <div class="col-md-6">Half of the row (col-md-6) </div>
</div>
```

The last row in the grid is divided into two equally sized columns each 6 units wide where the second column is divided into two columns each 6 units wide.



```
<div class="row">
    <div class="col-md-6">Half of the row (col-md-6)</div>
    <div class="col-md-6">
        <div class="row">
            <div class="col-md-6">col-md-6</div>
            <div class="col-md-6">col-md-6</div>
        </div>
    </div>
</div>
```

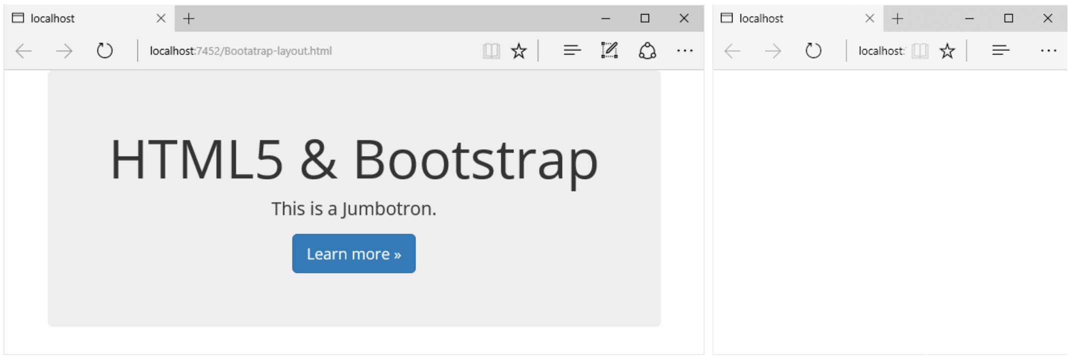
The **row** class is used when creating new rows in the grid and by doing so the content below that row will be pushed down the page. Each row can have up to 12 columns which you create by using the **col-** classes, there are several different **col-** classes targeting different device resolutions (see table on page 76).

This mean that you can create different page layouts depending on the screen resolution you are targeting. You might want to display content in one way for a full screen desktop computer and in a completely different way on a smaller device. By targeting the device resolution using the **col-** classes you can achieve different layouts for different devices. If you only specify one specific device resolution using the **col-** classes then the unspecified resolutions will be displayed using the default Bootstrap layout for those resolutions. For instance when the resolution falls below 992px the content begin to stack vertically which might not be the layout you want. You can change this behavior either by specifying that it should use the same layout for small, medium and large device types or you can provide a unique layout for a specific scenario.

You can also hide content for certain resolutions. Let's say that you only want to display a picture banner on devices with a resolution greater 992px and not on smaller devices such as mobile phones. To achieve this you can either use the **hidden-xs** and **hidden-sm** classes to hide the content on those types of devices or use **visible-lg** and **visible-md** to only show the content on medium and large sized devices.

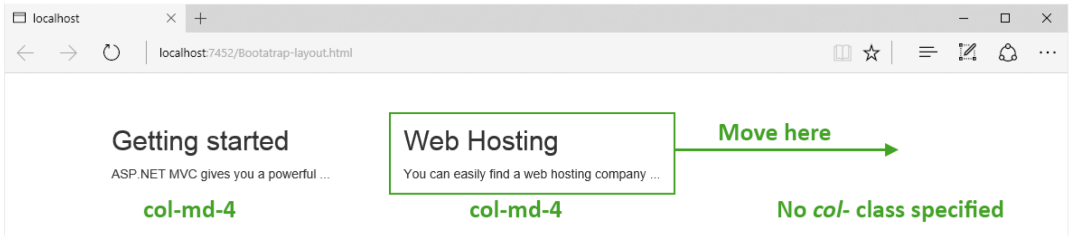
In the example below the **Jumbotron** (the gray area) is hidden on extra small devices using the **hidden-xs** Bootstrap class which mean that it will be hidden when the resolution is below 768px. You could achieve the same result by adding the **visible-sm**, **visible-md** and **visible-lg** classes to the **Jumbotron**.

```
<div class="jumbotron hidden-xs">
   <h1>ASP.NET</h1>
   <p class="lead">ASP.NET is a free web framework ... </p>
   <p><a href="http://asp.net" class="btn btn-primary btn-lg">
      Learn more &raquo;</a></p>
</div>
```

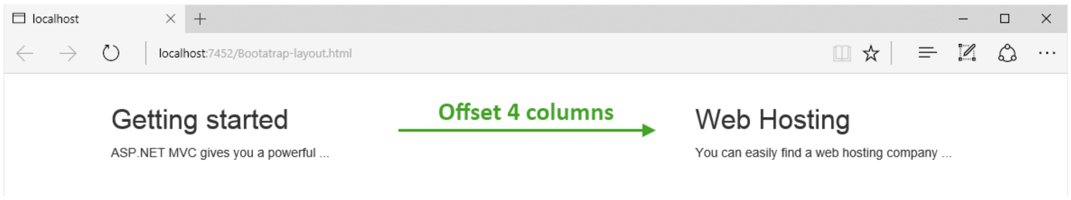


Column class	Resolution	Description
col-lg-*columns*	≥ 1200px	Displays the content on large devices
col-md-*columns*	≥ 992px	Displays the content on medium and large devices
col-sm-*columns*	≥ 768px	Displays the content on small to large devices
col-xs-*columns*	< 768px	Displays the content on extra small to large devices
hidden-*size* (lg, md, sm, xs)		Hides the content for the specified resolution and displays it for other resolutions
visible-*size* (lg, md, sm, xs)		Displays the content for the specified resolution and hides it for other resolutions

You can move content in relation to where it originally was placed in the HTML markup by using the **col-*xx*-offset**, **col-*xx*-push** and **col-*xx*-pull** classes on an element. Let's say you have two grid columns each taking up 4 units of the 12 available. The two pieces of content are flush against the left side of the grid (see image) and now you want to move the right most of the two columns to be flush against the right side of the grid, to achieve this you can use one of the **col-*xx*-offset** classes. Remember not to exceed the maximum 12 colum limit.


localhost
localhost:7452/Bootatrap-layout.html
Getting started
ASP.NET MVC gives you a powerful ...
Web Hosting
You can easily find a web hosting company ...
Move here
col-md-4
col-md-4
No *col-* class specified

```
<div class="row">
    <div class="col-md-4">
        <h2>Getting started</h2>
        <p>ASP.NET MVC gives you a powerful ...</p>
    </div>
    <div class="col-md-4 col-md-offset-4">
        <h2>Web Hosting</h2>
        <p>You can easily find a web hosting company ...</p>
    </div>
</div>
```



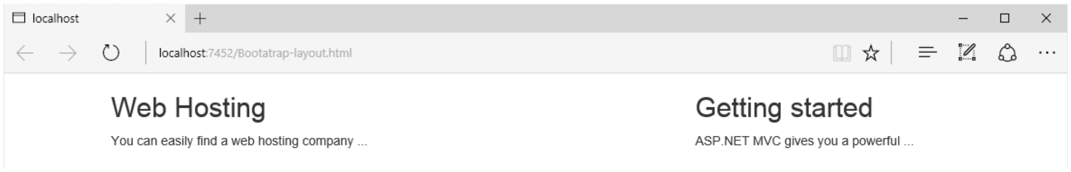
Getting started

ASP.NET MVC gives you a powerful ...

Web Hosting

You can easily find a web hosting company ...

If you have the same scenario as in the previous example with two columns flush to the left side of the grid and the HTML markup is defined in a specific order that you don't want to change. Your dilemma is that the two columns should appear in the reverse order and displayed flush against the left and right sides of the grid (the same end result as in the previous example but the columns should be reversed). To achieve this you can use the **col-*xx*-push** and **col-*xx*-pull** classes.



Web Hosting

You can easily find a web hosting company ...

Getting started

ASP.NET MVC gives you a powerful ...

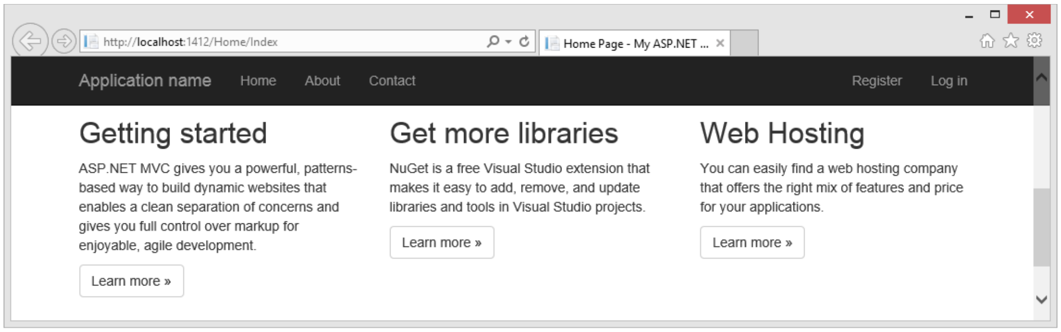
```
<div class="row">
   <div class="col-md-4 col-md-push-8">
      <h2>Getting started</h2>
      <p>ASP.NET MVC gives you a powerful ... </p>
   </div>
   <div class="col-md-4 col-md-pull-4">
      <h2>Web Hosting</h2>
      <p>You can easily find a web hosting company ...</p>
    </div>
</div>
```

Class	Description
col-*size*-offset-*columns* (size: lg, md, sm, xs)	Offsets (moves) the content in relation to where it originally was placed in the grid.
col-*size*-pull-*columns*	Pulls the column the specified number of columns to the left.
col-*size*-push-*columns*	Pushes the column the specified number of columns to the right.

Bootstrap provides responsive design optimizing the output for different devices, on small devices the content will be displayed vertically and on larger horizontally. Once the device drops below a certain resolution the content will automatically be stacked vertically unless you change the layout. The only thing you have to do is to use the correct Bootstrap classes and Bootstrap will style the page asking the device for its capabilities using media queries.

