Girls coding course programme

Introduction to web development

Course Notes for students

2019

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 accompany the sessions. 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 is coding?
- Ch.2 - How does the Internet work?

Part 1: HTML

- Ch.3 -Hello World
- Ch.4 - HTML Basics

Part 2: CSS

- Ch.5 - What is CSS?
- Ch.6 - Linking your HTML & CSS code
- Ch.7 - Writing some CSS & basic definitions
- Ch.8 - Selectors & Attributes
- Ch.9 - Using the ID & class selectors
- Ch.10 - Finishing up and getting fancy

Part 3: Introduction to UX

- Ch.11 - User experience
- Ch. 12 - User experience and analytics

Part 4: Course competition

- Ch.13 - Competition arrangements and criteria

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, Bootstrap, jQuery & JavaScript (don't worry you'll learn about these last three topics later!)
- 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 part will include a number of tasks to do 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
- 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, 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	Atom	

	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.		
4. Install GitHub Desktop Client	A GUI for you to start using GitHub to collaborate on projects, without having to touch your command line.	GitHub Desktop Client	

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 prep**: (*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".

4. In your documents, create a folder called 'coding_course'. This will be where you will keep all the websites you will be making over the next few sessions.

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 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, 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.

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 😊)

There are no secrets in HTML, CSS or js

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. 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

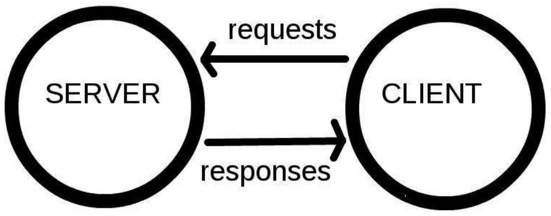
1. Open any website in Google Chrome, or the www.codefirstgirls.org.uk website
2. View the page source by doing one of the following:
   a. View > Developer > View Source
   b. Tools > View Source
3. Open the developer tools by doing one of the following:
   a. View > Developer > Developer Tools
   b. 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

Getting going & HTML

Chapter 1 - What even is the internet and the world wide web?

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**.

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.*

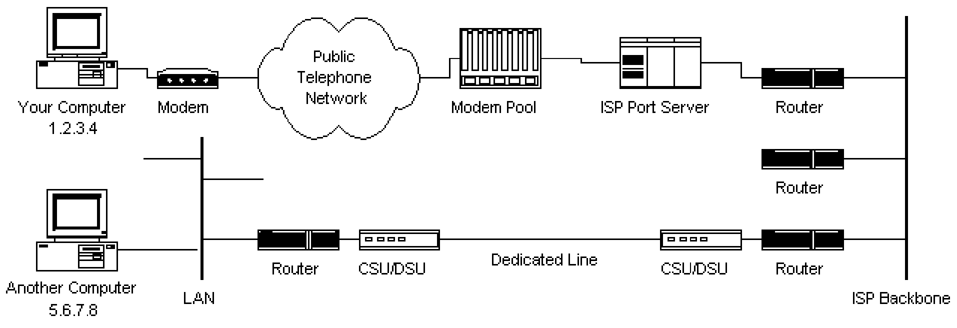
The computers that allow people to access the web by providing software (browsers) for them to do so are called **clients***.* They are the typical user's Internet-connected device, for example - your laptop or mobile phone, connected to the internet through a browser.

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.



How the Web Works

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	WordPress
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 2 - 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.

Part 1 - HTML

Chapter 3 - Hello World

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.

Demo: play around with this codepen to experiment with HTML tags

In it write:

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

Chapter 4 - 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. For example

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

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">
```

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

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
```

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

Task: start your first website

1. Go to the github repository for this session: https://github.com/CodeFirstGirls/beginners-part-one
2. Follow the instructions on the README file and carry out the tasks
3. Make sure you finish all the tasks, as we will be using this file as a basis of your personal website over the next few sessions!

Extra resources

W3 Schools HTML Tutorial

HTML Terms Glossary

Simple HTML Guide Cheatsheet

A HTML Validator that checks your HTML code

Part 2 - CSS

Chapter 5 - 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.

