vodafone

Girls coding course programme

Introduction to web development

Course Notes

Prepared for Vodafone by

CODEFIRST:Girls

Contents

Course overview

The course aims to provide a basic overview of the technologies used, along with the tools and resources to discover more.

The focus of this course is learning the basics of how and why things work and to provide the basis to build upon in future courses.

Sessions will be as hands-on and practical as possible. The notes provided will give you and the students a good resource to read through and base your lessons on. Try to be as interactive as possible.

The course curriculum overview is laid out below:

Getting going: Background to the Web and the Internet

- Welcome

Installations

Introduction to coding

- Ch.1 - What makes a website?
- Ch.2 - How does the Internet work?
- Ch.3 - How the Web Works

Part 1: HTML

- Ch.4 - Creating a basic HTML page
- Ch.5 - HTML Basics

Part 2: CSS

- Ch.6 - What is CSS?
- Ch.7 - Linking your HTML & CSS code
- Ch.8 - Start writing some CSS & basic definitions
- Ch.9 - Selectors & Attributes
- Ch.10 - Using the ID & class selectors
- Ch.11 - HTML <divs> and <spans>
- Ch.12 - The Universal selector

Part 3: Introduction to UX

- Ch.13- User experience

Part 4: Course competition

- Ch.14 - Competition criteria

Part 5: GitHub & Version control, Git

- Ch.15 – Version control & using GitHub
- Ch.16 – Create your group repository
- Ch.17 – Publishing on GitHub pages
- Ch.18 – Conflict scenario!

Part 6: Twitter Bootstrap

- Ch.19 – What's hard about CSS?
- Ch.20 – Twitter Bootstrap
- Ch.21 – Bootstrap Layout
- Ch. 22 – More Bootstrap
- Ch. 23 – Modifying Bootstrap

Part 7: JavaScript & jQuery

- Ch. 24 – JavaScript Preamble
- Ch. 25 – JavaScript & jQuery

Competition Guidelines

As part of this course, you will be creating your own website which will be entered into a course competition at the end of your course week! The criteria we'll be looking at are as follows. You can see the **Must have** criteria and the **Nice to have**. It would be great to have as many Nice to have features as possible, but don't worry we know you won't have time to include it all! Anything you don't get time to include now, can always be added later on.

Must have:

- A live website published on GitHub pages
- A minimum of **two** HTML files for:
    - 1 x home page/landing page linked to a separate CSS file
    - 1 x 'about' page
- A minimum of **one** CSS files
- Good formatting
    - Code split into the appropriate files (separate HTML files & CSS files)
    - Files indented properly
- Good organisation
- Version control using git
- Sensible git commit messages

Nice to have:

- A visually appealing design - good use of CSS and HTML elements, Twitter Bootstrap, jQuery & JavaScript (don't worry you'll learn about these last three topics next week!)
- A contact form (for example name and email)
- Social buttons
- As many different HTML elements you can manage
- Interactive elements (like forms) on your website don't need to be functional, but should be present if they need to be for the visual aspect of the design.
- A responsive site

So that's it! Other than that, you can make any kind of website you want.

You can see a few examples of what previous students on your course created on our website here: http://www.codefirstgirls.org.uk/course-competition.html

If you're short on ideas, here are some to get you going...

- A personal website

- "How to" website on an area of your expertise
- A survey or poll website.

The websites will be judged by your instructors at the end of the last session.

Welcome

It's an exciting time to learn coding! Technology is now ubiquitous, and has become the most accessible toolbox for progress in our society. (And the more pragmatic amongst us might appreciate the resultant demand for technical talent.)

A lot of us have grown up seeing the effects of technological advancement, many of us only as consumers. If you're reading this, it's probably safe to assume that you understand why it's important to learn to code, and have some idea of the things you can do with it.
We're going to equip you with the tools to push you in the right direction in development, and we hope to stretch your imagination of the things you can create with code. It's time to take a peek under the hood, and understand the creation of the products and tools so prevalent in our lives today.

Coding is both a science and an art. A bit like cooking; a chef will follow a recipe, and each type of dish requires certain ingredients, but the end result is subjective and contains a little bit of your personality. Learning to cook might seem intimidating at first, but once you try a recipe to make your favourite food (or a part of it), it all becomes a little easier and much more fun.

There are probably a number of questions that might come to mind when you first start coding, we'll try to answer some of them right now; the necessary foundational knowledge you need to know.

1. What is coding? Is it different from programming? Is it hard? What do developers do? Are they nice?
2. What will we be learning?
3. What can I do with the skills I will learn in this course?
4. How should I learn coding? (This is synonymous with is there a "best" way to learn coding?)
5. Is there anything I can do to prepare for the course?

NOTE:

DEMOS will be in blue with a salmon background - Like this. Your instructor will show them to do this as you talk through it. TASKS will be in red with a pale-blue highlighted background - Like this.

How the course will be run

The hardest thing about learning to program is knowing where to start and what to learn.
The course aims to provide a basic overview of the technologies used, along with the tools and resources to discover more.
The focus of this course is learning the basics of how and why things work and to provide the basis to build upon in future courses. We will not be able to cover everything in great depth or comprehensive detail.
Sessions will be as hands-on and practical as possible. Each week there will be a number of tasks to do in between the sessions to reinforce what you have learnt.
It's up to you whether you do the tasks or not, but the more you put in the more you will get out! If you're ever in doubt, Google it, check Stack Overflow, ask the person beside you, or ask one of us!
These notes are super wordy for the sake of reference, and not everything will be covered in class - the point is to have a useful place for you to come back and remind yourself of the things you've learnt in class, and more 😊

What will we learn

You can find a full list of the topics we'll cover at the start of these course notes.

The areas we will cover will include:

- HTML
- CSS
- UX - User Experience
- GitHub , repositories, and versions control
- Twitter Bootstrap
- JavaScript and jQuery

This will help you understand

- How websites are made and how the internet basically works,
- How to create and publish your own webpage,
- How to code in groups and contribute to the developer community,
- Skills & tools for coding and carrying on after the course.

We hope you will become curious about how the Internet works and will look at websites in a different way.

Our key focus will be on Front-End Web Technologies, mainly HTML, CSS, Twitter Bootstrap an important web framework, and JavaScript in the form of a framework, jQuery.

Now, let's introduce ourselves to each other and then let's check that you've installed all the right software and get you signed up for GitHub account!

Installations

There are many excellent resources on the web for learning a lot of the material we will be covering. We don't want to reinvent the wheel and will unashamedly point you towards better sources of information when they exist. Rather than teach you everything from scratch, we aim to guide and support your learning, focusing on core concepts and exercises to jump straight into coding.

As you grow in your development journey, you will find that Google and Stack Overflow are your best friends, and GitHub is almost like an online home.

One of the biggest challenges in a course like this is dealing with the different operating systems and hardware that you'll be working on. That does mean that there's a bit of setting up that you will need to do. Please be patient with this - installing software is sometimes fiddly and not always predictable. Most of the work we get you to do won't be like this, and if you have any problems, your instructor will be able to help you!

To Do	What/Why?	Links	Done?
1. Create a GitHub Account	GitHub is, informally, a code sharing and publishing service, and a developer network. Formally, it is a web-based repository hosting service for a version control system (***that tracks file version changes)*** called Git (***more later).***	GitHub ,	
2. Install Google Chrome	Chrome is a free web browser provided by Google. It comes with a good set of developer tools that we will be using over the course. *	Google Chrome	
3. Atom	A Text Editor is a Graphical user interface (GUI) that is built for writing and editing code that can be processed directly by the computer. We have chosen these as our recommended editors as they: (a) gives a good user experience on Mac, Linux, and Windows; (b) is easy to get started with; and (c) can be easily customised when you get more advanced.	Atom	
4. Install GitHub Desktop	A GUI for you to start using GitHub to collaborate on projects, without having to touch your command line.	GitHub Desktop Client	

Client	Note: Make sure you download the standard desktop client rather than the newer beta version.		

Additional Notes:

* Firefox also comes with an excellent set of developer tools (via its firebug extension). There doesn't seem to be much between Chrome and Firefox + firebug as far as the tools we'll be using go. The decision to recommend Chrome was fairly arbitrary, but means that everyone will be using the same software.

*** **Optional additional homework**: (***For those of you who would really like to be prepared***) Sign up to Codecademy. On the Codecademy web track: (Only the lessons are free, so we'll stick to doing those 😄)

1. Do the lessons from "1 Introduction to HTML".
2. Do the lessons from "2 HTML Structure: Using Lists".
3. Do the lessons from "3 HTML Structure: Tables, Divs, and Spans".

Introduction to coding

What is coding?

When we talk about "code", "source code", this is just a collection of computer instructions written using some language that is readable to humans. It's important to note that the purpose of coding is to manipulate data; which is the storage of information on your computer.

One of our key objectives will be to teach you how to create your own online website landing page. Therefore, we will be focusing on a two specific coding languages HTML – which is a mark-up languages, and CSS which is a style sheet language. Writing code in these languages only allows you to create static content without interactivity.

When code allows a computer to do more than just display static information (so where we need to do calculations or have some interactivity), we need to use a Programming language. For our purposes, we will only be at one programming language in this course – JavaScript. One way of thinking of this – programming languages allow you to carry out mathematical equations in a program, and not all types of code are meant for this purpose ("Programming" is a subset of "coding".).

The developer community is a vibrant, growing one. This article in Mashable probably explains it better:

"From the Internet's earliest days, programmers would congregate in chat rooms and on forums to ask questions, swap code, and brag about their latest software masterpieces. The old stereotype of the anti-social, code-obsessed geek couldn't be further from the truth. … Thanks to the very developers who used to haunt those fleeting chat channels and techie forums, we now have the bright, bold, user-friendly colours of the social web, where the current generation of coding wizards can connect with seasoned veterans to brainstorm the future of the Internet."

The direct corollary of this is the creation of shared web communities, such as GitHub and Stack Overflow, making it easier than ever to learn and contribute. 😊

Another way to classify web coding languages is by defining them as either "Front End" (or client side) web coding or "Back end "(or server side) web technologies. Our key focus will be on Front-End Web Technologies, mainly HTML, CSS, Twitter Bootstrap an important web framework, and JavaScript in the form of a framework, jQuery.

Front-End Web Development is about creating and styling the parts of a website, web service or application that users interact with (as opposed to the processes that happen behind the scenes; "back-end web development"). Those who are skilled in Front-End coding are able to plan and visualise how they want a website to be laid out. A good eye for design and an interest in the user journey helps too.
For this reason, Front-End technologies are also referred to as client-facing, or user-facing, technologies.

TASK: Find out! (*hint 1*, *hint 2*, *hint 3*)

An Overview of Different Web Technologies

 HTML *HyperText Markup Language*	 CSS *Cascading StyleSheets*	 JavaScript
a markup language **Describes** the **structure** of web pages. HTML documents are described by **HTML tags**, and each tag **describes** different element.	a stylesheet language **Describes** the **presentation** of web pages, including colors, layout, and fonts. It enables **responsive design**; for one to adapt the presentation to different types of devices.	a programming language Allows user interactivity, and enables web pages to be dynamic. (Usually without needing to reload the page) *It is a type of programming language, scripting.
Here is a basic HTML demo from W3Schools.	Here is a basic CSS demo from W3Schools.	Here is a basic JS demo from W3Schools.

How do they all come together?: Check out this basic example! (You're not expected to understand any of the syntax 😊)

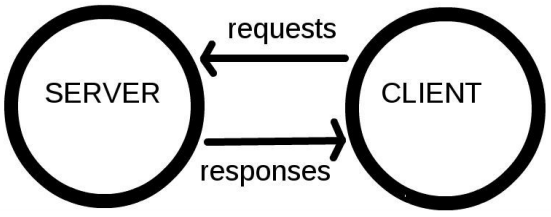
Getting going & HTML

Chapter 1 - What is the internet and how does it work?

Before we dive into coding, it is worth thinking about how the websites we are coding are shared online... How does someone find your website? How do we publish our websites for other people to find? **Don't worry! You won't be tested or anything, this is really just for your info/reference, and to give some knowledge and context to the websites we're building.** These notes complement this video. ***More video resources are in the "Extra Resources" section if you don't like reading*** 😛

Task: Finish reading this chapter. Then turn to your partner and explain how the internet works!

Computers connected to the Web are called **clients** and **servers**.



The computers that hold web pages, sites and applications are called **servers**. ***Web servers are large, specialised computers that are (almost) permanently connected to the internet.***

The computers that allow people to access the web by providing software (browsers) for them to do so are called **clients.** They are the typical user's Internet-connected device, for example - your laptop or mobile phone, connected to the internet through a browser.

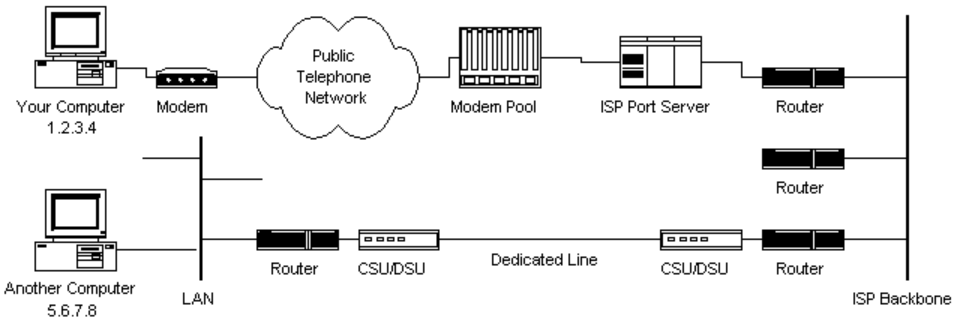
Servers respond to requests sent to them by clients. The **Internet** is the network layer that makes the Web possible, it lies on top of a very complex hardware infrastructure, which allows us to communicate by sending files to each other. We won't be covering the latter (the hardware) in this course.

You can think of the Internet like a super postal service.

Information from a client application is packaged nicely into easily transported, segmented, **data packets containing information about where the data should be sent**, and sent over a websocket (using the TCP/IP or UDP/IP protocol), we then use special tiny computers called **routers**, which acts as the default gateway to the rest of the Internet, and has only one job - to establish the route taken by the data packets.

To connect routers to each other, and to the internet, we make use of **radio waves** over **WiFi** (802.11), and **Ethernet** over existing telephone infrastructure by using **modems**, and connect our network to an Internet Service Provider (**ISP**) - a company that

manages special routers which all link together and can access special routers of other ISPs.



Chapter 2 – How the Web Works

Task: Finish reading this page. Then turn to your partner and explain how the web works! Once you've done that, explain the difference between the internet and the web.

How does the router know where to send your (client) requests? Many of our web interactions begin with a URL (***uniform resource locator***) being typed into your web browser address bar.

Let's look at an example: http://www.bbc.co.uk/news/.

This URL has several different parts to it:

- http : the ***protocol*** or ***how*** to fetch the information
- www.bbc.co.uk : the ***host*** or ***where*** to fetch it from
- /news : the ***path*** or precisely ***what*** information to fetch

When we type the URL into the address bar a request is sent over the internet and some information is returned to us.

The protocol describes how the information is transmitted. Other possibilities include https for secured communication, ftp for file transfer and git which you'll learn about later.

The host describes where the information should come from and the path tells that location precisely what information to send.