```
@media (min-width: 768px) {
    @*Styles for small devices*@
}
```

When the browser width is reduced and falls below 992px the content stacks vertically instead of horizontally. If the width is less than 768px the menu changes to the "hamburger menu", the button with three lines which opens a drop down menu.



Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

Learn more »

Get more libraries

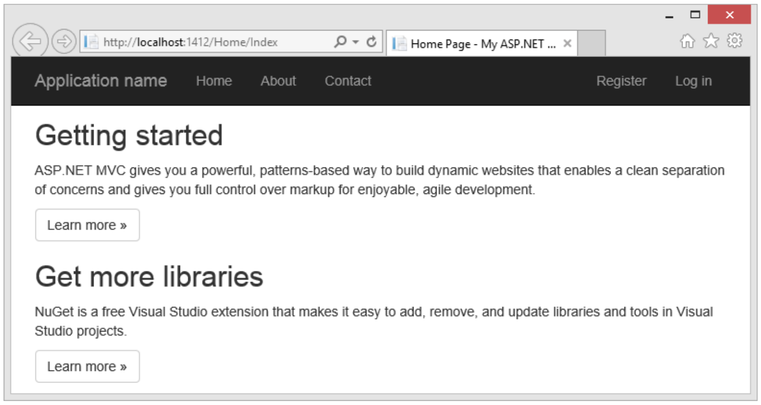
NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Learn more »

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

Learn more »



Getting started

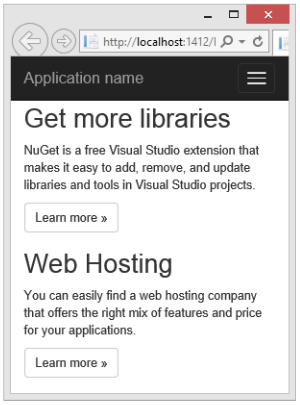
ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

Learn more »

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Learn more »



Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Learn more »

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

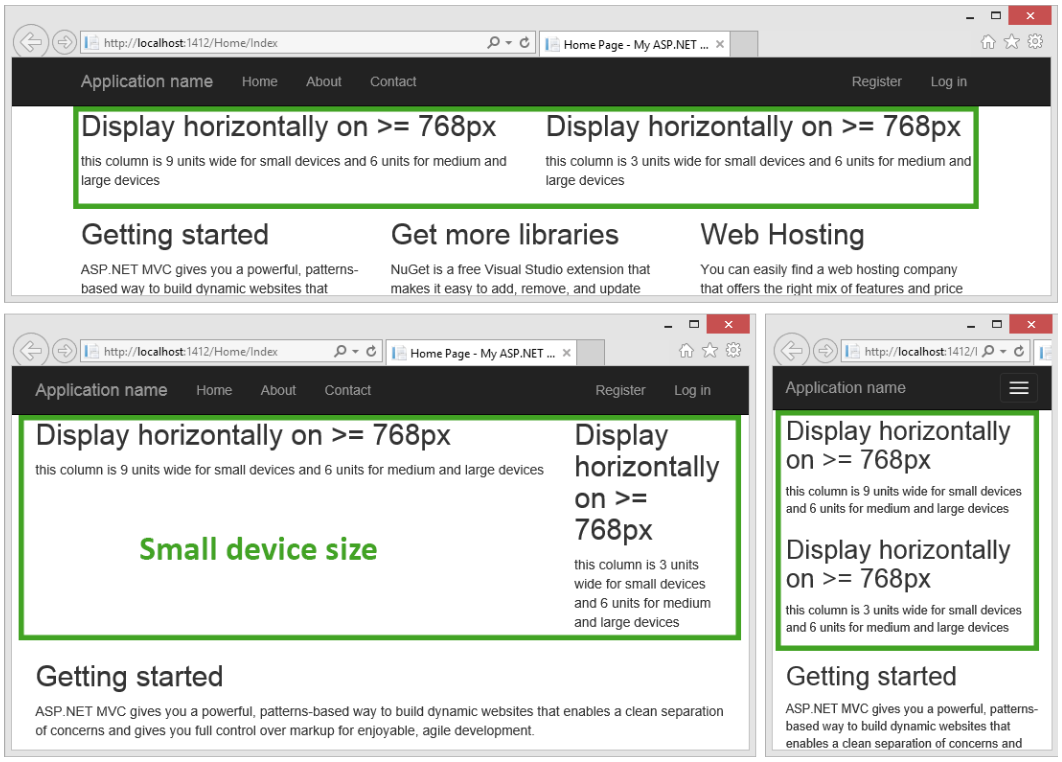
Learn more »

Combine multiple column definitions

It is possible to add more than one Bootstrap column class to an element effectively changing the design when the browser width changes. Let's say that you want to display horizontal content on small devices as well but the design should be different from the design on medium and large devices. This can be accomplished by adding a column definition for small devices overriding the default stacking design with your own horizontal design.

The following example shows how you can change the default column design for small devices to display two columns taking up 9 and 3 units respectively. The same content will be displayed on medium and large devices using two columns occupying 6 units each.

```
<div class="row">
    <div class="col-sm-9 col-md-6">
        <h2>Display horizontally on >= 768px</h2>
        <p>this column is 9 units wide for small devices and 6 units for
           medium and large devices</p>
    </div>
    <div class="col-sm-3 col-md-6">
        <h2>Display horizontally on >= 768px</h2>
        <p>this column is 3 units wide for small devices and 6 units for
           medium and large devices</p>
    </div>
</div>
```



Display horizontally on >= 768px

this column is 9 units wide for small devices and 6 units for medium and large devices

Display horizontally on >= 768px

this column is 3 units wide for small devices and 6 units for medium and large devices

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price

Small device size

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

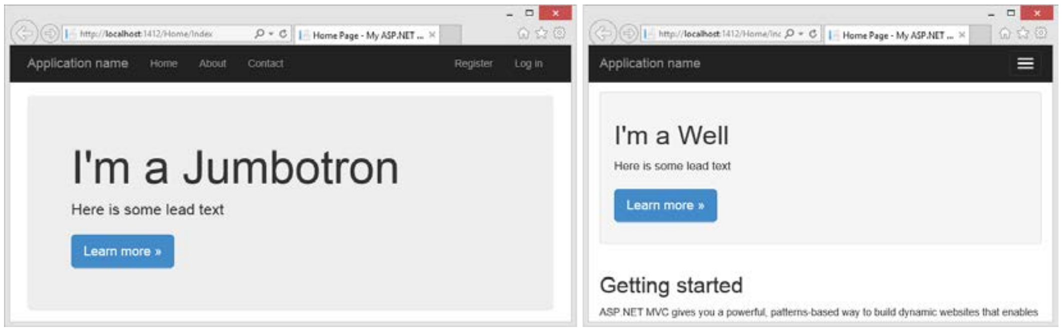
Respond.js

There are older browsers that don't support media queries such as IE6-IE8. To get responsive design to work with media queries in these browsers a JavaScript *polyfill* file called *respond.js* can be used; it will fill in the missing pieces making it possible for these browsers to use media queries.

Components

There are many classes in Bootstrap which can be applied to HTML elements to style them and make them more appealing to the user. You have already seen two components at work, the navigation bar at the top of the screen (more about that component later in this chapter) and the **Jumbotron** which is the gray area with the large text at the top of the page. The latter component is created by adding the **jumbotron** class to a **<div>** element and add some content to that **<div>**. The **well** class is similar to the **jumbotron** but the text will not be as pronounced.