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 6 - 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**

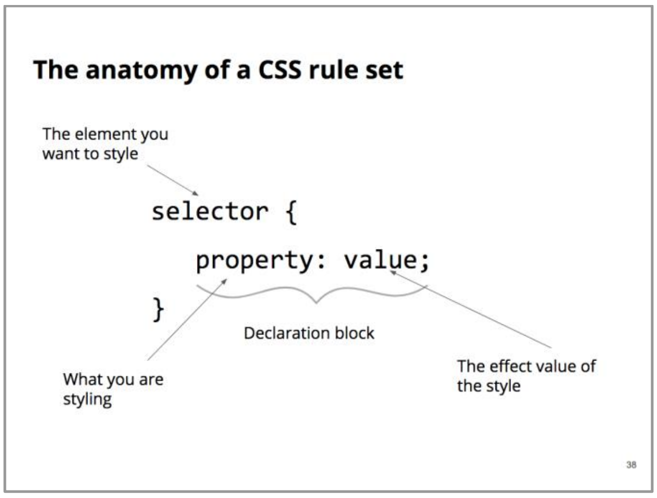
Relative links are the most flexible - they will work on your local file system. The only thing you have to be careful about is moving your files into different folders, which can cause links to break.

You should be using relative local links in these exercises.

1. **Task:**
   Open the exercise you completed in the last session in atom and Chrome. (if you haven't completed it, you can copy the solution from here). Make sure you open the whole folder in atom, not just the 'index.html' file.
2. In the exercise folder, create a new folder called 'css'
3. Inside this, create a new file called 'styles.css'

The rest of the session is a code-along, at the end of the session your code should look the same as in the example on github, found here. If, for any reason, you miss the session, copy the completed code from the example on github and follow the course script, included below for reference.

Chapter 7 - Writing CSS & basic definitions


The anatomy of a CSS rule set

```
selector {
   property: value;
}
```

The anatomy of a CSS rule set

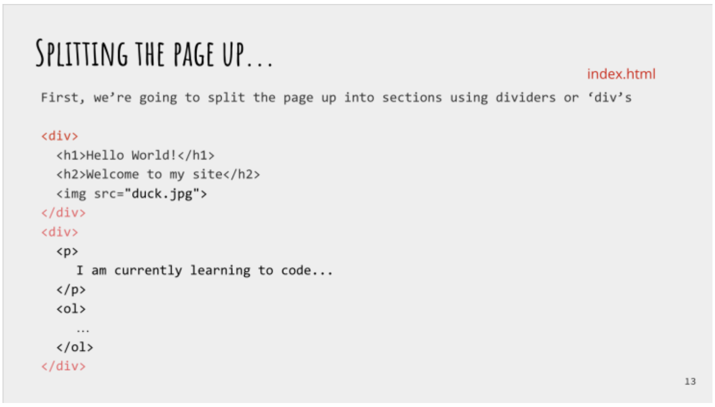
Ok, before we get carried away, let's have a look at the anatomy of a CSS rule. The CSS we just added is called a rule set:

```
selector {
      property: value;
}
```

Each rule set is made up like this:

- The **selector** is the thing you want to style.
- After the selector comes an opening curly bracket {
- Inside the curly brackets you add a declaration block each style change you want to add to the selector.
- A declaration block is made up of a property and value separated by a : and ended with a ;
- The **property** is what you want to change. The **value** is what you are changing the property into.
- At the end you close the rule set with a closing curly bracket }

Don't forget the semi colons!! It's such a 'gotcha'



Splitting the page up...

index.html

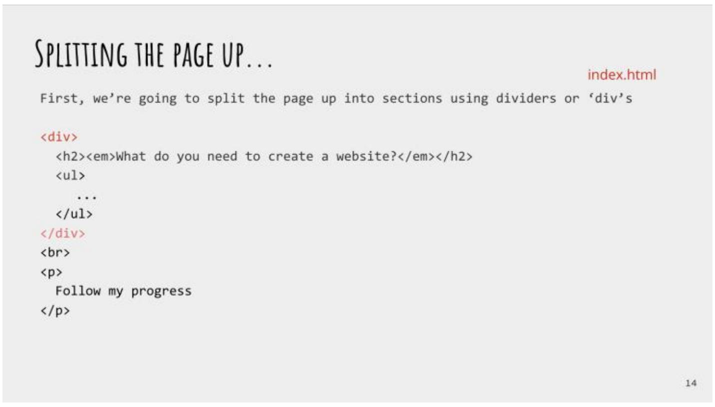
First, we're going to split the page up into sections using dividers or 'div's

```
<div>
  <h1>Hello World!</h1>
  <h2>Welcome to my site</h2>
  <img src="duck.jpg">
</div>
<div>
  <p>
    I am currently learning to code...
  </p>
  <ol>
    ...
  </ol>
</div>
```

13

Splitting the page up - Adding divs

'Div' stands for 'divider' and we use them to literally divide the page up into parts. This will make it easier to style. A divider is a 'block level' element. This means that it forces the next html element onto a new line



Splitting the page up...