In general a URL can be more complicated than this. URLs can also contain ***query parameters***, ***fragments*** and ***port information***. We will leave these for now but will point them out when we meet them later. Instead we will focus on exactly what information is being sent and who is sending it.

Each computer on the internet has an address (an ***IP address***) so that requests can be sent to it and files returned – much like a telephone number. The backbone of the internet is a network of ***routers*** that are responsible for routing files from one IP address to another.

Static vs dynamic sites

There are two main possibilities server-side: either your site is static or dynamic.

- In a **static** site, all pages are pre-prepared and the web server just sends them to the browser.
- In a **dynamic** site, pages are prepared on-the-fly pulling information out of a database depending on what the user asked for.

Most of the sites you can think of will be dynamic sites (e.g. facebook.com, reddit.com, amazon.com.).

Server-side technologies

There are many options for building a dynamic server-side site. Common choices are:

 Ruby on Rails	 PHP	 Django	 node.js	
Web development framework written in Ruby (programming language)	Programming language made popular in the early 2000s.	Web development framework written in Python (programming language)	A platform for building fast and scalable server applications using JavaScript.	A blogging platform (now capable of much more) written in PHP.

Putting up a website

If you want to put up your own website at your own domain name you need two things:

1. A web server to serve your site
2. A domain name to point towards it

There are many options for web servers - you don't have to physically have your own one. Many companies that will offer **web hosting**, often providing you with space on a shared server. Later in the course we will use the free hosting offered by GitHub through GitHub Pages. To buy a domain name you need to use a domain registrar. This is not necessary for our course.

Chapter 3 – What makes up a website?

A **website** is a collection of linked webpages (and their associated resources); a set of related files that is compatible to your browser.

A **web-page** is just a file that your web browser is able to read and display, written in a markup language; HTML.

The website's main webpage (or point of entry) is referred to as the homepage, or a landing page. This is predefined, and its document name must be set as index.html.
To display the page on the client-side device, a browser starts out by reading the HTML. This is why additional resources, such as CSS & JS, are written in dedicated documents and are "plugged-in" or linked into HTML documents, as we will see later.
A typical webpage depends on several technologies (such as CSS, JavaScript, Flash, AJAX, JSON) to control what the end-user sees, but most fundamentally, developers write web pages in HTML, without which there can be no webpages. Associated (additional) resources can be embedded into your HTML file. These include, and are not limited to:

- page styles; CSS (Cascading Style Sheets)
- scripts – for interactivity; JavaScript
- media – videos, music, etc.

You can think of a website as a very sophisticated letter, which you can interact with. And as you'll find out, the Internet is like a very sophisticated postal service. When you visit a URL, a complete packaged folder of documents is sent to you. Can anyone think of any potential ramifications of this?

Demo:

Here is a demonstration of how additional resources can affect the HTML document. Recap from course prep the role of each language & do a Q&A of languages – ask what they think each language does to the page and clarify, etc.

Because the web document files are sent to your browser, it is easy for you to look at them. **There are no secrets in HTML, CSS or js.** If there's a part of a webpage that you like, it's easy to find out how it is coded and use the technique yourself.

Every browser provides a way to look at the source of the page you're currently viewing.

In Chrome you do View > Developer > View Source. This will show you the raw HTML but isn't always the easiest thing to look at.

Several browsers also provide developer tools, which allow you to ***interactively*** view the page source. In Chrome you can access these by doing View > Developer > Developer Tools. If you use Firefox, you can get similar functionality with the Firebug plugin.

These tools are the best way to investigate a web page. Over the course you will be using them a lot on your own pages, especially when things aren't working exactly as you expect.

There are a few features of the Chrome developer tools that it is worth pointing out now.

Task: Viewing the HTML, CSS & JavaScript of a live web page

1. Open any website in Google Chrome, or the Code First: Girls website
2. View the page source by doing one of the following:
    a. View > Developer > View Source
    b. Tools > View Source
3. Open the developer tools by doing one of the following:
    a. View > Developer > Developer Tools
    b. Tools > Developer Tools
4. Use the square with a pointer in the bottom left to hover over bits of the page and find the related HTML.
5. Hover over the HTML code in the developer tools box and watch as different parts of the page are highlighted.
6. Try changing some of the CSS on the right hand side. To undo any changes just refresh the page.
7. Have a look on the Resources tab. See if you can find the CSS, JavaScript and image files used on this page.
8. Visit a few of your favourite websites and repeat this process.

Part 1 - HTML

Chapter 4 - Creating a basic HTML page

One of the nice things about HTML is you don't need any fancy software to test it out on your laptop: all you need is a text editor and a web browser.

Task: Create a new project folder and your first HTML file

1. Create a folder called coding_course to hold all your work for the course.
2. Inside your coding_course folder create another folder called first_site.
3. Open your text editor.
4. Write "Hello"
5. Save the file as index.html in the first_site folder
6. Open index.html in Chrome. Depending on your version of Chrome
   - File > Open
   - Cmd-o (Mac) or Ctrl-o (Windows)

Why index.html?

Navigating a website was a lot more like moving around the folder system on your laptop. You would go to a base site and there would just be a list of the files and folders available: an index. Because of this index.html is still the default file that a server will serve when you navigate to a folder in the web browser.

Task: Adding code to your HTML file and viewing in Chrome

1. Go back to 'index.html' in your text editor.
2. Change the text to

```
<h1>Hello</h1>
```

1. Save 'index.html' - Cmd-S [Mac] or Ctrl-S [Windows]. ***Hint: if there is a circle next to your file tab in your text editor, it's not saved!***
2. Go back to 'index.html' in Chrome and refresh the page (Cmd-R [Mac] or Ctrl-R [Windows])

This is your first line of HTML. It has an open tag and a close tag. They tell your browser to display the text in-between as a heading.

Chapter 5 – HTML Basics

Elements, Tags & Attributes

HTML uses a ***predefined*** set of **elements** for different **types of content**; they define the **semantic value** (or meaning) of their content. Elements include two matching tags and everything in between. They contain one or more "tags" which either contain or express content.

For example, the "<p>" element indicates a paragraph; the "<img>" element indicates an image.

HTML attaches special meaning to anything that starts with the less-than sign ("<") and ends with the greater-than sign (">"). Such markup is called a **tag**.

Tags are enclosed by ***angle brackets***, and the closing tag begins with a forward slash.

Make sure to **close** the tag, as some tags are closed by default, whereas others might produce unexpected errors if you forget the end tag. An example of a tag that closes by default is the image tag.

```
<img src="smileyface.jpg" alt="Smileyface">
```

The start tag may contain additional information, also known as an **attribute**. Attributes usually consist of ***2 parts***, its **name** and corresponding **value**. In the example below, the attribute name is “class”, and the class of the div is “main”.

```
<div class="main">
```

Good Code

To write **good** code, you must properly nest start and closing tags, that is, write close tags in the opposite order from the start tags. All text editors will have an Auto Indent feature which will help you format your code. In Atom you can navigate to: File > Lines > Auto indent.

Valid code:

```
<div>
<p>
<em>I <strong>really</strong> mean that</em>.
</p>
</div>
```

Invalid code:

```
<div> <p> <em>I <strong>really</em> mean</div> that</strong>.</p>
```

HTML DOM

The **Document Object Model** specifies the hierarchical layout of the HTML document. It is an agreed interface that is platform-independent, language-independent, and interacts with any HTML or XML (markup) document.
It is loaded in the browser, and represents the document as a node tree, with each node representing a part of the document. In other words, it tells the browser where to look for a specific thing in the document.
This is very useful for software development, as you will find out later in the course when we learn JavaScript, but **it's best if you start coding cleanly from the start.**
Every HTML5 document requires this layout:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    ...
  </body>
</html>
```

- The !DOCTYPE tells you what version of html you're using (html5, html4 …):
  - With html5 (latest) it's simple – you just write html.
- Everything is wrapped in an <html> … </html> tag
- Things within the <head> .. </head> are used to provide information about the page – "The Brain"
- Only things within the <body> … </body> tags are displayed on the page
- … for example the text within <title> … </title> will be displayed in the browser bar

HTML Syntax

Demo here - Instructor to talk through this!

Now have a go yourself - you will use these ideas to create a richer web page!

Task: Add HTML syntax to a file

1. Go to the GitHub repository (or folder) for this session: https://GitHub.com/code61/learning_html
2. Download the code into your coding_course folder (by clicking 'Download ZIP' in the bottom right).
3. Open the whole folder in your text editor
4. Open the file example.html in Chrome and look around with the developer tools
5. Open the file notes.html in your text editor.
6. Change notes.html into valid html so that it looks like notes_solution.jpg

Part 2 – CSS

Chapter 6 – What is CSS?

Recap

- A Website is just a collection of files: HTML, CSS & JavaScript.
- We can edit these files locally.
- We can create them locally (using a text editor) and view then in a browser.

Task: Creating a CSS file

1. Open your text editor and create a new file (go to file > new file or use the shortcut Ctrl + N [windows] or Cmd + N [Mac])
2. Write <h1>.
3. Save the file as index.css in the first_site folder.

CSS (Cascading Style Sheets) was created by the World Wide Web Consortium (W3C) to solve to problem of formatting and styling a document. This allows HTML to do its job – defining the content of a document.

A powerful example of this is CSS Zen Garden – by clicking on the links you completely change how the site looks, but the HTML remains unchanged.

Chapter 7– Linking your CSS and HTML code

There are a few ways to use CSS to add style to your HTML:

1. Add it directly to your HTML file.
2. Create a CSS file and link it to your HTML.

Usually it is better practice to create a separate file. The reason being that if you add it to your HTML file you then it is more complicated to update and if can make it more difficult to spot issues.

If you do create a separate CSS file, the way you connect that to your HTML file is by referencing that CSS file in the head of your HTML file.

Linking to a separate CSS file in the <head>

By separating these CSS rules into their own file you:

(a) reduce repetition in your code and (b) reduce the amount of information that has to be sent to the browser for each page – if the CSS file applies to the whole site, it only needs to be sent to the visitor once.

To link to an external CSS file you add the instruction with reference to your CSS file in the "head" section of your HTML file as below:

```
<!DOCTYPE html>
<html>
 <head>
        <title>My page</title>
        <link rel='stylesheet' type='text/css' href='exercise1.css'>
 </head>

 <!-- body goes here -->

</html>
```

More about links below.

More about the HTML <link> Tag

Linking to other files (stylesheets, JavaScript files, images) can be done in several ways, just like linking to another page. Say you have the following directory structure:

```
first_site
---- index.html
---- images
|   ---- background.jpg
---- stylesheets
|   ---- main.css
```

and you're going to deploy your site to "http://www.my_first_site.com". Suppose you want to link to main.css from index.html and to background.jpg from main.css. There are three different styles of links you can use:

1. Absolute links

Absolute external links include the complete url to the resource you're linking to. **Absolute links start with either http:// or https://**.

```
<!-- in index.html -->
<link rel="stylesheet" type="text/css" href="http://www.my_site.com/stylesheets/main.css">
/* in main.css */
```

```
body {
  background-image: url("http://www.my_site.com/images/background.jpg");
}
```

Absolute external links can be used to link to resources held on different sites, but wouldn't usually be used for links within your own site.

They're a bit fragile - if you change your domain name all the links will break. They also won't work when you're developing locally.

2. Root-relative links

Root-relative links contain the path to the resource ***relative to the site's root***. The site's root is (roughly) the folder that contains the site - in this case, first_site. **Root-relative links begin with a /**:

```
<!-- in index.html -->
<link rel="stylesheet" type="text/css" href="/stylesheets/main.css">
/* in main.css */
body {
  background-image: url("/images/background.jpg");
}
```

Root-relative links are a bit more flexible than absolute external links: e.g. if you change your domain name everything will still be fine. They're sometimes useful for your own static sites, but probably won't work when developing locally (because the root will be taken to be the root of your file system and not the folder containing the site!).

3. Document-relative links

Document-relative links contain the path to the resource relative to the file where the link is written. **Document-relative links don't begin with /**:

```
<!-- in index.html -->
<link rel="stylesheet" type="text/css" href="../stylesheets/main.css">
/* in main.css */
body {
  background-image: url("../images/background.jpg");
}
```

To link to the stylesheet from index.html we use stylesheets/main.css which says "look in the same folder I'm in (first_site) and find a folder called stylesheets and a file in it called main.css".

To link to the image from the stylesheet is slightly more complicated: we use ../images/background.jpg. This means "go to the folder above the one I'm in (I'm in stylesheets, so that's first_site) and find a folder called images and a file in it called background.jpg".

Important to know

In document-relative links (and in many other places e.g. command line navigation):

- . means in the folder **that I'm in**
- .. means in the folder **above the one that I'm in**

Relative links are the most flexible - they will work on your local file system. The only thing you have to be careful about is moving your files into different folders, which can cause links to break.

You should be using relative local links in these exercises.

Task: Link your CSS file to your HTML file

1. Open exercise1.html in Atom and in Chrome, and make a separate CSS file, called exercise1.css.
2. Link your exercise1.css file to your exercise1.html file within the <head> tag.

Chapter 8 - Writing CSS & basic definitions

Your instructor will demonstrate by forking this pen and typing in the CSS (below) during class. This demo follows the exercise students have downloaded from GitHub . Students will finish it according to the picture they have downloaded as well.

Start writing your CSS in exercise1.css (option 3) it should look something like this:

```
h1 {
color: red;
}
```

This is called a **rule set** - a single section of CSS including a ***selector***, ***curly braces***, and a ***declaration*** consisting of ***properties*** and ***values***.

The h1 bit specifies the tag that will be styled, this is a CSS **selector**.

The bit inside the { ... } specifies the styles that will be applied, this is the **declaration**. Here we change the colour of the h1 text red. If there is more than one declaration

within the curly bracket (e.g. make the size of the h1 text 30px), they are all collectively called a **declaration block**.

The **property** of HTML being changed in the example above is "color", and the **value** it is being changed to is "red".

If you save the changes to the CSS file, then open (or refresh) the page in your browser you should see the changes. By opening the developer tools, and hovering over the h1 you should be able to see your CSS rule at the side.

A few more properties

In the example above we changed the color of the h1 element. Here are a few examples of other simple properties you can try out while you're getting the hang of CSS:

```
p {
font-family: 'Arial';
}

h2 {
font-size: 20px;
}
h3 {
  background-color: green;
  font-size: 2px;
}
```

The last example, of the h3 rule set, is a demonstration of a **declaration block** - when you specify more than one declaration for a selector.

Task : Playing with the styling on the HTML file

1. Open exercise1.css in Atom and in Chrome.
2. Make the h1 turn red.
3. Continue with the exercise until exercise1.html looks like exercise1_solution.png.

Chapter 9 - Selectors and Attributes

DEMO, type in CSS. So far you've used HTML tags to specify CSS rules. For example,

```
h2 {
 font-size: 40px;
```

```
  color: pink;
}
```

will turn all your <h2> massive and pink. This is called an **element type** selector; when a selector targets an HTML element directly by the **tag name**.

It is often useful to be able to make CSS rules more specific, so you can apply different styles to different parts of the page. One common way to do this is to target specific HTML attributes - particularly, the ID & class selectors. You can also combine selectors.

Chapter 10 - Using the ID and class selectors

HTML tags can have ***attributes*** which provide additional information about the element. (This is **completely different** from a CSS attribute!) You've already seen some examples of this:

```
<a href="http://www.facebook.com">
<img src="cat_pic.png">
```

Recap: Both href and src are examples of html attributes. They are written in pairs with their values: attribute="value".

There are some attributes that can be added to any tag. Two examples of these are id and class:

```
<h2 id="products_title">Our scrumptious puddings</h2>
<ul id="products_list">
  <li class="product_item">Black forrest gateau</li>
  <li class="product_item">Rasberry lemon swirl cheesecake</li>
  <li class="product_item">Sticky toffee pudding</li>
  <li class="product_item">Death-by-chocolate cake</li>
</ul>
```

IMPORTANT!

Both ID and class are used to add some information to the HTML tags. The key difference is that ID should specify a ***unique element on the page***, whereas multiple elements can **share** the same class. This is useful when you have loads of code to sift through.

CSS lets you target the id and class attributes in special ways:

```
/* make the h2 with id="products_title" purple */
```

```
h2#products_title { color: purple; }

/* remove the bullets from all li with class="product_item" */

li.product_item { list-style-type: none; }
```

The id is targeted by adding #id_value to the tag and the class is targeted by adding a .class_value to the tag.

It is also possible to target any items with a given class or id by leaving out the HTML tag:

```
/* make any element with id="products_title" purple */

#products_title { color: purple; }

/* make any element with class="product_item blue" */

.product_item { color: blue; }
```

Chapter 11 - The Universal Selector

The universal selector matches any element within the context in which it's placed in a selector. The * character is the universal selector:

```
* {

font-family: 'Helvetica', sans-serif;

}
```

This should only be used for declaring things on the entire document, using it in a combination of selectors is redundant.

To find out more about the different kind of selectors, read this article.

Chapter 12 - HTML <divs> and <spans>

There are two important HTML tags, that we didn't used before: and . Both are really useful when it comes to using HTML attributes to target CSS classes.

 stands for ***division*** and is used to break the page up into different parts. It is a 'block-level' element, which means that it will start a new line before and after it.

 can be used to apply classes and ids to certain bits of text. It is an 'inline' element, which won't start a new paragraph before or after.