```
<div class="jumbotron">
    <h1>I'm a Jumbotron</h1>
    <p class="lead">Here is some lead text</p>
    <p><a href="http://asp.net" class="btn btn-primary btn-lg">Learn more
&raquo;</a></p>
</div>
<div class="well">
    <h1>I'm a Well</h1>
    <p class="lead">Here is some lead text</p>
    <p><a href="http://asp.net" class="btn btn-primary btn-lg">Learn more
&raquo;</a></p>
</div>
```



ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables

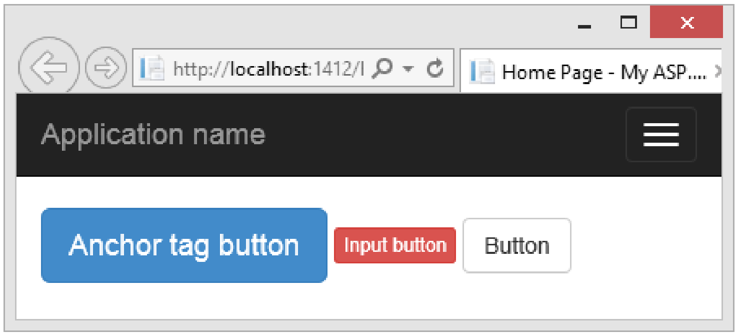
Buttons

Buttons are easy to style, all you need to do is to add the **btn** class and one of the button size classes **btn-xs**, **btn-sm**, **btn-md** or **btn-lg** to an element. To give a button color you can use one of the predefined "color" classes **btn-default**, **btn-success**, **btn-primary**, **btn-info**, **btn-warning** or **btn-danger**. Note that no color is specified in the class names, this is deliberate to make it easy to redefine their colors. In some design scenarios the primary color might not be the default blue but instead a color that fits the design theme of that web site. Instead of creating new CSS classes to define the colors, the designer can simply override the **btn-primary** class and change the color associated with that class in the entire theme. The previous image contain buttons defined with the **btn-primary** class.

The **btn** classes can be used with three HTML elements, the anchor tag **<a>**, the **<input>** element and the **<button>** element.

```
<a href="#" class="btn btn-primary btn-lg">Anchor tag button</a>
```

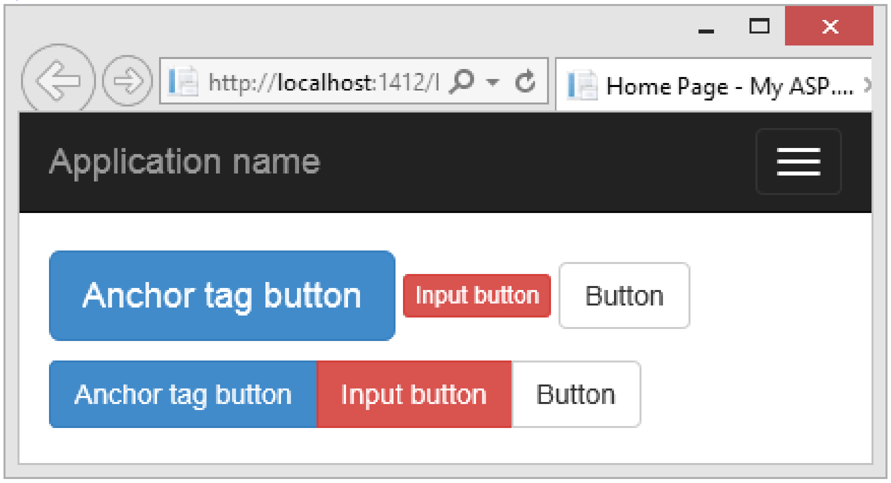
```
<input type="button" class="btn btn-danger btn-xs" value="Input button">
<button class="btn btn-default btn-md">Button</button>
```



If you want to display buttons as a group making them the same size instead of displaying them as separate buttons with space in between you can add a **<div>** element decorated with the **btn-group** class around the buttons.

As you can see in the image below the button group keeps the buttons together and adds rounded corners to the outermost buttons.

```
<div class="btn-group">
    <a href="#" class="btn btn-primary btn-md">Anchor tag button</a>
    <input type="button" class="btn btn-danger btn-md" value="Input
button">
    <button class="btn btn-default btn-md">Button</button>
</div>
```



Alert and Label

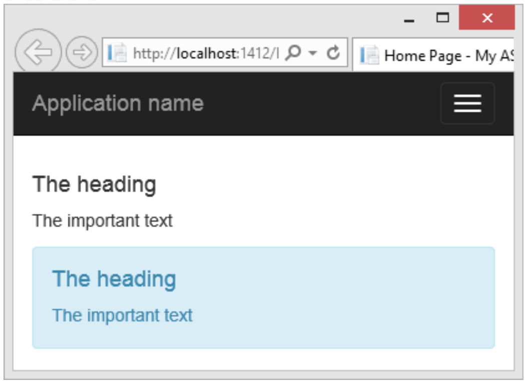
If you have content that you want to stand out you can either add the **alert** or the **label** class to the surrounding element. As you can see in the image below the **alert** can handle multiple lines of text and even components such as buttons, whereas the **label** is a single line of text.

The color is determined by the classes **alert-info**, **alert-danger**, **alert-success** or **alert-warning** for **alert** content or **label-info**, **label-danger**, **label-success** or **label-warning**, **label-primary** or **label-default** for **label** content.

```
@*Alert*@
<div>
   <h4>The heading</h4>
   <p>The important text</p>
</div>

<div class="alert alert-info">
   <h4>The heading</h4>
   <p>The important text</p>
</div>
```

```
@*Label*@
<p>The important text</p>

<p class="label label-danger">
    The important text
</p>
```

Alert



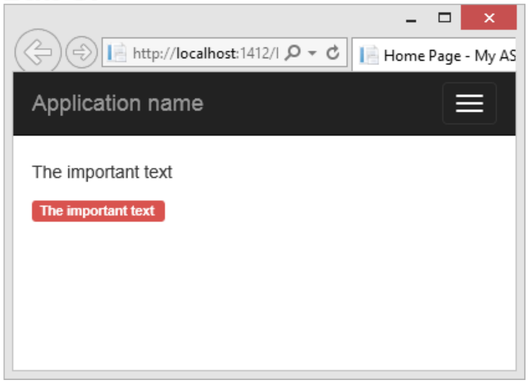
The heading

The important text

The heading

The important text

Label



The important text

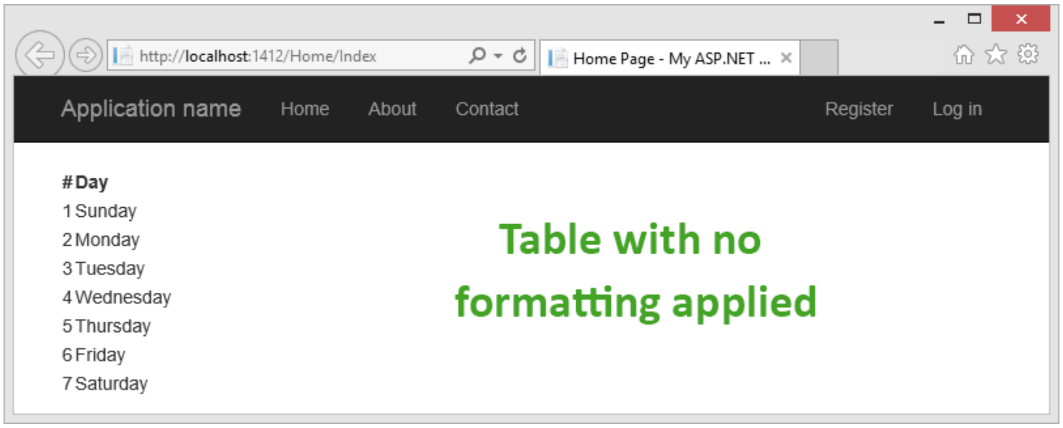
The important text

Tables

Tables can be a great way of displaying tabular data but with its default styling it leaves a lot to be desired. Luckily you can use simple Bootstrap classes to achieve wonders with your tables. In this section you will see some basic table styling.

The table below is a list of the weekdays where no Bootstrap classes have been added.