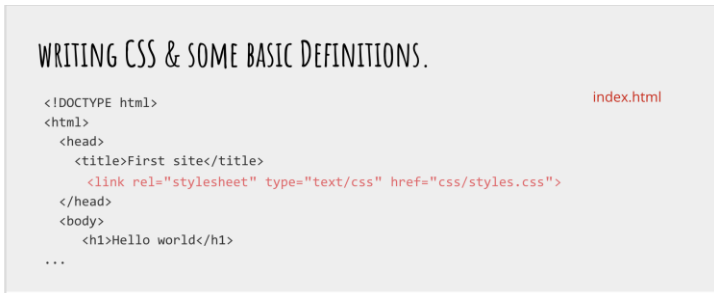
index.html

First, we're going to split the page up into sections using dividers or 'div's

```
<div>
  <h2><em>What do you need to create a website?</em></h2>
  <ul>
    ...
  </ul>
</div>
<br>
<p>
  Follow my progress
</p>
```

14

Now we have split the page up into three different parts (plus the last paragraph) which we can now easily style differently!



WRITING CSS & SOME BASIC DEFINITIONS.

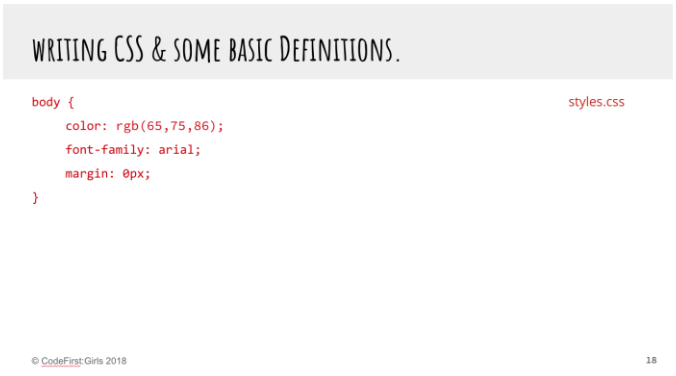
index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>First site</title>
      <link rel="stylesheet" type="text/css" href="css/styles.css">
  </head>
  <body>
     <h1>Hello world</h1>
...
```

Linking the stylesheet and first css definition

First things first before we can add all our styles, is we have to link the stylesheet to our index.html! As we have already discussed, we need to do this by adding a link in the head of the index.html.

Now let's add our first CSS rule!



WRITING CSS & SOME BASIC DEFINITIONS.

styles.css

```
body {
    color: rgb(65,75,86);
    font-family: arial;
    margin: 0px;
}
```

© CodeFirst:Girls 2018 18

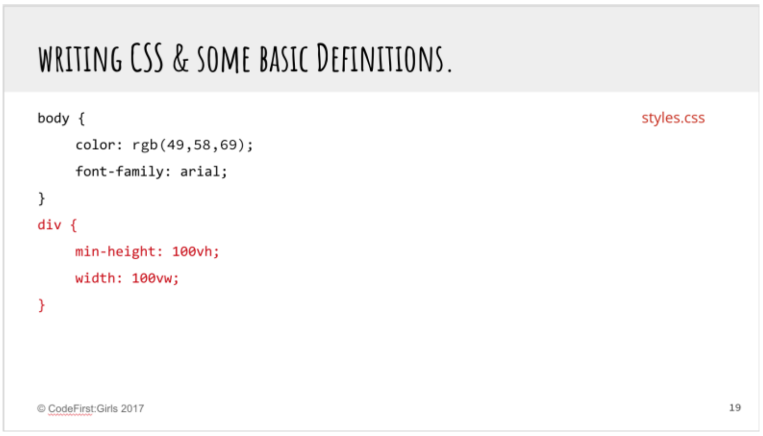
So here we have changed three different things on the body: color, font family and margin. Let's break these down.

Color changes the colour of the text on the page - and it's spelt the american way unfortunately. (watch out for this as it's a common gotcha). We are going to use RGB values which stands for Red Green Blue. All colours in computers are composed of different amounts of red, green and blue from 0 to 255. So If I wanted purely red I would put in `rgb(255,0,0)`. It may be a bit confusing at first but luckily, In the developer tools, chrome has a really handy tool to select colour, and it will give you RGB values for the exact colour you want to select.

Font-family changes (you guess it) the font for the selected element. Handily, this can also hold several font names as a “fallback” system, if the browser does not support the first.

Now, let’s talk about **Margin**. This is one of the primary positioning css rules which you can use to space elements out. Margin adds space around the outside of the element, so it pushes other elements away.

The browser adds a margin by default, and by setting it to 0px, we are removing it. ‘Px’ stands for pixel, which is a fixed length. (1 dot of pixel of the viewing device). This is the most commonly used css measurement value, but we will be covering others in the following slides.



WRITING CSS & SOME BASIC DEFINITIONS.

styles.css