```
<div id='info_section'>
```

```
<p>This is a paragraph in the info section. We can use a span to target <span class='important'>certain bits of important text</span>.<p>

</div>
```

Task: Separate the CSS in the HTML file into linked CSS file

1. Open the file html2/exercise2.html in Atom Text and Chrome (it's inside the html2 folder you downloaded earlier.
2. Separate the CSS in the <head> tags into a linked CSS file.

Continue until your page looks like html2/exercise2_solution.png

Part 3 -Introduction to UX

Chapter 13- User Experience

What we will learn

- What is UX (User Experience)?
- What is responsible for UX on a team?
- Why UX matters?
- How can analytics impact UX?

Coding is not the only way in which we can move into the wonderful world of technology. There are even people-centric roles, like user experience researchers or designers. After all, we have to understand who we're building for.

What is User Experience?

User experience is the overall feelings your product inflicts on those who use it. It is every look of disgust when something unexpected happens. It's the joy in personalising cards. UX is using dark colours for using websites at night, to reduce glare.

Have you achieved every goal you had in mind on every website you've ever used?
Have you returned to every website you have ever used? Why, why not?

Task: Fun Exercise!

1. Pair up with someone with a different phone to you - Android/iPhone/Windows/Blackberry etc.
2. Swap phones
3. Find a cat image online, save it to the phone, then find the downloaded file
4. **If you are not comfortable with swapping phones,** the owner of the phone will 'drive', while the other person will tell them what they would do

Building websites should take into consideration the problems people are having and why. The journey of your app begins even before someone has touched a phone.

What UX is not

UX is not the same as UI. UI stands for User Interface and usually refers to the aesthetics of a design. UX is more the psychology of how usable a product it.

While UI does have an impact on how people feel, UI is not user-centric.

For example, this website has a very pretty UI. It is slick, modern and looks very clean. However, since you have to guess where the next piece of text will be and constantly look around the page, the page does not have a great UX.

UX is also not new. The foundations of UX come from another discipline called Human Computer Interaction (HCI), which studies how people build relationships with technology and how we interact with computers.

User Experience is also not easy. It can be subjective, with challenges being solved in multiple different ways. While a solution may work for one demographic, it does not mean it will work for all.

Who is responsible for UX?

Everyone on a team is responsible for the overall user experience. Some may lead on certain aspects like developing the UX or researching with people. But everyone who touches the product in some way can affect its UX.

Marketing: need to understand the problems people have, to sell their product which solves that problem
Visual Designers: need to communicate the emotions of the brand using aesthetics
UX-ers: need to champion changes, research with people, and oversee the project
Business analysts: need to balance requirements and how they benefit the user and the business
Developers: need to build the product and the behaviour of the product
Quality analysts: need to know how to test the UX and spot issues in behaviour

Why does UX matter?

The user experience of a product can make or break a company. If it is difficult for someone to achieve a goal, then why would they return to try again? Especially when someone else makes it easier for them.

Let's look at the evolution of social media.

Friends Reunited was the bee's knees when it first came out. It found a problem – people losing contact with each other, and provided a solution. But they did not think about what people would do after they had connected. There was no information posted by people, meaning the content lacked. Other tech like digital cameras etc. made it harder for people to post their information in the first place.

Quotes from user testing

"Suddenly it's become like a test! Jesus. Oh my god. I'd be having a cup of tea now if I was at home."

"I don't know how I would react to Zara if I started seeing Indian models. It kind of takes away the international feeling of the brand."

"There's nothing more frustrating when you do a password and it tells you it's wrong, but it doesn't tell you what aspect is wrong"

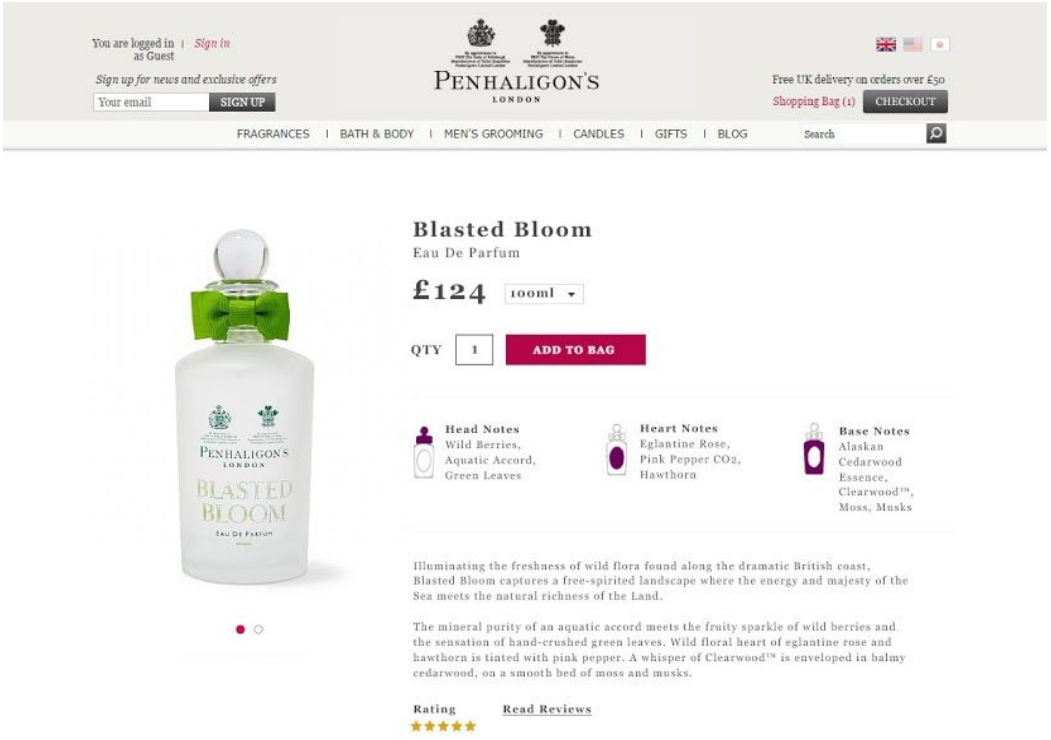
"Like I'm stupid because sometimes I can't get it right"

UX and Analytics

Analytics are research over a larger audience of people. The area of UX which looks closer at analytics is called Conversion Rate Optimisation. This is the discipline of using UX to make minor tweaks to understand if the changes have an impact on your website.

A/B testing can test changes from the colours of buttons to completely different page layouts. Tools like Google Analytics, HotJar and Optimizely can help target specific demographics with different versions of the website/page.

Control version - original project page design



Variation 1 - Changed layout of Add to Bag

This test tested the layout of the Add to Bag call to action. The goal was to increase engagement with the button. Final results showed an increase in revenue of 15.9%.

Task: Sketch, swap & explain

1. In your group, grab a pen and start sketching your website idea
2. Discuss what content needs to be on the pages and where it goes
3. Include things like images, text, buttons, navigation, footers etc.
4. If you have time, swap sketches with another team.

a. One person on your team will ask another person to explain the sketch to see if it is usable and it makes sense

b. Someone else on your team will look at another team's sketch, explaining what they think the website is

5. **Remember to add the image into your website folder, to upload it to GitHub !**

Part 4 - Course competition

Chapter 14 - Competition criteria

You've now started learning how to make websites, and now it's time to put those skills to the test! Your task is to make a website of your choosing, to be presented in the next session in small groups.

The criteria we'll be looking at are as follows. You can see the **Must have** criteria and the **Nice to have**. It would be great to have as many Nice to have features as possible, but don't worry we know you won't have time to include it all! Anything you don't get time to include now, can always be added later on.

Must have:

- A live website published on GitHub pages
- A minimum of **two** HTML files for:
    - 1 x index.html landing page linked to a separate CSS file
    - 1 x 'about' page
- A minimum of **one** CSS file
- Good formatting
    - Code split into the appropriate files (separate HTML files & CSS files)
    - Files indented properly
- Good organisation
- Version control using git
- Sensible git commit messages

Nice to have:

- A visually appealing design - good use of CSS and HTML elements, Twitter Bootstrap, Jquery & JavaScript (don't worry you'll learn about these last three topics next week!)
- A contact form (for example name and email)
- Social buttons
- As many different HTML elements you can manage
- Interactive elements (like forms) on your website don't need to be functional, but should be present if they need to be for the visual aspect of the design.
- A responsive site (again you'll learn about this next week!)

So that's it! Other than that, you can make any kind of website you want.

You can see a few examples of what previous students on your course created on our website here: http://www.codefirstgirls.org.uk/course-competition.html

If you're short on ideas, here are some to get you going...

- A personal website
- "How to" website on an area of your expertise
- A survey or poll website.

The websites will be judged by your instructors at the end of the last session. The winning website will receive a prize!

As always, if you have any questions, don't be afraid to ask, via email, or during the session, and most importantly, have fun!

Task: Forming teams and publishing your website via GitHub Pages

We'll now need you to split into your course competition groups (teams of 2 maximum 3). You will need to work with your group and instructors to complete the following tasks in the coming days as you work through the chapters:

1. Decide who on your team will be your team lead.
2. Your team lead to create your group project repository and corresponding folder via GitHub desktop client (see chapter 16).
3. Create a HTML file (index.html) and CSS file (index.css) in your project folder, making sure that you link your CSS file to your HTML file (see chapter 4, 5 & 6).
4. Additionally include at least one other HTML file (eg. about.html).
5. Team lead to add all team members as collaborators to the repository.
6. Other team members should now clone this repository to their local machine via GitHub desktop client (see chapter 16).
7. Work on these files collaboratively with your team, making sure that you save changes and push those changes as needed (see chapter 16).
8. Create a GitHub pages branch in your repository (see chapter 17).
9. Publish your website via GitHub pages (see chapter 17).

Part 5 - GitHub , Version Control, Git

Chapter 15 - Version Control & Using GitHub

So, what is Version Control and why is it important?

Have you ever worked in a group on a Powerpoint presentation or Word document? What is that process like? (Let's not think about Google Slides or Google Docs for the moment.)

Let's start with two people - you and your partner are presenting on the controversial topic of the future of the UK's relationship with the European Union. You've got a structure for the presentation and started delegating the work; You'll work on the first section discussing the state of the relationship, and your partner will start working on the possible future scenarios.

You both go away and make the slides and handout notes separately and meet again in a week. But when you meet, there's a blocker that's completely irrelevant to your material:

You've decided to make your presentation with a black-background theme, in Calibri, and your partner has chosen a completely different colour scheme, used different fonts, and you've both made about 10-15 slides, organised in a different style from one another, and different types of handouts.

Yes, you could have waited for one person to start making the slides and emailed it over, or decided a "style" from the start, but that would have added an extra meeting and maybe a couple of days of waiting back and forth. Either way, you need to spend at least a couple of hours coordinating, or fiddling with the styling of the slides and handouts.

What if there were a way to automate the merging of your material and choose what to keep and what not to keep?

Let's consider another scenario; Your CV. Over the years, you would have made a multitude of edits to your CV. How do you save all the different versions? What if you could see exactly what changes you've made and where?

Version Control (aka Revision Control aka Source Control) lets you track your files over time. Why do you care? So when you mess up you can easily get back to a previous working version.

Version control is also the key to collaborative software development - meaning you can work in teams on the same project easily and manage conflicts in code on the

same files, work on different versions all at the same time, and decide what you want to keep and what you want to bin at the end.

So Why Do We Need A Version Control System (VCS)?

To be concise, this is why we need it:

- **Backup and Restore.** Files are saved as they are edited, and you can jump to any moment in time. Need that file as it was on Feb 23, 2007? No problem.
- **Synchronization.** Lets people share files and stay up-to-date with the latest version.
- **Short-term undo.** Monkeying with a file and messed it up? (That's just like you, isn't it?). Throw away your changes and go back to the "last known good" version in the database.
- **Long-term undo.** Sometimes we mess up bad. Suppose you made a change a year ago, and it had a bug. Jump back to the old version, and see what change was made that day.
- **Track Changes**. As files are updated, you can leave messages explaining why the change happened (stored in the VCS, not the file). This makes it easy to see how a file is evolving over time, and why.
- **Track Ownership.** A VCS tags every change with the name of the person who made it. Helpful for blamestorming giving credit.
- **Sandboxing**, or insurance against yourself. Making a big change? You can make temporary changes in an isolated area, test and work out the kinks before "checking in" your changes.
- **Branching and merging**. A larger sandbox. You can **branch** a copy of your code into a separate area and modify it in isolation (tracking changes separately). Later, you can **merge** your work back into the common area.

Shared folders are quick and simple, but can't beat these features.

Git is a very powerful and popular version control system, and the only place you can run **all** Git commands is from the command line. **Please note git is not the same as GitHub ! ** You'll see the difference in practice very soon.

The below are some of the main elements of version control, check through them and talk them through with your partner or instructor if you're unclear:

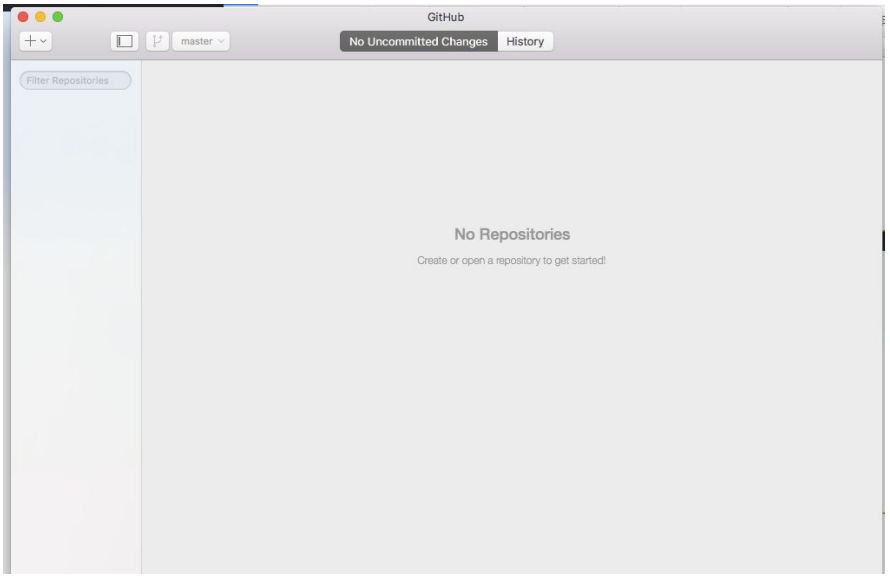
- Repositories
- A **repository** is the basic unit of GitHub , most commonly a single project. Repositories can contain folders and files, including images – anything your project needs. They **should** include a README and a license
- Branches
- **Branching** is the way to work on different parts of a repository at one time.

- When you create a repository, by default it has one branch with the name master.
- Commits
- On GitHub , saved changes are called **commits**. Commits are pretty glorious, because a bunch of them together read like the history of your project.
- Each commit has an associated **commit message**, which is a description explaining why a particular change was made. Thanks to these messages, you and others can read through commits and understand what you've done and why.
- Issues
- An **Issue** is a note on a repository about something that needs attention. It could be a bug, a feature request, a question or lots of other things.
- Pull Requests
- Pull Requests are the heart of collaboration on GitHub . When you make a **pull request**, you're proposing your changes and requesting that someone pull in your contribution - aka merge them into their branch.

Chapter 16 - Create your group repository

Task: Creating a GitHub repository

1. Open up GitHub desktop client on your computer.
2. You should see something like this.

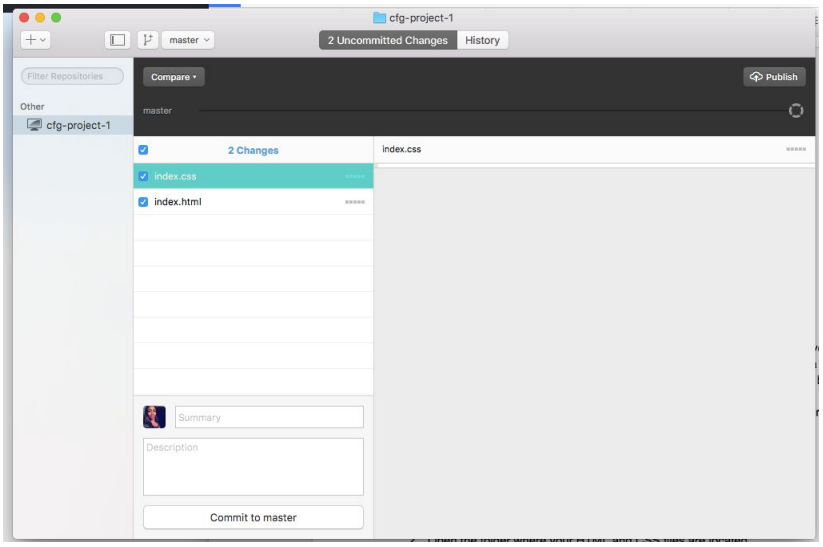


3. Click on the cross at the top left corner of the screen.
4. On the "create" tab type a name for your repository (note you name has to have a hyphen instead of spaces eg. cfg-project-1). Before you press "create repository" make a note of the local path as this shows where your corresponding folder for your repository has been created.

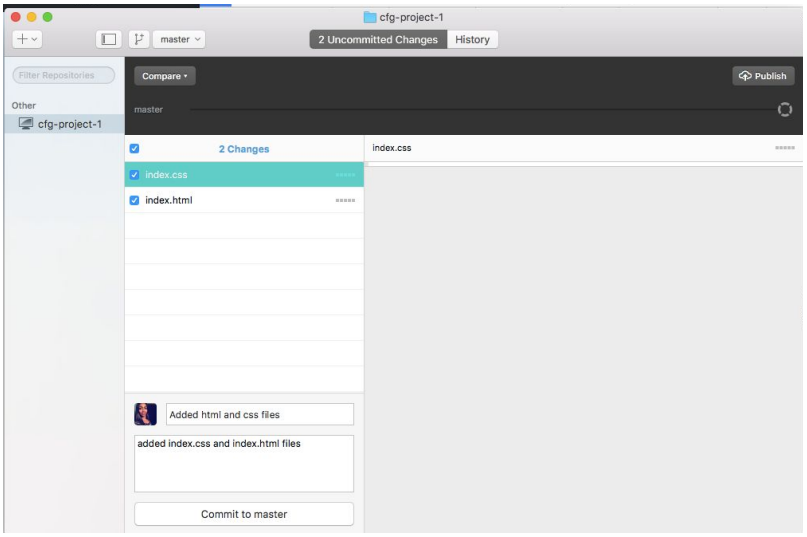
5. What the GitHub desktop client has done for you now is created a repository for you project and the corresponding folder.

Task: Adding your CSS and HTML files to your repository folder

1. Open your repository folder (located in the "local path" location).
2. Open the folder where your HTML and CSS files are located.
3. Drag the HTML and CSS files into your repository folder.
4. If you now look at your GitHub desktop client you should see your HTML and CSS files listed in your repository.



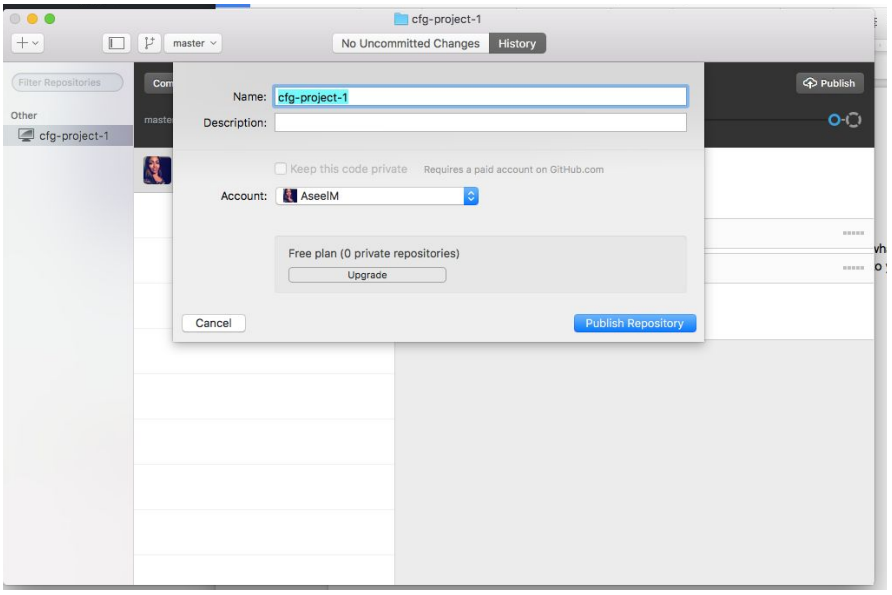
5. You now have to add a comment to describe what you have done. To do this add a short summary of what you have done in the section titled "summary" in your GitHub desktop client (eg. added html and CSS files) and as an optional extra you can also add a more full description if you choose. Then click "commit to master".



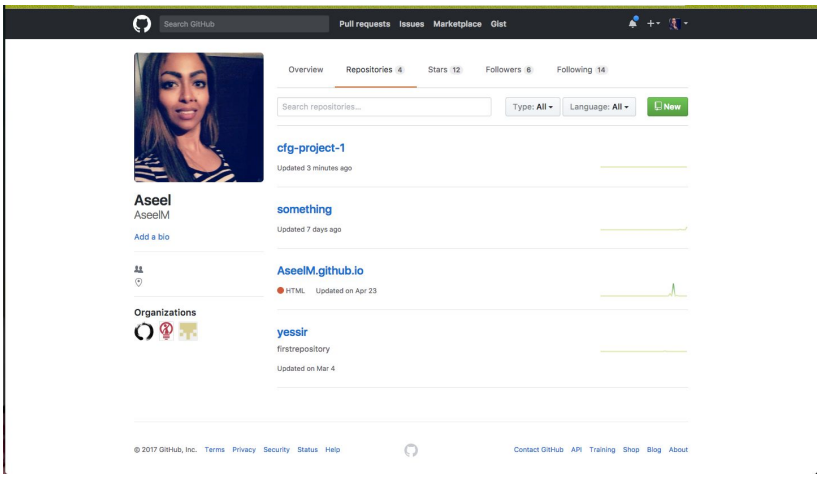
Task: Pushing your repository to your main GitHub account on the web

Note: So far what we have done is create the repository on your local machine, what we are now going to do now is synchronise what you have done on your local machine to your main account.

1. Click "publish" on the top right hand side of your GitHub desktop client. You should see something like the below image.



2. Click "publish repository" and the desktop client should show a message saying it is pushing to GitHub .
3. Go to the your GitHub account on the web. When you click on the "repositories" tab, you should now see the repository that you have created.



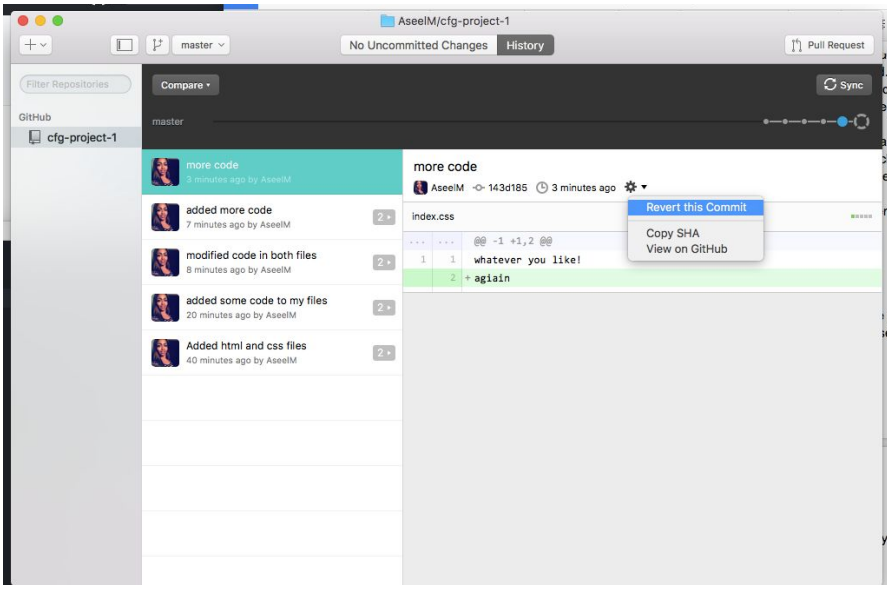
Task: Pushing changes in your code to your repository

1. Once you have finished making changes to your HTML/ CSS files in your repository folder in Atom make sure you press save.
2. Open your GitHub desktop client. You should click on the "uncommitted changes" button on the top of your screen and you should be able to see the changes that you have made to your HTML and CSS files. Any changes where you have added new code will be highlighted in green and any deletes will be highlighted in red.
3. To commit these changes to your main GitHub account you now need to add a summary of what changes you have made (you have to do this every time you commit a change).
4. Once you have added your summary click on "commit to master" this means you have now committed these changes to your GitHub repository on your local machine.