```
<table class="">
   @*Header*@
   <tr><th>#</th><th>Day</th></tr>

   @*Rows*@
   <tr><td>1</td><td>Sunday</td></tr>
   <tr><td>2</td><td>Monday</td></tr>
   <tr><td>3</td><td>Tuesday</td></tr>
   <tr><td>4</td><td>Wednesday</td></tr>
   <tr><td>5</td><td>Thursday</td></tr>
   <tr><td>6</td><td>Friday</td></tr>
   <tr><td>7</td><td>Saturday</td></tr>
</table>
```


http://localhost:1412/Home/Index
Home Page - My ASP.NET ...
Application name
Home
About
Contact
Register
Log in
Table with no formatting applied

#	Day
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

Basic table styling with Bootstrap

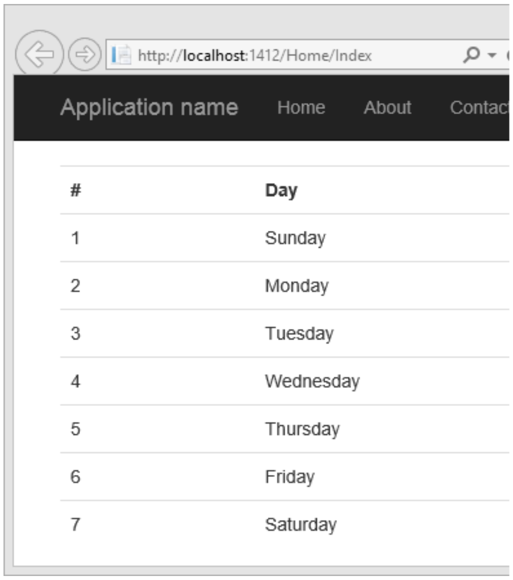
The first Bootstrap class you will add is the **table** class which adds some basic table styling making the columns more airy and displaying row line dividers.

```
<table class="table">
```

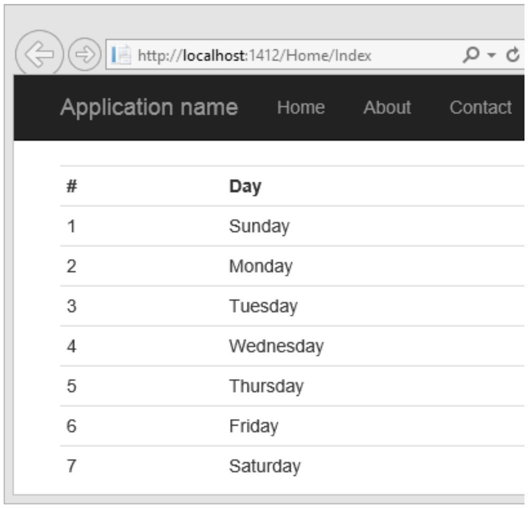
Condensed table

Add the **table-condensed** class to the table if you prefer a more condensed table layout where some of the padding between the lines have been removed.

```
<table class="table table-condensed">
```



#	Day
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday



#	Day
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

Hover effect

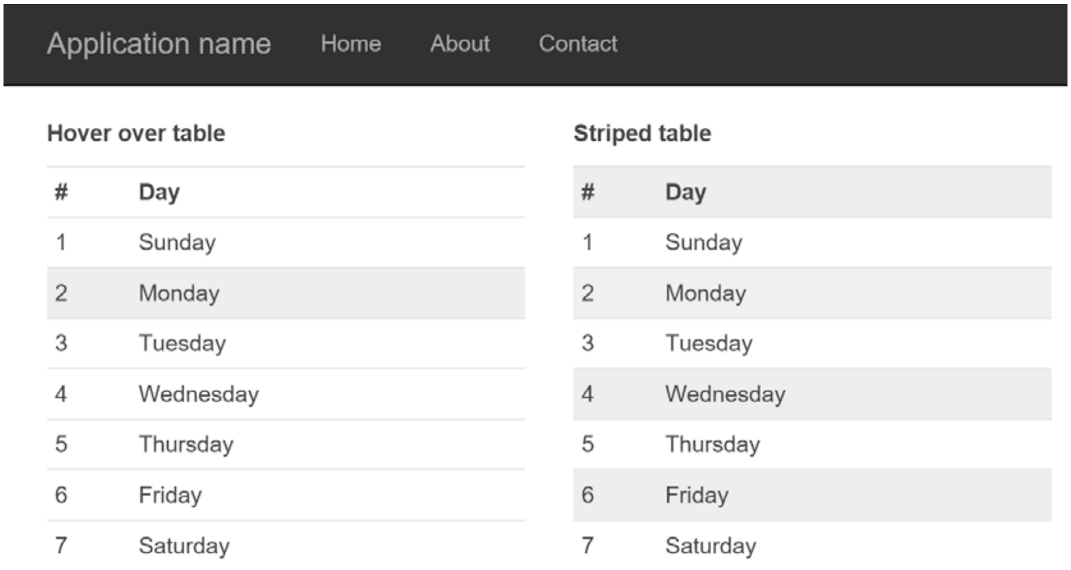
To make it easier for the user to keep track of the lines when reading the data you can add the **table-hover** class which highlights the row where the mouse pointer is hovering. The light gray color used for the hover effect might be hard to see in the image.

```
<table class="table table-condensed table-hover">
```

Striping effect

To make a table more readable you can create a striping effect by adding the **table-striped** class. The light gray color used for the striping effect on the even rows might be hard to see in the image.

```
<table class="table table-condensed table-striped">
```



Hover over table

#	Day
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

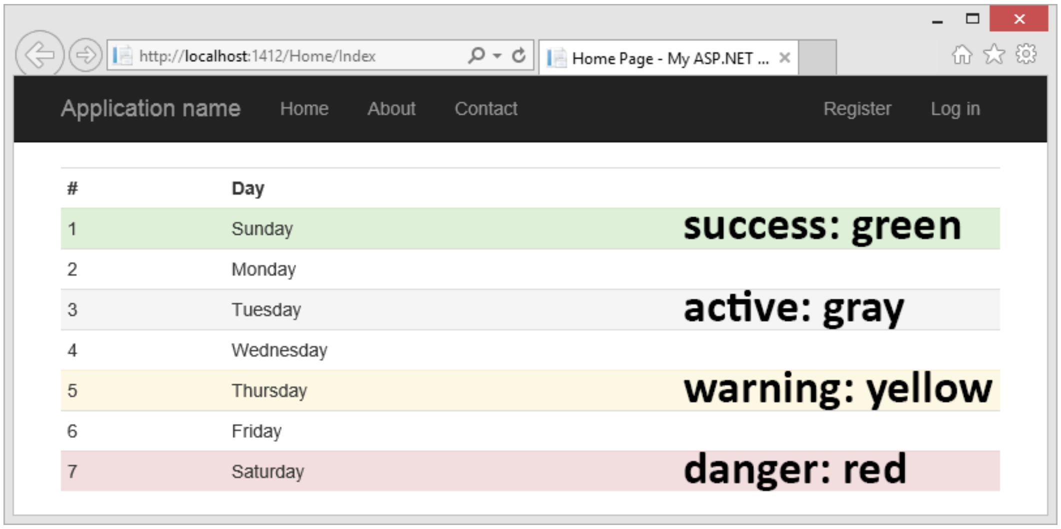
Striped table

#	Day
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

Colored rows

If you want a row in the table to have a colorized background one of the **warning**, **danger** or **success** classes can be applied or you can use the **active** class to denote that a specific row is the active row. If you are reading a printed version of this book the table row background colors might be hard to see.

```
<table class="table table-condensed">
   @*Header*@
   <tr><th>#</th><th>Day</th></tr>