```
body {
    color: rgb(49,58,69);
    font-family: arial;
}
div {
    min-height: 100vh;
    width: 100vw;
}
```

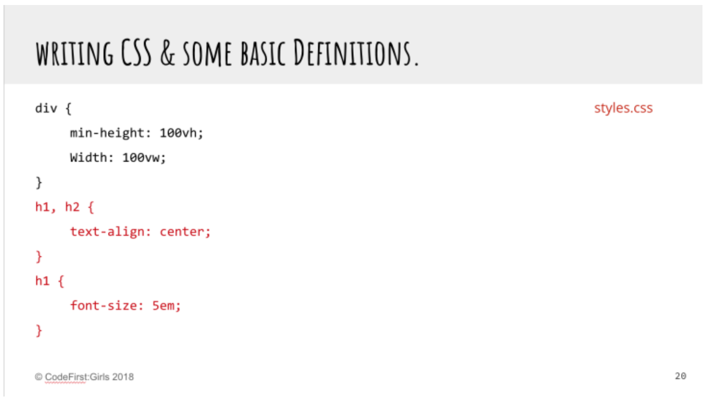
© CodeFirst:Girls 2017 19

Now we've added some styles to the whole body, lets select the divs.
Again here, we are selecting the three divs we added to the HTML earlier, so they will al be styles the same.

Min-height specifies the minimum height the element can be. This is important for responsible sites as they now should get larger and bigger according to the different size of screen. However, by specifying min height, it will never get below this height no matter how small.

Similarly, the width is defining the width of these divs.
So, what is this '**vh**' and '**vw**'? These however stand for 'viewport height' and 'viewport with'. 100 viewport width means that the element needs to take up 100% of the viewport width. This is a really useful little rule!

There are several different size systems that you can use in css. The one you are probably most familiar with is **pixel**, which is a standard length. Then **viewport width/height** (there is also **window height and width**) and then the **%**. That will be the **%** width of height of the parent element.



WRITING CSS & SOME BASIC DEFINITIONS.

styles.css

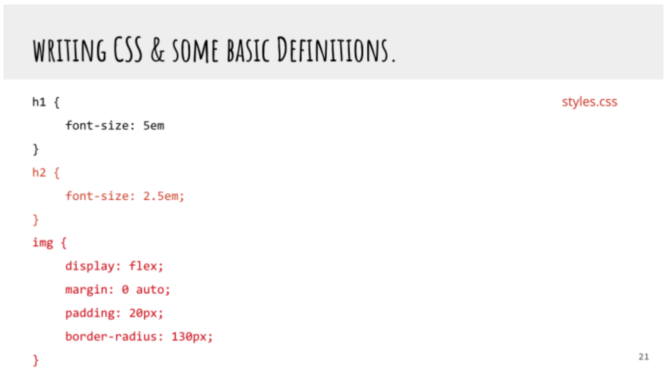
```
div {
     min-height: 100vh;
     Width: 100vw;
}
h1, h2 {
     text-align: center;
}
h1 {
     font-size: 5em;
}
```

© CodeFirst:Girls 2018

20

Now lets target some other things!
Here we are centering the text of the h1 and h2 elements to the center, and then making the font size of the h1.

Here we have introduced another unit, '**em**', just to jazz things up! Em means it is relative to the font size of the element, so **5em** is 5 times the relative font size of the element. Again, this is a useful property for responsive design.



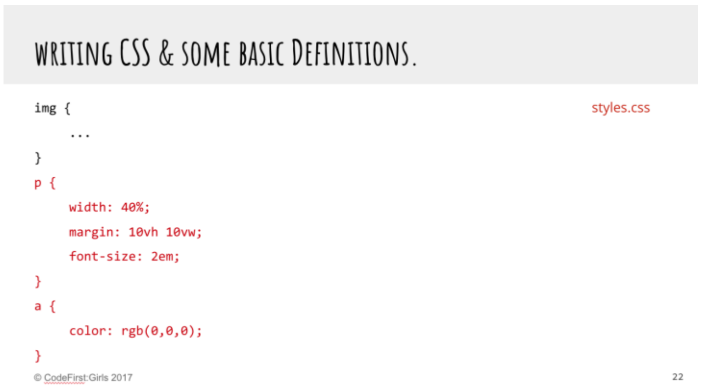
WRITING CSS & SOME BASIC DEFINITIONS.

styles.css