5. To now push these changes to your main account online click "sync" at the top right hand side of your GitHub desktop client.
6. The changes should now have disappeared from your GitHub desktop client "uncommitted changes" tab but should be visible on the history tab.

Task: Undo your last commit

Note: There are two ways to undo your last commit. The first way is if you realize you have made a mistake directly after you have pressed "commit to master", you should see an "undo" button visible, click on this to undo your last commit. If you have already synced your commits or don't see the "undo" button, do the following:

1. Click on your history tab in your desktop client and select the commit that you want to undo.
2. Click on the "settings wheel" and click on "revert this commit" button.



3. This has now automatically created a new commit with your changes reverted.

Task: Creating a branch

Note: **Branching** is the way to work on different parts of a repository at one time. When you create a repository, by default it has one branch with the name master. The instructions below are to help you create new branches in addition to the master branch.

1. On your GitHub desktop client click on "branch" icon .
2. Name your branch and click on "create branch" button.
3. Now make the changes that you want to your HTML and CSS files. These changes should now show as uncommitted changes on your branch in your desktop client.
4. To commit these changes to your branch add a summary to your commit as you have before and then click on "commit to branch".
5. To push your commit to your main online GitHub account press "publish".

Task: Adding a collaborator (i.e. your teammate) to your repository

1. Go to your online GitHub account and select the repository that you want to add a collaborator to.
2. Now click on the "settings" tab for the repository and then click on the "collaborators" tab.

3. Search for the username of the person you want to add to your repository, once you have found the person click on “add as collaborator”.
4. Your collaborator now needs to accept the invite that has now been sent to their email. Once they have accepted they should be able to see this repository in both their online GitHub account.

Task: Cloning a repository

Note: To be able to clone a repository it either has to be a public one or you have been added as a collaborator to.

1. In your GitHub desktop client go to the “+” dropdown at the top left hand side of the screen, then click on the “clone” button.
2. Select the repository you want to clone and then click on the “clone” button.
3. Step to choose the folder you want to save your cloned repository folder to (eg. your “documents” folder) and click OK.
4. You should now be able to see this repository on your GitHub desktop client.
5. Once you have made changes to your CSS or HTML files remember to commit these changes and sync them as you have done before. Your other collaborators should now be able to see these changes when they click on the history tab of the repository.

Note: To reduce conflicts before you start working on collaborative HTML and CSS files in the shared repository always make sure you press “sync” on the branch that you want to edit on your desktop client before you start to make any changes. When you sync from your desktop client you are only syncing that specific branch, so make sure your sync each individual branch required.

Task: Merging a branch back into your master

1. Click on the master branch and then click the “compare” button then select the branch that you want to compare and merge with the master.
2. Click on the “update from <branch name>” button, and press “sync”.

Chapter 17- Publishing on GitHub Pages

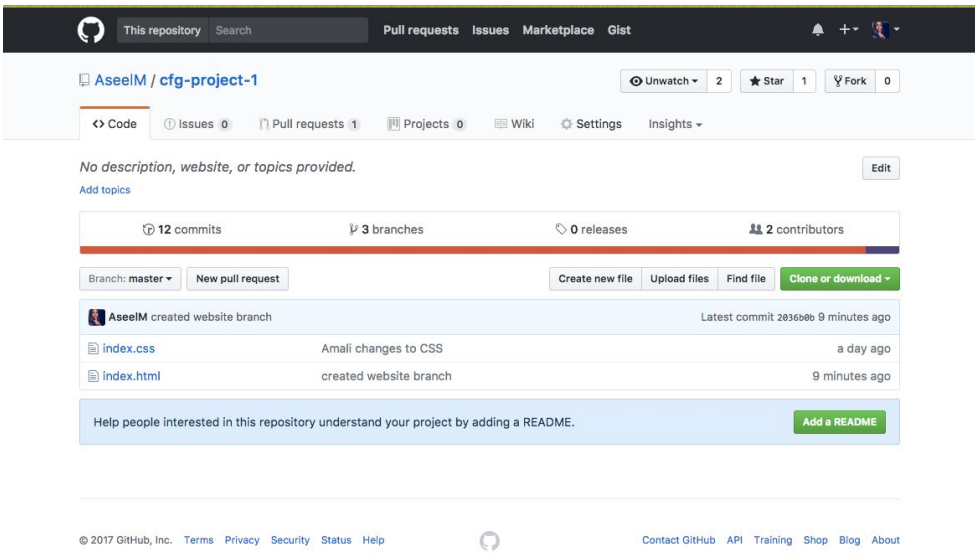
How GitHub Pages Works

GitHub Pages is a free hosting tool offered by GitHub . They only serve basic static web changes where no logic such as logging in and storing data is required.

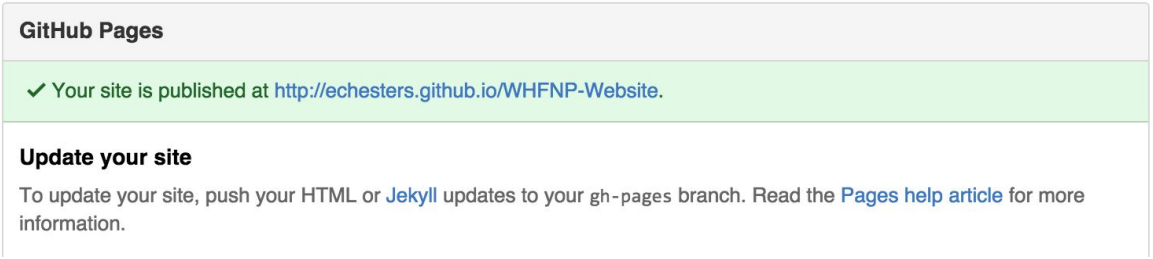
In order to publish your website through GitHub pages you will need to create a branch in your repository called "gh-pages" GitHub looks for this branch and builds a webpage from that.
Any code that you want displayed on your website needs to be on this branch. If it's on master, GitHub Pages will ignore it.
Once you publish your site on GitHub pages, your website will show up at the url [your-GitHub -username].GitHub .io.

Task: Publishing your website on GitHub pages

1. Create a new branch in your repository called "gh-pages" (see chapter 16).
2. Go to the repository on your online account. You should see a tab called "settings" at the top of the screen.



3. Click on the "settings" tabs and scroll down the page to the "GitHub Pages" section. You should now be able to see the web address for your website.



Click on that link and you should now have your website up and running!

Any changes you want to take affect on your website **must** be on this branch. Now you can make changes, push them live and build your own free website.

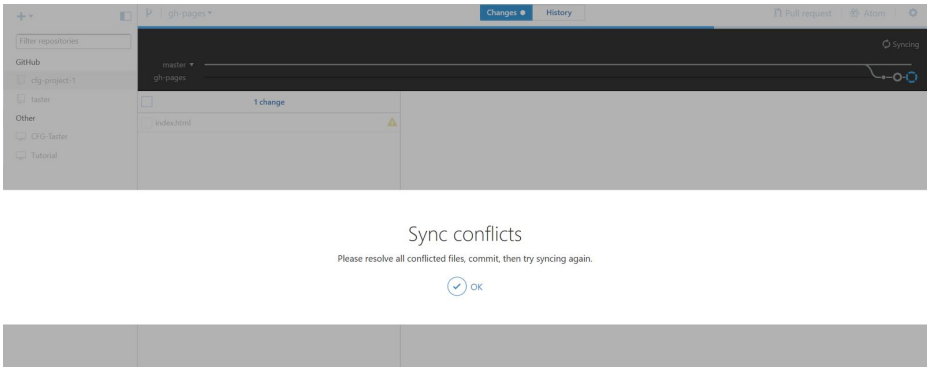
Chapter 18 - Conflict scenario!

Conflicts arise when you have multiple parties adding code to a shared repository, where one party is working on an old version of the code that has been updated.

Task: Setting up a conflict scenario (so we can explain to you how to resolve it later!)

1. Get into pairs.
2. Person 1 should invite person 2 as a collaborator to the repository they created for the project exercises (the repository created in chapter 15).
3. Person 2 should clone this repository to their local machine. Now to create a conflict!
4. Person 1 now needs make a change to their HTML or CSS files (eg. change some wording or a tag) and commit and sync those changes.
5. Person 2 should not sync but instead now make a different change to the files on their local machine.
6. Person 2 should now try to commit and sync those changes.

What happened? Oh no, you have a conflict!



So what can you do to fix this?

Sadly for Person 2, whoever syncs last must deal with a 'merge conflict'. Person 2 please follow these instructions to fix this:

Task: Fixing merge conflicts

1. Person 2 needs to open the files in a text editor (e.g. Atom).
2. The conflicts will have lines around them like this <<<<<<<<<<. And the two versions will be separated with marks like this ========.
3. Choose which version you want to keep and delete the other the correct lines to keep.
4. Edit the lines if necessary.
5. Remove the lines with arrows <<<<<<<<<<< and ==========.
6. Add and then commit and push the changes.

How to avoid and reduce merge conflicts

- Anytime you want make changes to code in a shared repository make sure that you sync first.
- It's worth planning with your partners which sections you each want to work on at any one time. This will avoid you both trying to make change to the same code at the same time.

Part 6 - Twitter Bootstrap

Chapter 19- What's hard about CSS?

Demo here. It shows the different cases of CSS needed to create a polaroid frame with a box-shadow that is compatible to all browsers.

You've seen quite a bit of CSS now; it all seems quite straightforward - you write some CSS, tweak it 'til it looks good and you're done! In theory this is exactly how CSS works and is why CSS is brilliant.

Unfortunately, the realities are not quite so straightforward. Different browsers will render CSS with subtle differences. Take a look at the cat picture. The styling is relatively simple - all we've done is add a border and a shadow.

Just because your site looks good in Chrome, doesn't mean it will look good in Internet Explorer. Making your site look good (or even presentable) in multiple browsers takes time, effort and experience.

What else is hard about CSS?

About 5 years ago, 'all' you would have had to worry about is the cross-browser display issues. Since then, the mobile web has exploded and you have another (far more important) concern: how will your site look when viewed on a mobile?

Making web pages that look good when viewed at multiple different sizes is a whole new level of complexity.

Chapter 20- Twitter Bootstrap

Twitter Bootstrap is a **Web Application Framework**; set of CSS (& JavaScript) files, released by the makers of Twitter, and maintained by some of its developers.

A framework is a skeleton of code providing generic functionality. It's a collection of code you can add to your website project to speed up development. And it has several advantages:

- You can use it to extend your own code
- You can override the framework code

Bootstrap provides provides a set of ready-made CSS files with pre-built functions for common web development and presentation requirements. All the solutions are cross-browser compatible - you don't have to worry about coding your own fallbacks - and it is responsive.

To make use of Twitter Bootstrap, you need to do two things:

1. Link to the Twitter Bootstrap stylesheet in the head of your html page.

2. Attach the relevant Twitter Bootstrap class to your html element.

To understand how to use Bootstrap, or any framework for that matter, it is **vital to read the documentation** (it's basically a guidebook). The documentation for it is here.

Here are some basic examples for using Bootstrap, take a look.

Responsive design

Responsive design means designing your sites so that they look good on **all screen sizes**.

Twitter Bootstrap promotes a 'mobile first' philosophy, encouraging you to design your site so that it looks **good at all sizes from the very beginning.** It provides a lot of useful CSS that helps you to do this.

Bootstrap includes a **responsive**, **mobile first fluid grid system** that lets you split the screen up into 12 columns and lets you customise the size of your HTML element as a fraction of 12. See this example for a easy layout option, and look at this example for responsive design – change the size of your browser to see the difference between the "xs", "sm" and "md" classes. Can you tell what they mean?

Getting started with Bootstrap

There are two ways to include Bootstrap to your projects:

- Link to the Bootstrap CDN
- Download the Bootstrap files and add them to your website folder

Bootstrap CDN

Let's first start with the easy option: using the Bootstrap CDN. A CDN is essentially a server on the internet where files have been placed for anyone to link to from anywhere. This means you don't need to download anything. You simply link to the location of the CDN and you're done.

You can find the urls for the Bootstrap CDN files in the Bootstrap documentation.

However using a CDN can be risky:

- **If for some reason your call to the CDN is not working, your crucial files will not be loaded by the browser.**
- **If the owner of the CDN moves the files to a different location and you are not aware of that, your crucial files will not be loaded by the browser.**

Both those reasons will break the layout and functionality of your website.

Downloading Bootstrap

A safer way to use Bootstrap is to download the files to your website folder instead. Again, you can find the Bootstrap files download in the Bootstrap documentation.

For our projects we are going to use the download method.

The best way of understanding how Bootstrap works is by using it. We are going to do a series of tasks to teach you how:

- To download the Twitter Bootstrap files
- Add the files to a website project
- Link the files in the HTML of the project
- Learn how to add the Bootstrap classes to the HTML
- And how to override the Bootstrap styling to our liking

TASK: Download Twitter Bootstrap and unzip the folder

Go to the download section on the Twitter Bootstrap website:

http://getbootstrap.com/getting-started/

You'll see three options. Ignore the Sass one.

- Bootstrap: we want this one, it's the skinny version of Bootstrap and contains all the files we need.
- Source code: The Bootstrap documentation contains a lot of ready made examples. If you want to use any of those, you will need to download the Source code files.

1. Click the Download Bootstrap button
2. Wait for the files to download and then go to your Downloads folder
3. Unzip the folder
4. Inside the folder you see three other folders:
    a. CSS: this contains all the CSS Bootstrap files you may need
    b. Fonts: Bootstrap has a set of icons you can use in your projects, and the font files for these are in here
    c. JS: this contains all the JS Bootstrap files

5. Place this folder on your desktop where you can find it back easily, we'll use it in a minute to add the files to a website project

TASK: Download the Sam's Sarnies exercise and add the Bootstrap CSS

We are going to download the files to create a website for Sam's Sarnies. And we are going to use this exercise to implement Bootstrap and make it look pretty.

1. First go to the GitHub repository to download the files: https://GitHub.com/code61/bootstrap_exercise
2. Click on the green Clone or download button.
3. Then choose Download ZIP
4. Go to your downloads folder, unzip the file and move the folder into your coding_course folder
5. Now go to the Bootstrap folder you placed on your Desktop
6. Open the folder and copy the **CSS** folder to the bootstrap_exercise folder we have just downloaded
7. Open your bootstrap_exercise folder up in Atom. Make sure you open the entire folder. If you need a refresher:
    a. Open Atom
    b. Go to File in the top menu and choose Add Project Folder...
    c. Navigate to the folder in your file system and click Open
8. Open index.html in Atom
9. Open index.html in Chrome as well so you can see what it looks like out of the box - nothing special.
10. Let's change that. Go to your index.html file in Atom and inside the <head> element add the following line of code:
    `<link href='css/bootstrap.min.css' rel='stylesheet'>`
    - Notice the href location of the CSS file, it is using a document-relative link
    - We added the entire Bootstrap CSS folder to our project folder
    - We are linking to the file from inside index.html, which is in the root of the project folder, hence: CSS, forward-slash, bootstrap.min.css

- We are also linking to the minified version of the CSS file. It contains exactly the same stuff as the normal CSS file, but the minified file has a smaller file size. Smaller file sizes load faster, which is better practice.

11. Go to index.html in Chrome and refresh the page. It's still not looking awesome, but you notice that the fonts have changed, a grey background appeared behind the hero image and the buttons at the bottom have grown a little in stature.
12. Finally add the following line of code inside your <head> element as well.
    <meta name="viewport" content="width=device-width, initial-scale=1">
    This is line will make your website projects mobile friendly. It will ensure everything is rendered properly and that touch zooming will work as expected. You should add this to any website you create, even if it doesn't use Bootstrap.

Chapter 21 – Bootstrap layout

The essence of Bootstrap

Now that we have added the Bootstrap CSS file in our project we can start using Bootstrap. Essentially Bootstrap is a huge collection of styles attached to CSS classes. If you want to add Bootstrap styling to your HTML you need to do the following:

- Go to the Bootstrap documentation and find the style you want to add
- Copy the Bootstrap class exactly as it is listed in the documentation
- Add the class to your HTML element

If you look at **index.html** from Sam's Sarnies you can see that it already contains one Bootstrap class. The **.jumbotron** class is responsible for the grey background behind the hero section.

Bootstrap layout

Let's have a look at layout. The Sam's Sarnies website currently has all the content sitting on the left hand side of the viewport. To change that we'll need to do something about the visual layout of the page.

Bootstrap has several options for page layout. Go to the Bootstrap documentation page and in the menu on the right hand side click on Grid system. Grids are used to create a page layout. The Grid section has a lot of options. Some of these options will create a responsive design out-of-the-box. Others will need some adjusting.

Keep the Bootstrap documentation open in a tab in your browser while we are going through all the upcoming tasks. We will be referring back to it a lot.

Page margins

Bootstrap makes it easy to center content inside the viewport by using the **.container** class. You can add the class to a ** ** element and then "contain" all the HTML you want to be centered inside this ** .**

You'll be using container divs a lot in your Bootstrap projects.

Columns

Inside a .**container** you might want to divide things up further and maybe create columns. Bootstrap works on a grid layout with 12 columns. To create a column layout you use a combination of **.row** and **.col** classes

1. First you need to create a **.row** which is going to contain your columns. You do this by adding a ** ** with a class of **.row.**
2. Inside this **.row** div you then create other divs, one for each column.
3. **Each column `<div>` needs a class starting with `col`**
4. The next part of the class indicates from which viewport width this class applies. You can see a table of the Grid options in the Bootstrap documentation.
    a. -**xs-** is for extra small devices like phones
    b. **-sm-** is for tablet devices
    c. **-md-** is for medium sized desktops and laptops
    d. **-lg-** is for large desktop devices
5. The final part is the number of grid columns you want your element to be. The grid layout has 12 columns in total, so if you want to create a three column layout each column would need to be 4 grid columns wide.

Columns

Example of two even sized columns