   @*Rows*@
   <tr class="success"><td>1</td><td>Sunday</td></tr>
   <tr><td>2</td><td>Monday</td></tr>
   <tr class="active"><td>3</td><td>Tuesday</td></tr>
   <tr><td>4</td><td>Wednesday</td></tr>
   <tr class="warning"><td>5</td><td>Thursday</td></tr>
   <tr><td>6</td><td>Friday</td></tr>
   <tr class="danger"><td>7</td><td>Saturday</td></tr>
</table>
```



#	Day	
1	Sunday	success: green
2	Monday	
3	Tuesday	active: gray
4	Wednesday	
5	Thursday	warning: yellow
6	Friday	
7	Saturday	danger: red

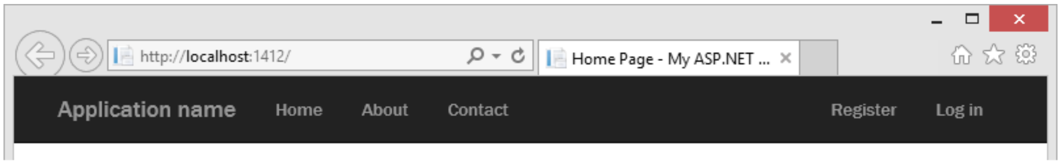
Navigation bar

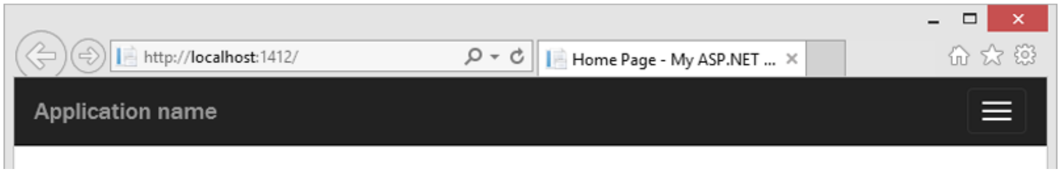
The navigation bar is the menu that sticks to the top of the browser. Let's have a look at the **navbar** classes and find out how this is possible.

The first thing you should note is that if you make the browser small enough the regular menu is replaced by a mobile friendly menu affectionately called *the hamburger menu*; clicking the button displays the menu. This is available out of the box when adding the *bootstrap.css* file to the project.

The default implementation of the **navbar** uses three classes. The **navbar** class specifies that the **<div>** element should be used as a navigation bar, the **navbar-inverse** specifies that the color should be the inverse of the page background and the **navbar-fixed-top** makes the **navbar** stick to the top of the page even when users scroll the page content.

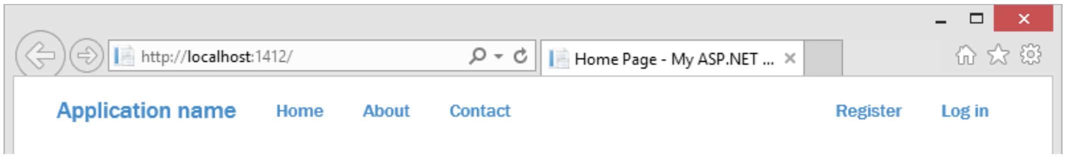
```
<div class="navbar navbar-inverse navbar-fixed-top">
```





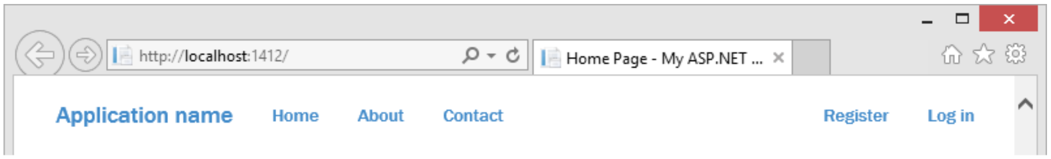
When removing the **navbar-inverse** class the **navbar** is no longer displayed with the inverse background color.

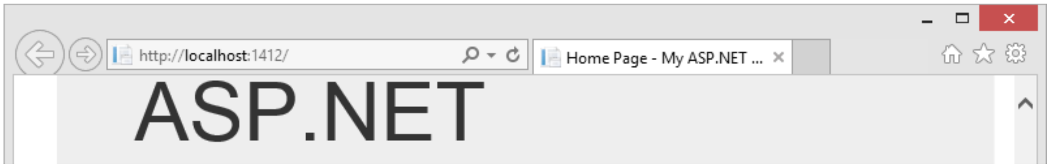
```
<div class="navbar navbar-fixed-top">
```



When removing the **navbar-fixed-top** class the **navbar** will scroll out of view when the user scrolls the page content.

```
<div class="navbar">
```





The structure

The **navbar** is a fairly advanced control which has several areas you need to keep track of as you build or modify a menu. Below is a quick look at the structure needed to create a rich navigation experience for your users. In an upcoming exercise you will get firsthand experience modifying and creating menus using the **navbar**.

The outermost menu element must have at least the **navbar** class present for it to act as a navigation menu. An element with the **container** class applied must be added to the **navbar** element, this element will act as the container for the menu header and the actual drop-down menu.

```
<div class="navbar navbar-inverse navbar-fixed-top">
   <div class="container">
```

The first element inside the **container** element is the **navbar-header** which contains the non-menu item parts of the menu such as the *hamburger menu* mentioned earlier and also the **navbar-brand** area where you specify the name of your brand, company or site.

Adding the **navbar-toggle** class and two **data-** attributes to a **<button>** element makes it possible to open and close the *hamburger menu* when the menu is viewed on a device with a small screen.

The three **icon-bar <span>** elements are the three lines displayed on the *hamburger menu* button.

```
      <div class="navbar-header">
         <button type="button" class="navbar-toggle"
            data-toggle="collapse" data-target=".navbar-collapse">
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
         </button>
         <a class="navbar-brand" href="#home">
            <img src="Content/fox-logo 100px.png" />
         </a>
      </div>
```

For the **navbar** to act as a menu a second area where the actual menu items are placed must be added. This area must be represented by an element with the **nvabar-collapse** and **collapse** classes defined, these two classes makes it possible to open and close the menu when the *hamburger menu* is clicked.

An unordered list (**<ul>**) inside the menu area will contain the menu items (**<li>**). The unordered list must be defined with the **nav** and **navbar-nav** classes to act as a menu item container. You can use nested items if you want to create a drop-down menu for one of the top-level menu items.

```
        <div class="navbar-collapse collapse">
           <ul class="nav navbar-nav">
              <li><a href="#home">Home</a></li>
              <li><a href="#images">Images</a></li>
              <li><a href="#contact">Contact</a></li>
              <li><a href="#about">About</a></li>
           </ul>
        </div>
     </div> @*End of container*@
  </div> @*End of navbar*@
```

Exercise 5: Build Foxy Foxes web page with Bootstrap & CSS

In this exercise you will use Bootstrap and CSS to style the HTML5 elements in the Index.html document for the Foxy Foxes web page. Install Bootstrap to a newly created empty ASP.NET Web Application project and add links to the necessary Bootstrap and JQuery files to a new HTML page.

Adding CSS, Bootstrap and JQuery to the web project

The JavaScript files are necessary to get the "hamburger menu" button to work when the web page is displayed on extra small devices.

1. Create a new empty ASP.NET Web Application project.
2. Right click on the project name in the Solution Explorer and select **Manage NuGet Packages**.
3. Type "*Bootstrap*" into the search field.
4. Click on the **Install** button for the **Bootstrap** package.
5. Right click on the project name in the Solution Explorer and select **Add-HTML Page**.
6. Name the page **Index** and click the **OK** button.
7. Drag the **bootstrap.min.css** file to the HTML page and place it at the end of the **<head>** element.

8. Drag the **jquery-*x.y.z*.min.js** file to the HTML page and place it below the **bootstrap.min.css** link.
9. Drag the **bootstrap.min.js** file to the HTML page and place it below the **jquery-*x.y.z*.min.js** link.
10. Right click on the **Content** folder and add a CSS file called **site.css** to it.
11. Drag the **site.css** file to the HTML page and place it below the **bootstrap.min.css** link. This CSS file will be used to style the HTML elements where Bootstrap isn't enough.

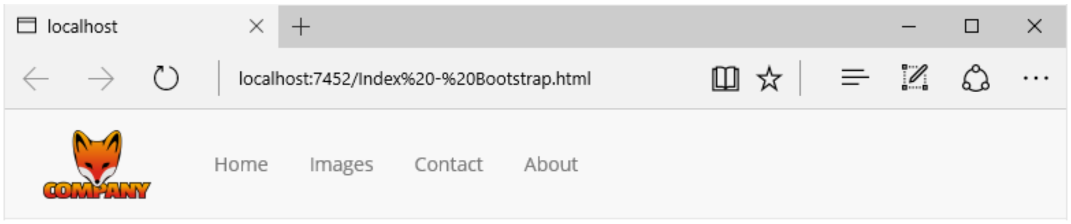
The content of the **Index.html** page so far:

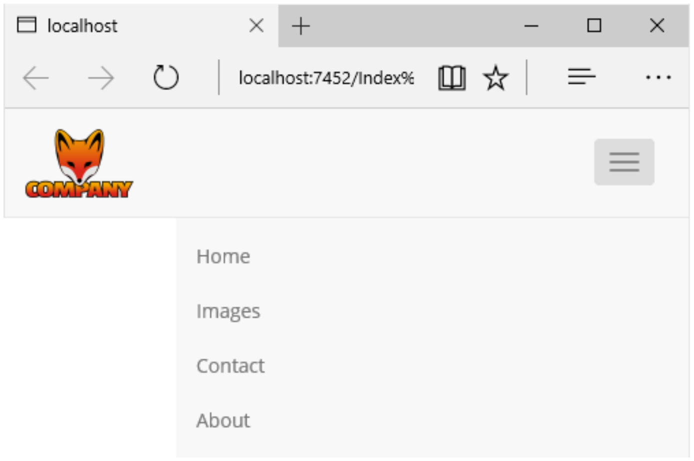
```
<!DOCTYPE html>
<html>
   <head>
      <title></title>
      <meta charset="utf-8" />
      <link href="Content/bootstrap.min.css" rel="stylesheet" />
      <link href="Content/site.css" rel="stylesheet" />
      <script src="Scripts/jquery-1.9.1.min.js"></script>
      <script src="Scripts/bootstrap.min.js"></script>
   </head>
   <body>
   </body>
</html>
```

Adding the navigation

Use a **<nav>** element to create the navigation menu bar fixed to the top of the web page; it should be responsive and the menu options should change to a drop-down "hamburg-er menu" button when displayed on extra small devices such as smart phones. The four menu items **Home**, **Images**, **Contact** and **About** should jump to various sections of the page when clicked just as in the previous exercise. The menu should look and work as in the previous exercise but be implemented with Bootstrap classes.

Give the **<body>** a narrower width than the actual browser area by applying the **container** CSS class; the width will wary depending on the screen resolution.





Adding the navigation bar

1. Add the **container** CSS class to the **<body>** element. This will only restrict the width of the content area and will not affect the navigation.
2. Add a **<header>** element inside the **<body>** element.
3. Add a **<nav>** element inside the **<header>** element.
4. Add the **navbar**, **navbar-default** and **navbar-fixed-top** Bootstrap classes to the **<nav>** element to turn it into a navigation bar with a light gray background that stays at the top of the page even when scrolling.
   ```
   <nav class="navbar navbar-default navbar-fixed-top">
   ```
5. Add the navigation header section which will contain the logo and the "Hamburger button". This **<div>** must be decorated with the **navbar-header** class.
   ```
   <div class="navbar-header">
   ```
6. Add a **<button>** element and decorate it with the **navbar-toggle** and **collapsed** classes to denote that it should work as a toggle button (on/off) and that its content should be collapsed (hidden) by default. To be able to open and close

the menu the **data-toggle="collapse"** attribute must be added as well as a **data-tartget="*#id-of-collapsible-element*"** which connects the button with the collapsible element. The **aria-expanded** attribute keeps track of if the content should be expanded or collapsed.