```
<div class="container">
  <div class="row">
    <div class="col-xs-6">
      <!-- Column content goes here -->
    </div>
    <div class="col-xs-6">
      <!-- Column content goes here -->
    </div>
  </div>
</div>
```

This is an example of two even sized columns would look like in your HTML.

TASK: Creating a row on Sam's Sarnies page

We are going to add some of this to Sam's Sarnies.

TASK: Adding a .container div

Add a .container div around the page content of Sam's Sarnies

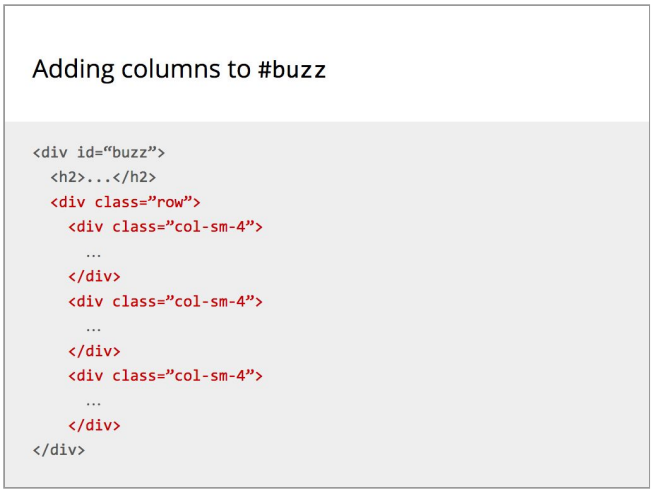
```
...
<body>
  <div class="container">
    <div class="jumbotron">
      ...
    </div>
    <div id="buzz">
      ...
    </div>
    <div id="mission">
      ...
    </div>
  </div>
</body>
</html>
```

First add a .container around the page content and have a look at the effect in Chrome. All the page content is now centered inside the viewport

TASK: Adding columns to the .jumbotron

1. In the .jumbotron at the top of the page we are going to create two columns to contain the heading and the image.
2. First create a .row div around the <h1> and the <img>
3. Now create two column divs, one around the <h1> element and one around the <img> element
4. Make the column with the heading twice as wide as the column with the image
5. We are using the -sm- class extension so that the columns will appear on devices with a minimum viewport width of 768px. That's tablets and bigger.
6. Save your file and see the effect of this in Chrome.

TASK: Adding columns to the #buzz


Adding columns to #buzz