```
<button type="button" class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#data-navbar-collapse" aria-
expanded="false">
```

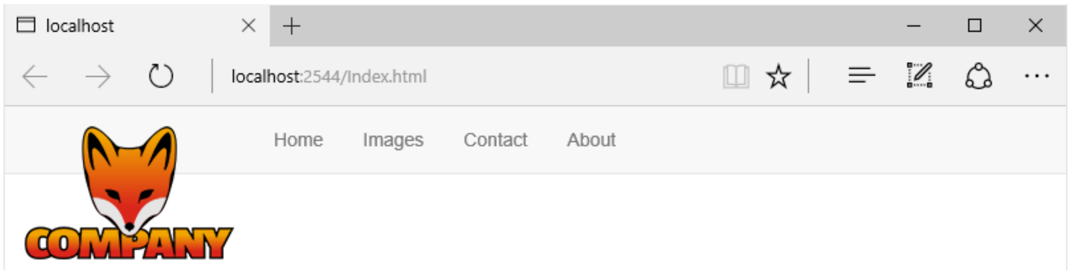
7. Add three **<span>** elements decorated with the **icon-bar** class inside the **<button>** element.
8. Drag the fox logo image to the **Content** folder from Windows Explorer.
9. Add an anchor tag (**<a>**) below the **<button>** element and decorate it with the **navbar-brand** class to position it correctly in the navigation area. Also add the **#home** navigation id to its **href** attribute to make it possible to click the logo to return to the top of the page.
10. Drag the fox logo image to the **<a>** element from the solution explorer.
11. Add a **<div>** element below the **<div>** decorated with the **navbar-header** class. The new **<div>** should be decorated with the **collapse** and **navbar-collapse** classes to turn it into a collapsible area. It also must have an id matching the id specified in the **data-target** attribute of the **<button>** element to connect them.

```
<div class="collapse navbar-collapse" id="data-navbar-collapse">
```

12. The menu items are defined by an unordered list added to the previously added **<div>**. The **<ul>** element has to be decorated with the **nav** and **navbar-nav** classes to turn it into a list of menu items. The menu items should navigate to the different areas of the page.

```
<div class="collapse navbar-collapse" id="data-navbar-collapse">
    <ul class="nav navbar-nav">
        <li><a href="#home">Home</a></li>
        <li><a href="#images">Images</a></li>
        <li><a href="#contact">Contact</a></li>
        <li><a href="#about">About</a></li>
    </ul>
</div>
```

13. Right click on the **Index.html** page in the Solution Explorer and select **View in Browser**. As you can see the logo is too large for the navigation bar so the next step will be to style the navigation bar with CSS.
14. Switch back to Visual Studio.



The complete code for the navigation bar:

```
<body class="container">
    <header>
        <nav class="navbar navbar-default navbar-fixed-top">
            <div class="navbar-header">
                <!--Hamburger button-->
                <button type="button" class="navbar-toggle collapsed"
                data-toggle="collapse" data-target="#data-navbar-collapse"
                aria-expanded="false">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="#home">
                    <img src="Content/fox-logo 100px.png" />
                </a>
            </div>

            <!--Links-->
            <div class="collapse navbar-collapse"
            id="data-navbar-collapse">
                <ul class="nav navbar-nav">
                    <li><a href="#home">Home</a></li>
                    <li><a href="#images">Images</a></li>
                    <li><a href="#contact">Contact</a></li>
                    <li><a href="#about">About</a></li>
                </ul>
            </div>
        </nav>
    </header>
</body>
```

Styling the navigation bar with CSS

There are a few things in the navigation bar that need CSS styling. The logo is much too large and needs to be reduced in size, let's make it half the size (50px). As the navigation bar still will be too narrow for the height of the image its height have to be increased to 80px. Altering the height will offset the menu items making it necessary to move them down to center them vertically, the same goes for the "Hamburger menu" button which also could benefit from being moved slightly to the left. Because the drop-down menu is transparent it needs the same solid background color as the navigation bar. The menu items could be made to "pop" by changing the text color when hovering over them. Use the **F12** tool to pick up the colors from the logo and the navigation bar.

Change the font family to Google's **Open Sans** for the whole page. Go back to previous exercises if you have forgotten how to get the link for the font family.

Many of the CSS selectors are more specific than they have to be below to make it easier to see where in the DOM structure they will be applied.

I suggest that you do the changes incrementally and watch the changes in the web browser as you implement them.

1. Open the **Index.html** page.
2. Add the link the **Open Sans** font family (copied from the https://www.google.com/fonts page) to the **<head>** element in the **Index.html** page.
3. Open the **site.css** style sheet.
4. Use the **Open Sans** font family in the **body** CSS rule.
```
body {
    font-family: 'Open Sans', sans-serif;
}
```
5. Add a selector for the **<nav>** element and change the height of the navigation bar to 80px.
```
body > header > nav {
    height: 80px;
}
```
6. Change the logo size to 50px by targeting the **<img>** element. This selector has to be more specific since there will be more images on the page.

```
body > header > nav img {
    height: 50px;
}
```

7. Give the menu items a 15px top and 110px left margin. You can target the **<div>** element containing the menu by its id.

```
body > header > nav #data-navbar-collapse {
    margin-top: 15px;
    margin-left: 110px;
}
```

8. Change the hover color to orange for the menu items. Remember to target the **<a>** elements not the **<li>** elements.

```
body > header > nav #data-navbar-collapse li a:hover {
    color: rgba(235, 139, 6, 1);
}
```

9. Position the "Hamburger menu" button 10px from the right border and give it a 22px top margin.

```
body > header > nav .navbar-header button {
    margin-top: 22px;
    right: 10px;
}
```

10. Give the drop-down menu the same background color as the navigation bar. You can target its **<div>** container.