```
<div id="buzz">
  <h2>...</h2>
  <div class="row">
    <div class="col-sm-4">
      ...
    </div>
    <div class="col-sm-4">
      ...
    </div>
    <div class="col-sm-4">
      ...
    </div>
</div>
```

Let's do something similar on the #buzz . If you look inside the buzz div you will see four different sections: a header element and three elements, one for each testimonial

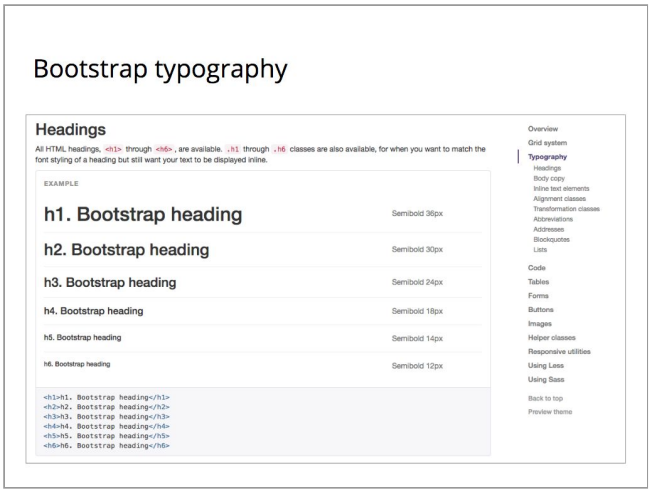
1. Create a containing with a row class around the three testimonials divs
2. Give each a class of col-sm-4, this will create three equal columns
3. Save your file and look in Chrome to see your three columns

Chapter 22 - More Bootstrap

We have now adjusted the layout of the page to look more like a real website. And all we have done is adding some classes to HTML elements and the Bootstrap CSS is doing all the hard work.

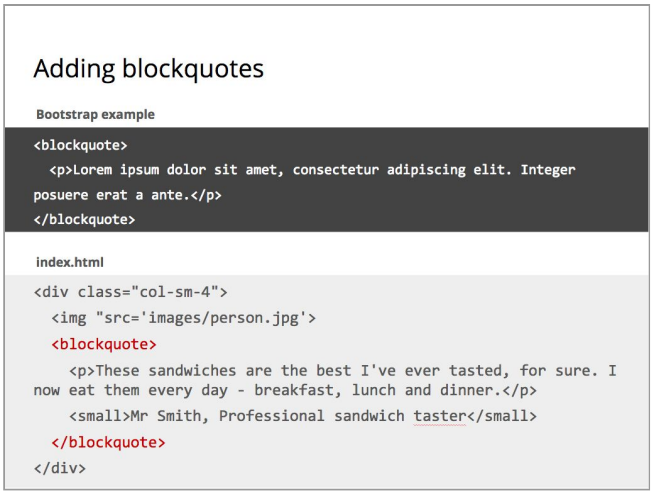
Let's do some more.

Typography



Have a look at the typography section of the Bootstrap documentation. Click on Typography in the menu on the right hand side. You can see why the headings on our Sarnie website are looking the way they do. Bootstrap CSS is setting the font-size and font-weight.

Blockquotes



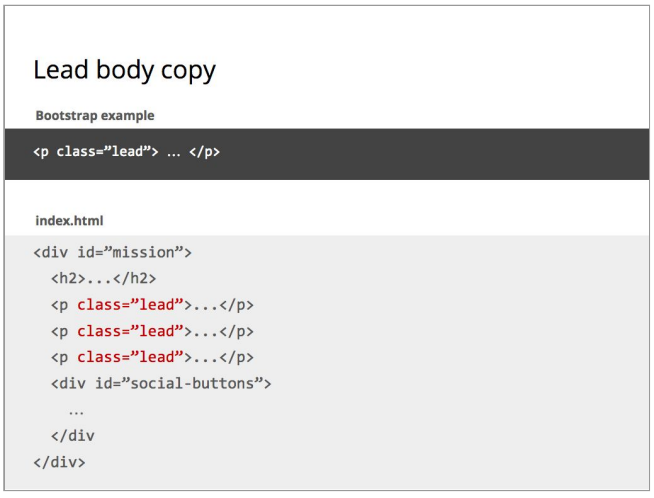
In the **Typography** section of menu on the right hand side click on **Blockquotes**. We are going to use the Bootstrap code to make our testimonials look better on the Sarnies website.

Task: Adding blockquotes

1. Look at the Default blockquote section in the Bootstrap documentation. That's what we want to do with our testimonials
2. Inside the testimonial divs add a <blockquote> element around each quote. Include both the <p> element with the quote and the <small> element with the person's name - if there is one
3. Make sure you do this for each testimonial

We mentioned before that Bootstrap is based on adding CSS classes to HTML elements. And as you notice the **<blockquote>** we added does not contain any CSS classes. In this case Bootstrap is simply providing CSS styling for a HTML element.

Lead body copy

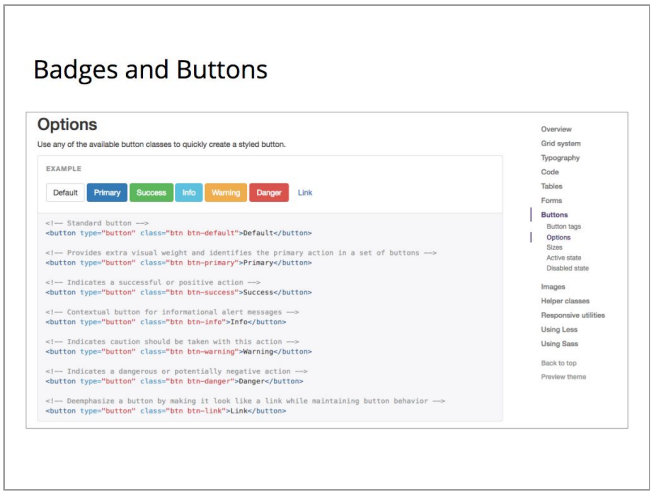


Task: Adding lead body copy

Go back to the Bootstrap documentation and in the Typography section of menu on the right hand side click on Body copy. Have a look at the Lead body copy section. If we add the lead class to an HTML <p> tag the text will have a larger font-size and stand out.

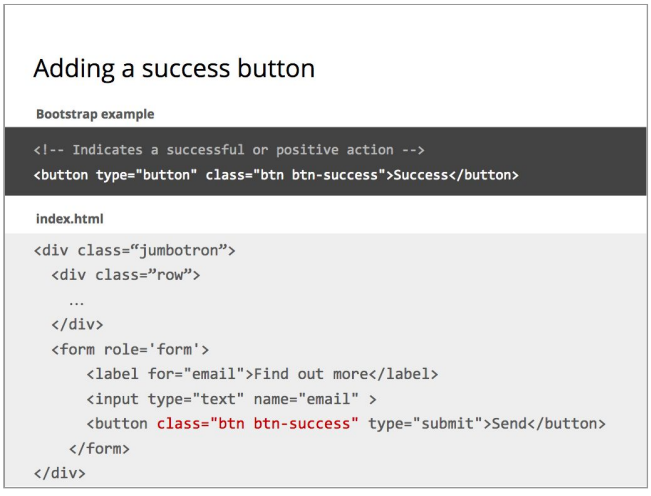
1. In your Sarnies website go to the #mission and add the lead class to all the <p> elements.
2. Save the file and look at the changes in Chrome.

Badges and Buttons



Next up are buttons. Go to the Bootstrap documentation page and in the menu on the right hand side click on Buttons. In the Options section you can see several different types of buttons you can use. We are going to add extra classes to all the buttons on the Sarnies website.

Success button



We'll start with the Send button at the top of the page and we are going to make it green to look like a Success button.

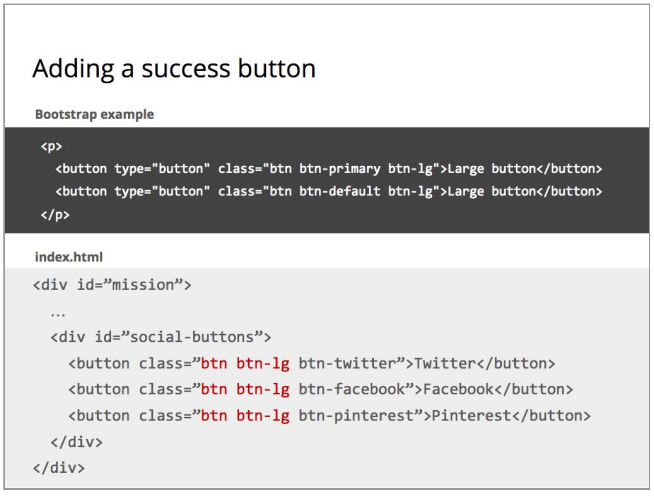
Task: Adding a success button

1. In your code find the send button, it sits inside the .jumbotron

2. Look at the example code in the documentation and add the same classes to your send button

You are adding two classes. The first one, `.btn`, is the generic class that will style a button to look like a button. The second class `.btn-success` gives extra styling to the button and in this case that means colouring it green.

When you add more than one class to an element you separate each class with a space.



Social links buttons

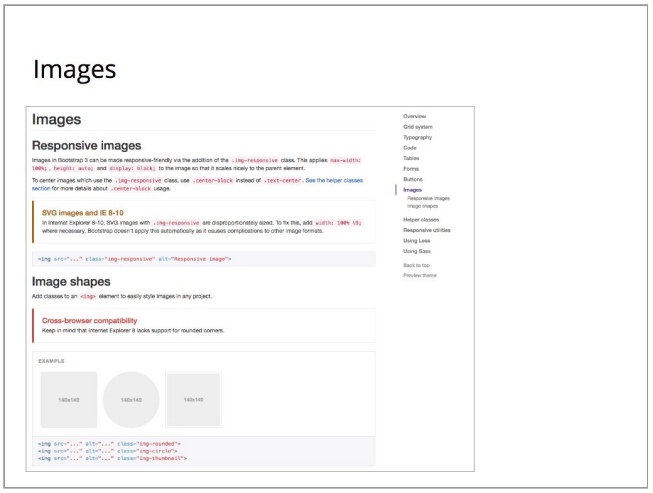
Go to the bottom of your HTML and find the `#social-buttons <div>`. You can see those buttons already have classes added to them. We will be using those later. For now we want to make the buttons larger. If you look at the Sizes section of the Bootstrap Buttons documentation you have an option for 4 different button sizes.

Task: Adding social link buttons

1. Add the classes btn and btn-lg to your social media buttons
2. Be sure to keep the existing classes as well. You should end up with three different classes on each button
3. The order of multiple classes on one HTML element doesn't matter

Save your file and look in Chrome to see the effect.

Images



Next up are images. Go to the Bootstrap documentation page and in the menu on the right hand side click on Images.

Responsive images

If you look at Sam's Sarnies in Chrome you can see that the sandwich image at the top appears too big for the container in which it sits. If you make your browser window a bit smaller you can see more clearly.

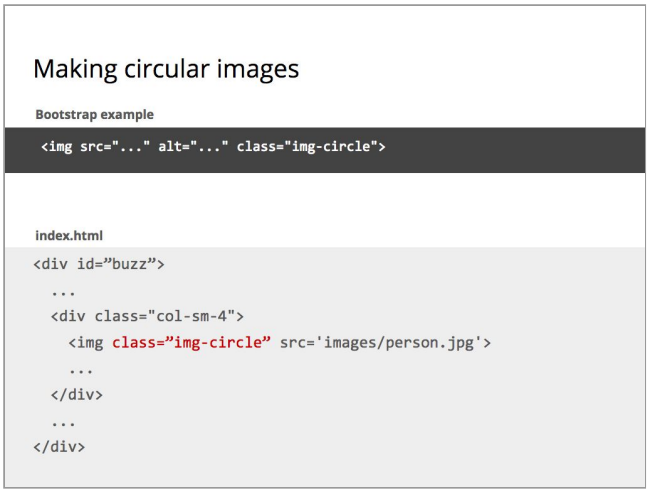
Making images responsive

```
<div class="jumbotron">
  <div class="row">
    ...
    <div class="col-sm-4">
      <img class="img-responsive" src="images/sandwich.png">
    </div>
    ...
  </div>
</div>
```

We can fix this by adding a img-responsive class to our sandwich image. This class will take care of the scaling of the image so that it doesn't grow outside the container.

Save your file and have a look. Adjust your browser window to see it working.

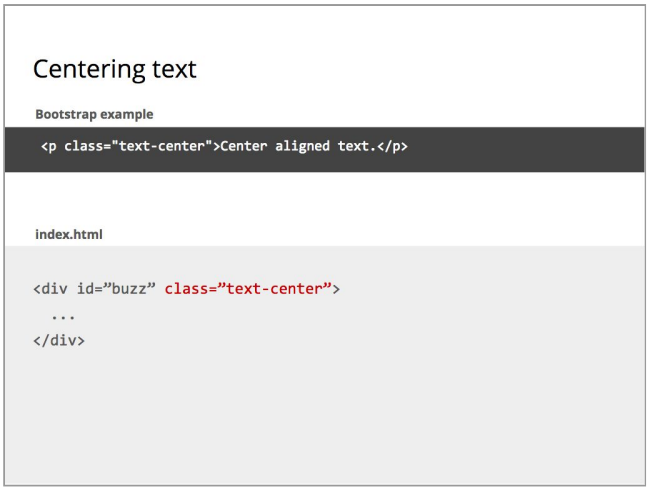
Round images



We are going to make the images of the three people circular. In the Images shapes section you can see that adding a class `img-circle` to the `<img>` should do the trick.

In your HTML go to the `#buzz <div>` and add a class `img-circle` to each image. Save your file and look in Chrome at the now rounded images.

Center text



As an additional trick we are going to center all the content inside the `#buzz <div>`. Go to the Bootstrap documentation page and in the menu on the right hand side click on Typography again. Then underneath Typography click on Alignment classes.

Task: Centre text

To center elements you need to use the text-center class.

1. In your HTML file go to the #buzz
2. Add a class with a value of text-center. This will center everything inside that div.
3. Save your file and have a look in Chrome

Making a stripy table

Task: Go to this code pen https://codepen.io/dianaklee/pen/yYVbzv and remove the Bootstrap classes table and table-striped from the HTML

We have added a fair few nifty little things to our Sam's Sarnies web page. Let's have a look at a CodePen demo to create a stripy table. You remember from our first exercises that HTML tables don't look like much without additional styling.

Tables are not an easy thing to style nicely when you're going to write your own CSS. But Bootstrap is going to help us out here. If you look in the Tables section of the Bootstrap documentation you can see that there are various table classes we can use to make a table look more attractive.

TASK: Add the Bootstrap classes table and table-striped back in the HTML

In this example a stripy table is created by adding the classes `table` and `table-striped` on the `<table>` element. And that's all that is necessary to turn a dreary table into a much nicer looking one.

Chapter 23 – Modifying Bootstrap

Remember what Sam's Sarnies looked like earlier? Much better now, don't you think? There are plenty things left to do, but I hope you are beginning to get a feel as to what is possible by using Twitter Bootstrap.

But maybe you don't like how Bootstrap makes some things look? Also, imagine if all websites in the world used Twitter Bootstrap out-of-the-box without tweaking anything. The web would look a little tedious.

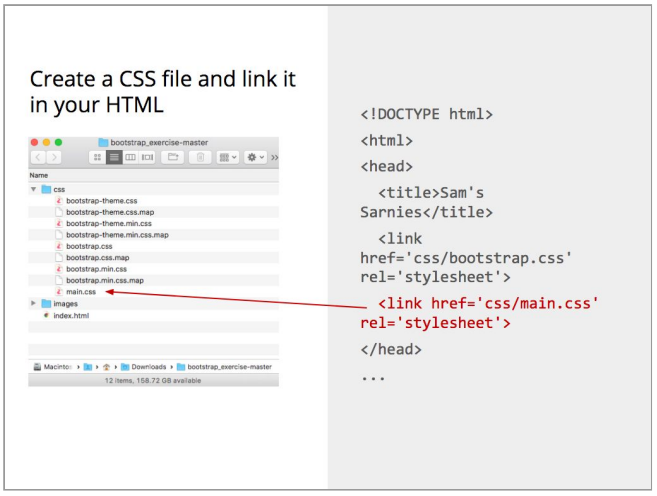
Well, here is the good news. We can override the styling Bootstrap gives us with our own styles. But a word of warning first.

Bootstrap has been designed and heavily tested for good cross-browser compatibility. Unless you know what you are doing, or have a lot of time, it's probably best to stick with the Bootstrap layout and keep your overrides to fonts, colours and general small things that leave the layout structure well alone.

The correct way of overriding Bootstrap styles is by creating a new CSS file of your own and linking to it in your HTML. Never ever make changes in the Bootstrap files themselves.

- Always link your own CSS file after the link to the Bootstrap file
- Use the Bootstrap CSS classes to style elements as you wish

TASK: Create a CSS file for overrides and link it in your HTML



Let's create some CSS overrides, but first we need to create a CSS file.

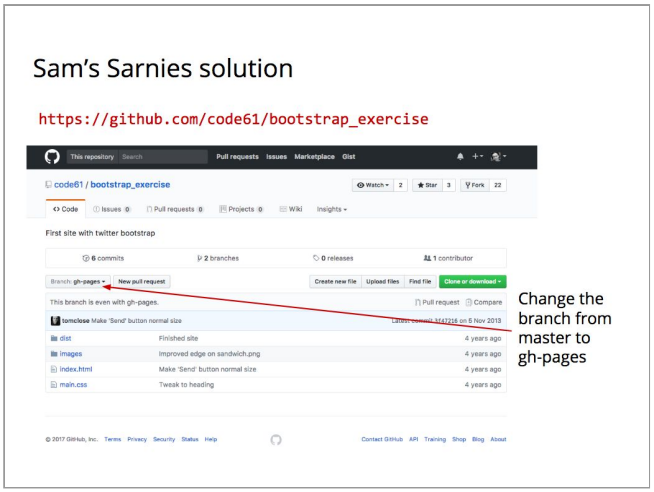
1. Go to your project in Atom and inside the CSS folder create a file called main.css
2. In index.html link to your new main.css file. Make sure the link sits after the link to the Bootstrap CSS file

Maybe you are wondering why the link to your own CSS with overrides needs to come after the Bootstrap CSS. Maybe you remember from the previous class that CSS stands for Cascading Style Sheets. It means that a browser will apply the CSS rules in the order it comes across them. If you add your Bootstrap file first, then a browser will apply the rules in there. If Bootstrap is followed by another file, then a browser will re-apply any rules it finds in there. Because we want to override Bootstrap we need to add our styles last.

Sam's Sarnies solution

So far we have improved Sam's Sarnies web page by adding Bootstrap classes to our HTML. But you can still do a lot more if you wish. If you look in your course notes on page 48 is a link to what Sam's Sarnies should look like. For homework go ahead and finish of the exercise.

If you get stuck you can go back to the GitHub repository for Sam's Sarnies. Change the branch name from master to gh-pages and you can see the solution.

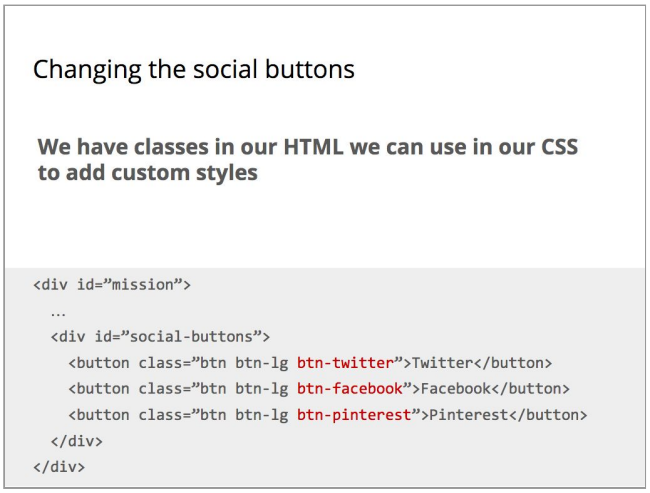


NOTE: The following sections (until the JavaScript & jQuery section) are additional tasks you can do if you have extra time.

Even more awesome stuff

The next step for Sam's Sarnies web page is to start making use of that custom CSS file we created and begin to override some of the Bootstrap styling.

TASK: Change the buttons to social buttons



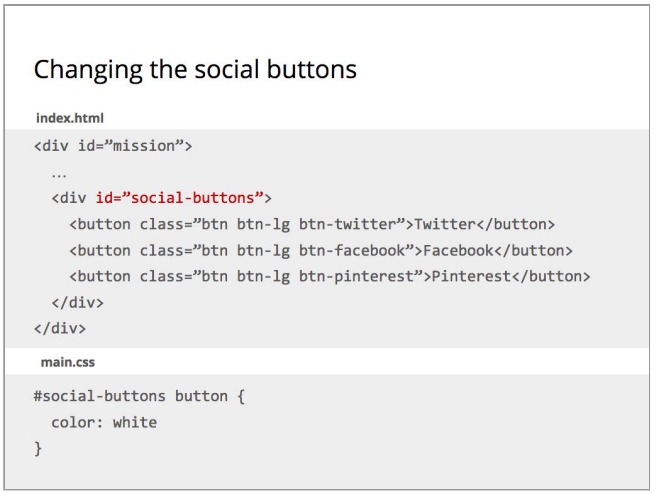
Now that we have our own CSS file we can finally start to write some CSS.
We'll begin with adding custom colours to the social media buttons at the bottom of our page.

Look at the code for the buttons in index.html. Each button has three different classes. Two are Bootstrap classes: `.btn` and `.btn-lg`. They are responsible for making the buttons bigger and adding rounded corners. For our purposes we are interested in the third class: `btn-twitter`, `btn-facebook` and `btn-pinterest`

1. **In your CSS create three rule-sets for each button and add the correct background colour and border colour for each button. They are different for each button.**
2. **Save the file and look at the effect in Chrome. You can see the buttons are now coloured in the social media brand colours.**

But the text is still black and we want it to be white. All three buttons have two other classes in common, we could either of those to fix our problem, but if we do, then any other button using those classes will change colour too.

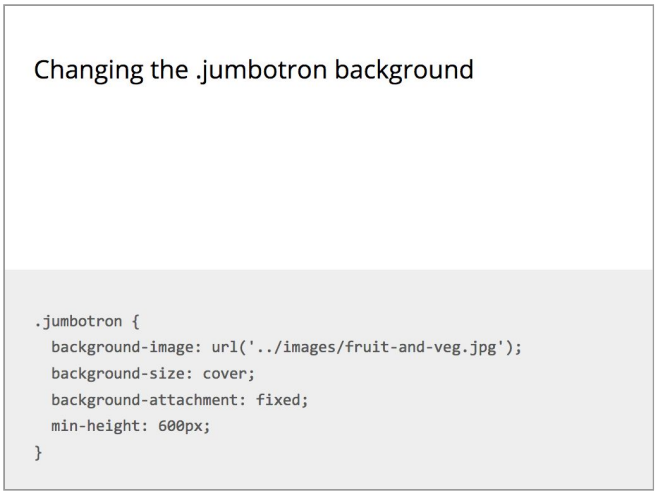
We need something more specific.



The buttons sit inside a div with and ID of `social-buttons`. An ID is unique on the page, there is no other element using that ID. This is perfect for our needs.

In your CSS add a rule to colour the text of the buttons inside `#social-buttons` white

TASK: Changing the jumbotron background



Next up we are going to get rid of that sad grey background behind the jumbotron and replace it with the fruit and vegetable picture in our images folder.

Add the CSS for the background image to your main.css file. There are a few things to take note of here:

- **The file path of the image is again a document-relative link. You are linking to the image from your CSS file. The CSS file sits inside a folder called /css and the image sits inside a folder called /images. The file path is instructing the browser to go up one folder in which the CSS file is, and then find another folder called /images and in ther locate fruit-and-veg.jpg**
- **The filename of the image is case sensitive and includes the file extension. A typo or the wrong file extension will result in no image being shown by the browser. Browsers are picky things. If something is not exactly right, it will skip over it and ignore it.**
- **Background-size and attachment will make sure that the image covers the entire container inside which it is displayed.**

Save the file and have a look in Chrome. It will look good, but not yet good enough. We want the background image to stretch from edge to edge of the browser window. If you remember earlier we added a `.container <div>` around all the content of our page. The `.container <div>` centers all the content inside the viewport.

TASK: Adding .container divs inside the three main page sections

Moving the .container divs

```
...
<body>
<div class="jumbotron">
  <div class="container">
  </div>
</div>
<div id="buzz">
  <div class="container">
  </div>
</div>
<div id="mission">
  <div class="container">
  </div>
</div>
</body>
</html>
```

To fix our problem we will need to move .jumbotron to be outside `.container`. This will make the background image bleed from edge to edge. But that's not enough. We also need to create a new `.container <div>` inside `.jumbotron` so that the content in there is centered inside the viewport.

Go ahead and update your HTML. You should do the same for the `#buzz` and `#mission` divs.

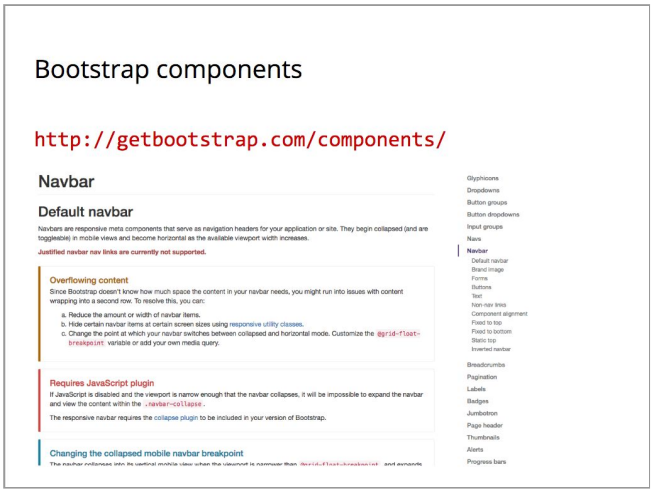
TASK: Change the background colour of the mission section

Changing the background colour of #mission