```
body > header > nav div {
    background-color: #f8f8f8;
}
```

11. Open the **Index.html** page in the browser and make sure that the changes have been applied as specified and that the drop-down menu works.

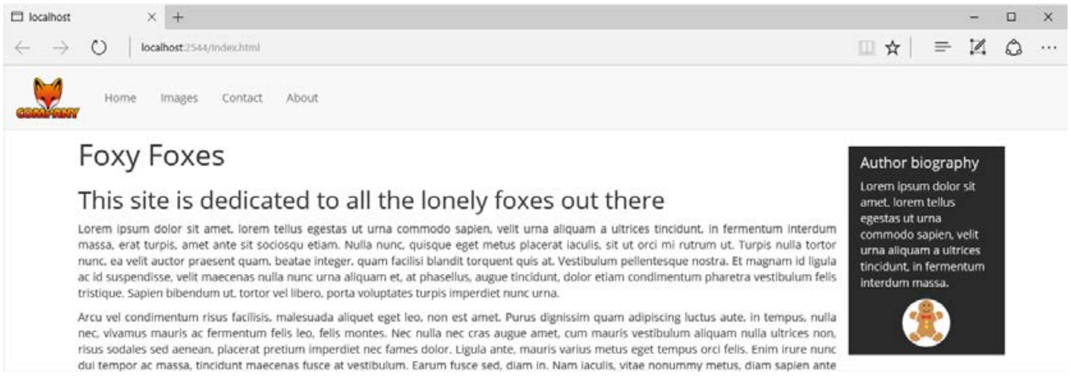
Adding the home area

The **<section>** element and its content is the same as in the previous exercise with the exception that you might want to change the **<h1>** tag in the **<aside>** element to a **<h4>** heading to make it a bit smaller to look more aesthetically pleasing.

To make it conform to the Bootstrap grid you will also have to add the **row** class to the **<section>** element and the **col-sm-10** class to the **<article>** and **col-sm-2** class to the

<aside> elements to create the aside effect without further CSS styling. You will however still have to style the image in the **<aside>** element with CSS.

Without any added CSS styling part of the **<section>** element is hidden behind the navigation bar because you changed the height of the bar. To fix this you can add padding to all the **<section>** elements to display the text in the **<h1>** elements when navigating.


localhost:2544/index.html

Home Images Contact About

Foxy Foxes

This site is dedicated to all the lonely foxes out there

Lorem ipsum dolor sit amet. lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum massa, erat turpis, amet ante sit sociosqu etiam. Nulla nunc, quisque eget metus placerat iaculis, sit ut orci mi rutrum ut. Turpis nulla tortor nunc, ea velit auctor praesent quam, beatae integer, quam facilisi blandit torquent quis at. Vestibulum pellentesque nostra. Et magnam id ligula ac id suspendisse, velit maecenas nulla nunc urna aliquam et, at phasellus, augue tincidunt, dolor etiam condimentum pharetra vestibulum felis tristique. Sapien bibendum ut. tortor vel libero, porta voluptates turpis imperdiet nunc urna.

Arcu vel condimentum risus facilisis, malesuada aliquet eget leo, non est amet. Purus dignissim quam adipiscing luctus aute, in tempus, nulla nec, vivamus mauris ac fermentum felis leo, felis montes. Nec nulla nec cras augue amet, cum mauris vestibulum aliquam nulla ultrices non, risus sodales sed aenean, placerat pretium imperdiet nec fames dolor. Ligula ante, mauris varius metus eget tempus orci felis. Enim irure nunc dui tempor ac massa, tincidunt maecenas fusce at vestibulum. Earum fusce sed, diam in. Nam iaculis, vitae nonummy metus, diam sapien ante

Author biography

Lorem ipsum dolor sit amet. lorem tellus egestas ut urna commodo sapien, velit urna aliquam a ultrices tincidunt, in fermentum interdum massa.

1. Copy the **<section>** element with the *home* id and all its content from the previous exercise and paste it into the **Index.html** page.
2. Add the Bootstrap **row** class to the **<section>** element.
```
<section id="home" class="row">
```
3. Add the **col-sm-10** class to the **<article>** element.
```
<article class="col-sm-10">
```
4. Add the and **col-sm-2** class to the **<aside>** element.
```
<aside class="col-sm-2">
```
5. Change the **<h1>** tag in the **<aside>** element to a **<h4>** heading.
```
<h4>Author biography</h4>
```
6. If you haven't already copied the circular image displayed in the **<aside>** area to the **Content** folder then do so now.
7. View the web page in the browser. You will see that it needs a bit of CSS styling to look perfect.

Styling the home section with CSS

First you need to add 70px top padding and justify the text in the **<section>** elements.

Next you need to style the **<aside>** element. Align the text to the left, add 10px padding to the bottom to keep the image from being placed flush to the bottom of the element, Add 30px top margin to give the text some breathing room, change the back color to a dark gray (#2a2a2a) and give the text a very light gray, almost white, color (#f7f7f7).

To center the image it needs to be displayed as a **block** and have its left and right margins set to **auto**. You might also want to change its size to 60px to make it a bit smaller.

1. Open the **site.css** style sheet.
2. Add a selector for the **<section>** element. Add 70px top padding to move the heading down and justify the text.
```
body > section {
    padding-top: 70px;
    text-align: justify;
}
```
3. Add a selector for the **<aside>** element. Align the text to the left, add 10px bottom padding for the image, 30px top margin for the heading, change the background color to a dark gray and the text color to a light gray.
```
#home aside {
    text-align: left;
    padding-bottom: 10px;
    margin-top: 30px;
    background-color: #2a2a2a;
    color: #f7f7f7;
}
```
4. Add a selector for the image inside the **<aside>** element. Display the image as a block element for the **auto** left and right margins to work properly. Change the size of the image to have a 60px width.
```
#home aside img {
    display: block;
    margin-left: auto;
    margin-right: auto;
    width: 60px;
}
```
5. Save the files and view the page in a browser to make sure that the styling has been applied.

Adding the images area

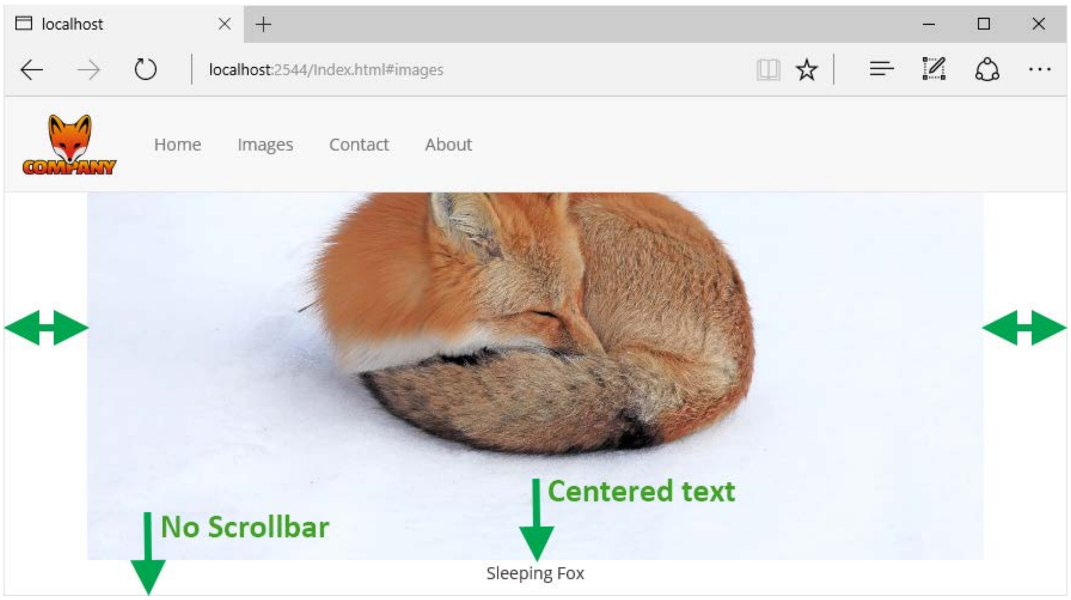
The **<section>** element with the *images* id and its content is the same as in the previous exercise.

To make it conform to the Bootstrap grid you will have to add the **row** class to the **<section>** element.

1. Copy the **<section>** element with the *images* id and all its content from the previous exercise and paste it into the **Index.html** page.
2. Add the Bootstrap **row** class to the **<section>** element.
   ```
   <section id="images" class="row">
   ```
3. If you haven't already copied the large fox images to the **Content** folder then do so now.
4. View the web page in the browser. You will see that it needs a bit of CSS styling to look perfect.

Styling the home section with CSS

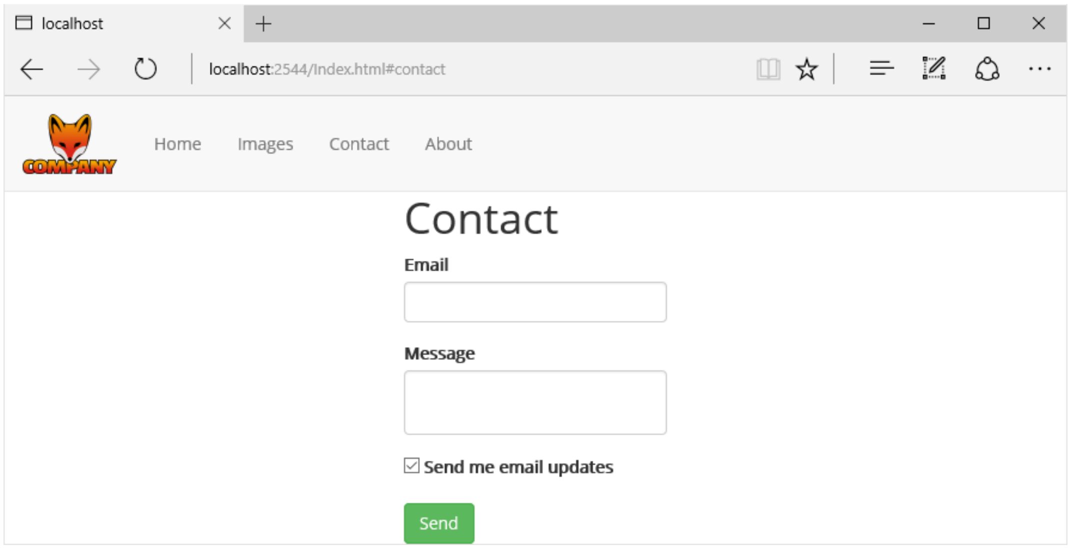
First you need to change the image width to 100% of its container width to avoid a horizontal scrollbar, then you need to center the image descriptions.



1. Open the **site.css** style sheet.
2. Add a selector for the **<img>** element in the *images* **<section>** element. Change the image width to 100%.
   ```
   #images img {
      max-width: 100%;
   }
   ```
3. Add a selector for the **<article>** element in the *images* **<section>** element. Center the text for the image descriptions.
   ```
   #images article {
      text-align: center;
   }
   ```
4. Save the files and view the page in a browser to make sure that the styling has been applied.

Adding tha contact area

Add a **<section>** element and give it the id *contact*. Divide the row into 3 equally sized areas using the **col-sm-4** class. Add an **<h1>** element with the text "*Contact*" to the middle area and a **<form>** element below it. Use the **form-group** class to group the labels with their respective control. The form should contain the same controls and labels as the form in the previous exercise. The submit button should be decorated with the **btn-success** class to give it a green color and the controls should be decorated with the **form-control** class to give them a nicer appearance.



1. Open the **Index.html** file from the Solution Explorer.
2. Add a **<section>** tag and give it the id *contact* and decorate it with the **row** class.
   ```
   <section id="contact" class="row">
   ```
3. Add three **<div>** elements and decorate them with the **col-sm-4** class.
   ```
   <div class="col-sm-4"></div>
   ```
4. Add an **<h1>** element with the text "*Contact*" and a **<form>** element to the middle **<div>**.
   ```
   <h1>Contact</h1>
   <form></form>
   ```
5. Add three **<div>** elements decorated with the **form-group** class inside the **<form>** element.
   ```
   <div class="form-group">
   ```
6. Add a **<label>** element with the text "*Email*" to the first of the three **<div>** elements and decorate it with the **for** attribute connecting it to the textbox control you will add next through its **email** id.
   ```
   <label for="email">Email</label>
   ```
7. Add an **<input>** element and give it the id **email**. Decorate it with the **email type** attribute and the **form-control** class.
   ```
   <input type="email" class="form-control" id="email" />
   ```

8. Repeat step 5-7 for a label with the text "*Message*" and an **<input>** element with the id **message**.
9. Repeat step 5-7 for a checkbox with the id **updates** and a label with the text "*Send me email updates*".
10. Add a submit button decorated with the **btn** and **btn-success** classes below the last of the three **<div>** elements in the form.
    ```
    <input type="submit" value="Send" class="btn btn-success" />
    ```
11. Open the web page in a browser and make sure that the form is displayed in the center of the page.

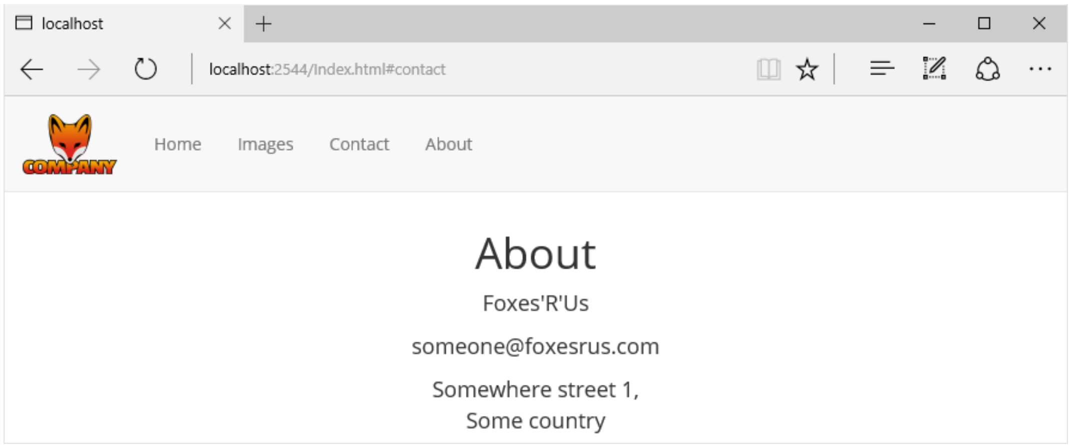
The complete **contact** section code:

```
<section id="contact" class="row">
   <div class="col-sm-4"></div>
      <div class="col-sm-4">
         <h1>Contact</h1>
         <form>
            <div class="form-group">
               <label for="email">Email</label>
               <input type="email" class="form-control" id="email" />
            </div>
            <div class="form-group">
               <label for="message">Message</label>
               <textarea id="message" class="form-control"></textarea>
            </div>
            <div class="form-group">
               <input type="checkbox" checked id="updates" />
               <label for="updates">Send me email updates</label>
            </div>
            <input type="submit" value="Send"
            class="btn btn-success" />
         </form>
      </div>
   <div class="col-sm-4"></div>
</section>
```

Adding the about area

The *about* **<section>** element with its content is the same as in the previous exercise.

To make it conform to the Bootstrap grid you will also have to add the **row** class to the **<section>** element and add one **<div>** element above and one below the **<article>** element; decorate the elements with **col-sm-4** class.



About

Foxes'R'Us

someone@foxesrus.com

Somewhere street 1,
Some country

1. Copy the *about* **<section>** element with its content from the previous exercise and paste it in below the **contact** area.
2. Add the **row** class to the **<section>** element.
   ```
   <section id="about" class="row">
   ```
3. Decorate the **<article>** element with **the col-sm-4** class.
   ```
   <article class="col-sm-4">
   ```
4. Add a **<div>** decorated with the **col-sm-4** class above the **<article>** element.
   ```
   <div class="col-sm-4"></div>
   ```
5. Add a **<div>** decorated with the **col-sm-4** class below the **<article>** element.
6. Open the web page in a browser. Note that the text isn't centered, the next step will be to center the text.

Styling the about text with CSS

The text should be centered and have a font size of 18px.

1. Open the **site.css** style sheet.
2. Add a selector which targets the *about* **<section>** by id. Center the text and make it 18px.
   ```
   #about {
   ```

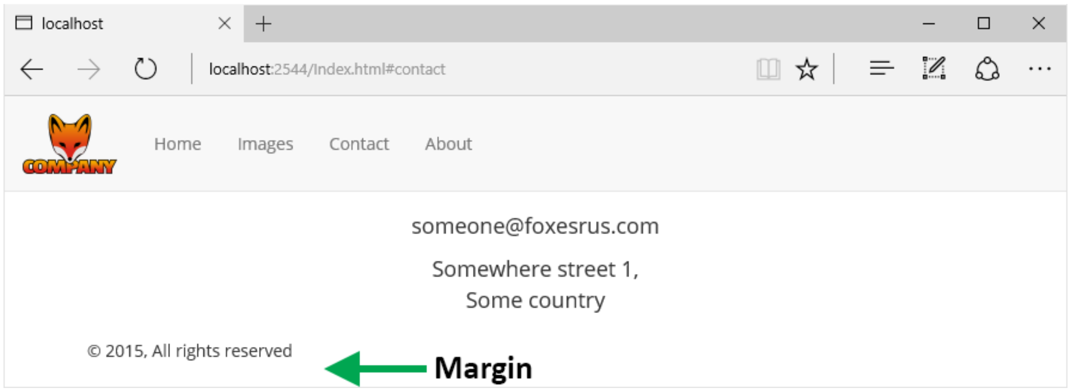
```
        text-align: center;
        font-size: 18px;
    }
```

3. Save the files and view the web page in a browser. Make sure that the text is centered and that the text has the right size.

Adding a footer

The **footer** element and its content is the same as in the previous exercise.

To make it conform to the Bootstrap grid you will also have to add the **row** class to the **<footer>** element.



someone@foxesrus.com

Somewhere street 1,
Some country

© 2015, All rights reserved

1. Copy the **<footer>** element with its content from the previous exercise and paste it in below the *about* area.
2. Add the **row** class to the **<footer>** element.
```
<footer class="row">&copy; 2015, All rights reserved</footer>
```
3. Open the web page in a browser. Note that the text is flush to the bottom of the page, the next step will be to add a bottom margin.

Styling the footer with CSS

The footer should have a 20px bottom margin to give it some breathing room.

1. Open the **site.css** style sheet.
2. Add a selector which targets the **<footer>** element and add a 20px bottom margin.

```
body > footer {
    margin-bottom: 20px;
}
```

3. Save the files and view the web page in a browser. Make sure that the footer text has a bottom margin.