```
#mission {
  background: rgba(32, 35, 41, 0.9);
  color: #ddd;
}
```

Now each section is separate we can add a full width background colour to the #mission section.

In your CSS add a rule-set to change the background colour of the mission section and the font-colour as well.



Navbar

We are nearly there with Sam's Sarnies. One last thing we need to do is add a navigation section at the top. Go to the Bootstrap documentation page and at the top of the page select the Components page: http://getbootstrap.com/components/
In the sidebar menu on the right click on Navbar.

TASK: Add a navbar to your page

Scroll down to the section called Inverted navbar.
That's what we are going to create.

1. **Right click on the black nav bar and choose Inspect to bring up the Web Inspector.**
2. **Highlight the line of HTML beginning with `<nav class="navbar navbar-inverse">`**
3. **Right click again, with the HTML selected, in the pop-up menu select Copy and then Copy Outer HTML**
4. **Go to your index file and paste the code at the top of your HTML `<body>` section, above the `.jumbotron <div>`.**
5. **It may have copied it all in one block. You can place all code on a separate line to help you understand it.**

We also want the navigation to stay at the top of our page. Scroll up in the Bootstrap documentation to the section titled Fixed to top.

To fix a navigation bar at the top of the page we need to add a class to our navigation bar and we need to create a container div inside the navigation

1. **Add the `navbar-fixed-top` class to your HTML nav element**
2. **The code we copied already contains a `.container <div>` inside the `<nav>` element. It has a class of `.container-fluid`, which makes it sit at the far left of our viewport. Change `.container-fluid` into `.container`**

A fixed to top navigation also needs some additional CSS in your main.css file.
Add the extra CSS at the top of your main.css file. This CSS will add extra padding to the top of the `<body>` element so that the content of the `.jumbotron` sits below the nav bar.

Adding a search box

We want to add a search bar inside our navigation block. Scroll further up the Navbar section of the Bootstrap documentation until you get to a part titled Forms.

1. **Copy the code for the form (no need to go via web inspector)**
2. **Place it inside the `.collapse.navbar-collapse <div>`. That's just above the first of two closing divs at the bottom of the `.navbar <div>`**
3. **If you save the file and look in Chrome you will see that the search box sits on the left next to the navigation links. We want it to be on the far right.**
4. **In the Bootstrap documentation go to the CSS page and in the menu on the right hand side click Helper classes and then Quick floats**
5. **To align something to the right you can use a `.pull-right` class**
6. **Add the `pull-right` class to your search form element**
7. **Save the file and look in Chrome to see the effect.**

Part 7 – JavaScript & JQuery

Chapter 24 – JavaScript Preamble

We have reached the final stretch of the of our Introduction into Web Development course.

What have we learned so far

- We had a brief introduction how the Web works;
- We learned how to display content online in a structured way using HTML;
- We learned how to style and design that content using CSS;
- We learned how to work collaboratively on a website using GitHub and Git;
- And we learned about tools developers use to make their coding lives easier in the form of Twitter Bootstrap.

So what's next? We are going to dive into programming for the web by learning about JavaScript and jQuery.

How JavaScript Came About

Back in the beginning of the the Web, when the first browsers were being created (Netscape vs Internet Explorer), Marc Andreessen (the co-creator of the Mosaic/Netscape Navigator browser, whose engine is now used to power Firefox, and co-founder of Netscape) realised the Web needed to become more dynamic - It was all HTML and CSS at the time, and there was limited interaction. So they experimented using different languages that had already existed to allow data to be dynamically manipulated through websites, and were not too happy with the results. In short, they recruited Brendan Eich to try embedding one of these languages, but in the end, he decided to create a language that would be prototype-based, and JavaScript was born.

But wait! What even is Programming?

Hold on! What even is programming? There are so many different programming languages available - why do they all exist, and where do I start? And OMG I'M SO CONFUSED.

To be concise:

1. **Programming** is the way humans tell computers to do logical things for them in a systematic fashion. It lets humans create ways for other people to interact using computers.

2. There are a wide variety of programming languages, BUT to decide which to use, we need to decide what we want to use it FOR, and then **pick the language best suited for the use**.
3. Programming languages are organised into **paradigms**, that is - ways of thinking and communicating with the computer.
4. JavaScript is powerful because it is:

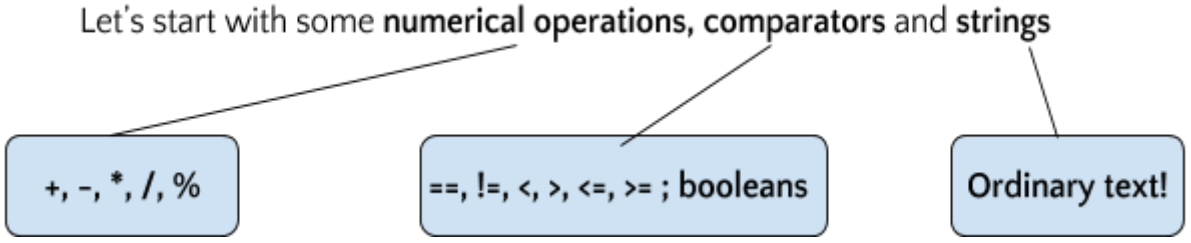
a. **multi-paradigm** (it can accommodate a range of thinking/programming styles),
b. **prototype-based** (things that you have defined can be changed easily #datMVPlife),
c. **designed to be used for the Web** (whereas other languages have been designed for scientific computing, e.g. Python - although Python is a very flexible language too, or native application development, e.g. Swift / Objective C for iOS, and .NET for Windows),
d. and finally, it is a **full-stack language** (it can be used **both** on the **client** and **server** - other languages, such as Python or Ruby, are server-side languages).

In essence, it is **super flexible**, and therefore super awesome 🙌😊🙌

Chapter 25– JavaScript & jQuery

So... how do I start using JavaScript?

Good question! Because JavaScript is a standard Web technology, you can write it directly in your browser! Let's open our Developer Tools and start writing some JavaScript!



All three of these can be used to form an **expression**; which is an **operation** performed on a **data type**.

A **function** is a **group of expressions** which **come together** to **perform a particular task** when it is **called upon** (more formally, invoked) by an an **action** (more formally, an event). This can be an action made by the user, or by the server, or some sort of external event which your program is listening to.

A **variable** is what JavaScript uses to **store, organise and manage raw data** which is being handled by functions & expressions. Variables are therefore manipulated by

functions and expressions. A function takes a variable (or sometimes exact / raw data), digests it and does something to it, and returns a changed value.
JavaScript uses semicolons, like CSS does, to decide when to stop executing a particular task.

If you want to write JavaScript in your webpages, there are a couple of ways of doing it:
- You can link it using the same way we have been linking our CSS files (***recommended)***
- Or you can write it within <script> tags directly in your HTML file.
-

And what is jQuery?

JavaScript can be quite complicated to learn, and tedious for basic functionalities. jQuery is a **JavaScript library** that is useful for building interactive web pages.

Recap: A **library** is an implementation of an API; it is a set of functions that a developer can call, usually organised into classes. It contains the compiled code that implements the functions and protocols (maintains usage state).

jQuery is so common in web pages that, for beginners, 'learning JavaScript' has in many cases become 'learning jQuery'. This is the approach that we're going to take in this course.

Getting jQuery into your website

jQuery is just a JavaScript file that can be **downloaded** from the jQuery downloads page. There are a couple of different versions – I'd go for the latest. **Alternatively,** you can use the **JQuery CDN,** and **link directly** to their hosted online version.

Let's go through together and link the JQuery library to your website by following the below steps:

You include the JQuery library in your site by adding the following in between your <head></head> tags:
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>

When you're putting the JQuery code you write in your page, you can either write a separate .js file, or you can just put it in between **<script></script>** tags in your HTML file.

Please note if you're trying to use the Bootstrap JQuery plugins, you need to include both the JQuery library and Bootstrap's JavaScript, so add this code to your HTML file:

```
<!-- jQuery library -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
```

```
<!-- Latest compiled Bootstrap JavaScript -->
<script src="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>
```

W3Schools has good documentation of getting started with Bootstrap: http://www.w3schools.com/bootstrap/bootstrap_get_started.asp

Using jQuery to manipulate CSS

You can do a lot of stuff with jQuery. Here we'll just look at the basics: selecting elements on the page and doing stuff with them.

Read the API documentation here, and give it a shot

You can **experiment** with jQuery using the Console section of the **Chrome developer tools**. You will need to be on a page where jQuery is loaded (e.g. these course notes or the demo page you will download in the exercise).

One of the nice things about jQuery is its ability to select elements via their CSS selectors. To select elements jQuery uses the $(' ') function. For example:

```
$('li').css('color','blue');  // selects all the li elements on the page

$('li .important')    // selects all the li with class="important"

$('#main-title')    // selects the element with id="main-title"
```

jQuery then has several ways of manipulating those elements.

In Class Demo:

1. Use this demo to see how an element can be manipulated by un-commenting the lines of jQuery in the JS section
2. Can you make the list items go green?
3. Let's try another demo: See how you can input text onto a page using jQuery

Task: Adding a navbar with a search box to your site

1. Look at the html for the basic starter template and the navbar section of the Bootstrap docs.
2. Use it to add a navbar to your site.
3. Make it a navbar-fixed-top. You will need to add

```
body { padding-top: 70px; }
```

to main.css.

4. Add a search box to the navbar. Use the pull-right class to put it on the right hand side.

Read the API documentation here, and give it a shot.

Course Competition time!

We've now covered everything we need to finish up your awesome group projects, in time to present them at the end of the session! Before we ask you to continue working on your projects, here's a reminder of the course competition criteria:

Course competition criteria recap

The criteria we'll be looking at are as follows. You can see the **Must have** criteria and the **Nice to have**. It would be great to have as many Nice to have features as possible, but don't worry we know you won't have time to include it all! Anything you don't get time to include now, can always be added later on.

Must have:

- A live website published on GitHub pages
- A minimum of **two** HTML files for:
    - 1 x landing page linked to a separate CSS file
    - 1 x 'about' page
- A minimum of **one** CSS files
- Good formatting
    - Code split into the appropriate files (separate HTML files & CSS files)
    - Files indented properly
- Good organisation
- Version control using git
- Sensible git commit messages

Nice to have:

- A visually appealing design - good use of CSS and HTML elements, Twitter Bootstrap, Jquery & JavaScript
- A contact form (for example name and email)
- Social buttons
- As many different HTML elements you can manage
- Interactive elements (like forms) on your website don't need to be functional, but should be present if they need to be for the visual aspect of the design.
- A responsive site

So that's it! Other than that, you can make any kind of website you want.

You can see a few examples of what previous students on your course created on our website here: http://www.codefirstgirls.org.uk/course-competition.html

If you're short on ideas, here are some to get you going...

- A personal website
- "How to" website on an area of your expertise
- A survey or poll website.

The websites will be judged by your instructors at the end of the last session.

Task: We'd like you to now get into your groups and work on the following:

- As you continue to carry out the next tasks please ensure you save, add, commit & push your code (refer to Part 3, Ch.13 -17 notes & the Cheat Sheet below). Avoid conflicts! Don't forget these steps:
  - Save your changes.
  - Add your changes.
  - Commit your changes.
  - Push your changes
  - Publish your repository on GitHub pages.
- Continue to work on your 'landing-page.html' file & linked 'landing-page.css', make sure your landing page includes:
  - HTML DOM (refer to Part 1, Ch.1 -5 of the notes).
  - A title (refer to Part 1, Ch.1 -5 of the notes).
  - Subtitles (refer to Part 1, Ch.1 -5 of the notes).
  - Different colours on the title & subtitles (refer to Part 2, Ch. 6 -12 of the notes).
  - A navigation bar (refer to Part 7, Ch.25 tusk 1)
  - Optional additions to your landing page can be found below in 'Course competition inspiration!)- see if you're able to include any of them!
- Work on the 'about-page.html':
  - Add a title (refer to Part 1, Ch.1 -5 of the notes).
  - Add some content (e.g. About the team/ About the site).
  - Optional additions to your landing page can be found below in 'Course competition inspiration!)- see if you're able to include any of them!
- Use Bootstrap, JQuery and JavaScript to add some more cool features to your site. You can find some further inspiration and guidance below.

Appendix

Course competition inspiration area!

Google Forms

Google Forms uses the <iframe> tag to embed a mini-form document into your web page, where you want it. This can be a quick and easy way to collect information from users on your website via online forms. Does this sound like a good addition to your website? Then follow the below instructions to embed your Google Form for the course competition!:

Task: Create a Google form and linking to your HTML file

1. If you don't already have one create a Google account, so that you can then log directly into Google Forms. Or alternatively click on the Drive icon, then click more and you'll see a link for Google Forms.
2. You'll then be presented with a selection of forms you can use, either a blank form or a template form to add to.
3. Using either of the forms, you'll be able to easily add in your own content to the form (e.g. changing the fields depending on the questions and data you'd like to collect, and amend the appearance of the form). Full instructions can be found here.
4. Be sure to configure how your form functions and is accessed (e.g. enabling multiple answers responses from a single user and customising the submission confirmation message for example).
5. Once you are happy with the form you can make it available by clicking on the "Send" button in the top right corner of the screen. You can then choose how you share the form, via email, a link, or embedded on a web page. Click the "<>" icon to "Embed HTML".
6. Copy and paste the link provided by this icon, which will look something like this: <iframe src= "https://docs.google.com/forms/d/[...] </iframe>". Add this code into the relevant HTML code where you'd like your form to appear.

Awesome features of Bootstrap

Drop-down menus

Check you've included the Bootstrap and JQuery stuff above, then look at this section: http://getbootstrap.com/JavaScript/#dropdowns

Popup boxes

In web development, popups usually refer to those things that come up in separate little windows on your computer, whereas the boxes that popup inside the website are known as 'modals'. Check you've included the Bootstrap and JQuery stuff above, then look at this section: http://getbootstrap.com/JavaScript/#modals

Image slideshows/carousels

Lots of sites include big revolving images at the top of the page, or maybe images of a product you can click through to see more of. Bootstrap already has some of these options included. Check you've included the Bootstrap and JQuery stuff above, then look at this section: http://getbootstrap.com/JavaScript/#carousel

A form to send an email:

Forms can be made using HTML and formatted with CSS. There are two main functions a web form can have: to send data to a database or to generate an email. Seeing as we're not covering databases in this course, this might be useful for your project if you wanted your site to have a space for feedback, requests or submissions that could just be sent to your email (though without a server they won't actually do that).

This is an example of a simple HTML form:
http://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_mail

Here is W3Schools section on input types (what you need for forms):
http://www.w3schools.com/html/html_form_input_types.asp

Mozilla (the people who make the Firefox browser) also have an HTML forms guide:
https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms

If you're using Bootstrap, then this form will look pretty neat without you doing any additional styling. Further info on Bootstrap forms can be found here and in the Bootstrap documentation.

Hover effects when you hover over links or buttons

If you're using Bootstrap, you'll have noticed that buttons have nice rollover effects i.e. when you hover over a button it changes style or colour. You can also set the link hover colour with the CSS :hover selector.

This is W3Schools section on hovers:
http://www.w3schools.com/cssref/sel_hover.asp

These are some demos and explanations of what can be done from Codrops: http://tympanus.net/codrops/2013/06/13/creative-button-styles/

Some ideas for jQuery

If you're wanting to use jQuery to do a specific thing and already using Bootstrap, you might want to check out Bootstrap's jQuery plugins which include a lot of common stuff like dropdowns, popups, image carousels etc. If you're wanting to learn about how jQuery works and understand what's going on in it a bit better, the below will hopefully help.

A simple todo list

This is just a list of things, with a text input and a button. When the button is clicked an event is triggered and our code appends another list point to our list. Because we're doing this in the browser, we don't have a way to store this, if you refresh the page, or if someone else goes to it, they won't see what you've done (think of it more like writing on a whiteboard that gets erased when you've finished, or an etch-a-sketch). Example: http://codepen.io/anon/pen/JYxGpv

Changing the order of things in lists (like upvoting)

This is a list with links inside that, when you click it, moves the item up in the list. Again, as with the example before, because we're doing this in the browser, it won't stay after you move away from the page. You could make this a lot more awesome with more interesting styling. Example: http://codepen.io/anon/pen/EVrPeE

Next steps

As the sessions have now ended we can't make any of this compulsory! However, if you've enjoyed what you've done so far and are keen to learn more then you can continue your coding adventure with the below resources!

Group Project

Task: Got the bug for coding? Continue to work on your project! You can continue to improve your sites by adding more features to them. To help you carry on with the good work you can check out the below additional resources.
Who knows- maybe you'll even want to start a brand new project!

Further Resources - HTML

- This video talks about how the Internet works in 5 minutes
- A summary of the different components of the Internet
- File organising for your website
- Introduction to servers by Eli the Tech Guy
- An article from Mozilla's Developer Guides: Introduction to HTML
- W3 Schools HTML Tutorial
- HTML Terms Glossary
- HTML DOM
- Web Monkey HTML Cheatsheet
- Simple HTML Guide Cheat Sheet
- A HTML Validator that checks your HTML code

Further resources - CSS

Definitions of CSS Terms , W3 Schools CSS Tutorial , Shay Howe's HTML & CSS Tutorial/Guide
This article which explains about CSS Specificity (and more).
This article has more information on CSS selectors.
A CSS Validator

Further resources - GitHub

How-To Geek explains GitHub , GitHub Guides
An introduction to Version Control
Another (Visual) Introduction to Version Control

Further resources - Git

A Git Cheatsheet from GitHub
A Cheat Sheet & Reference from Git
A Non-Programmer's Guide to Git
Pro Git
Think Like a Git
GitHub List of References
A Visual Git Guide
A Mac-Specific Guide

Further Resources - Bootstrap

Bootstrap provides some JS functionalities as well, built on jQuery too.
Read more about Bootstrap here:
What's the difference between a Framework and a Library?
Ben Howdle talks about different JS Frameworks
The Mozilla Web Developer Guide

Further Resources - jQuery

Obviously, we've only just scratched the surface of what's possible with jQuery. Things get a lot more interesting when you can create bits of JavaScript to be run in response to a user action.
This allows you to build up interactions like "when the user clicks the submit button, check that their email is a valid email, if it isn't make the field go red and add the words 'email is invalid' at the bottom of the form".
We had a whistle stop tour of jQuery this course. However, if you want to learn more about jQuery you might want to try some of the following resources:

- The Codecademy jQuery Course
- The jQuery Learning Center
- The CodeSchool jQuery Course
- W3 Schools Tutorial