```
h1 {
     font-size: 5em
}
h2 {
     font-size: 2.5em;
}
img {
     display: flex;
     margin: 0 auto;
     padding: 20px;
     border-radius: 130px;
}
```

21

Now we are targeting the h2, to make it smaller, and the image.
'Flex' is a very useful display setting which allows for 'flexible placing' around the window. Padding here is similar to the margin property, but instead of adding space around the outside of the element, it adds space inside of the element. It sets the padding area on all four sides of an element. It is a shorthand for padding-top, padding-right, padding-bottom, and padding-left. Border radius adds corners to the element, so here we are making it a circle by cutting off the corners!



WRITING CSS & SOME BASIC DEFINITIONS.

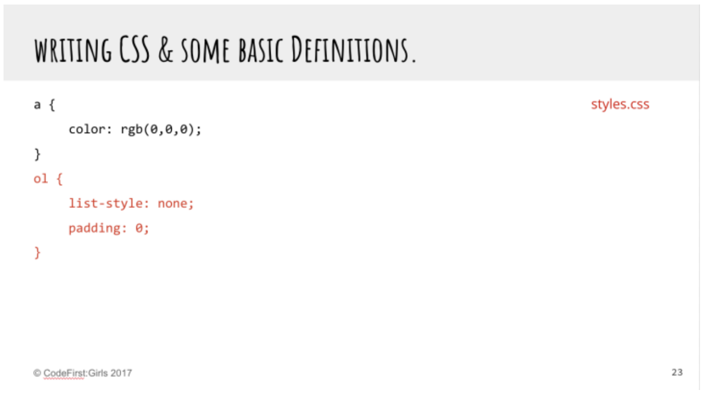
styles.css

```
img {
    ...
}
p {
    width: 40%;
    margin: 10vh 10vw;
    font-size: 2em;
}
a {
    color: rgb(0,0,0);
}
```

© CodeFirst:Girls 2017 22

Now we are styling all the paragraphs. We only want them to be 40% of the width of their parent elements, and a responsive margin.

Let's also change the colour of the links (anchor tags) to make them look nicer than just blue!



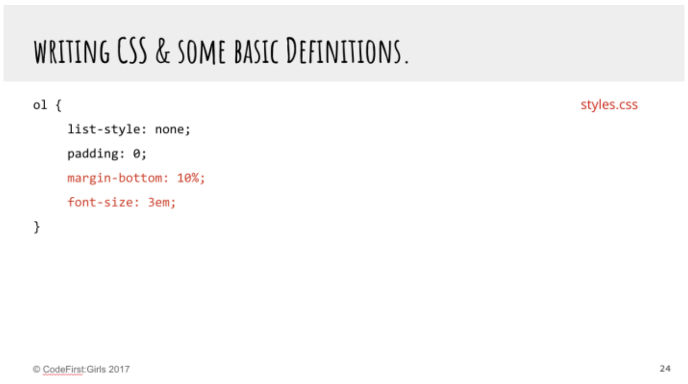
WRITING CSS & SOME BASIC DEFINITIONS.

styles.css

```
a {
    color: rgb(0,0,0);
}
ol {
    list-style: none;
    padding: 0;
}
```

© CodeFirst:Girls 2017 23

Lists are very useful things, but almost never do we use their default styling, as it's a bit ugly. So let's just remove that like so!



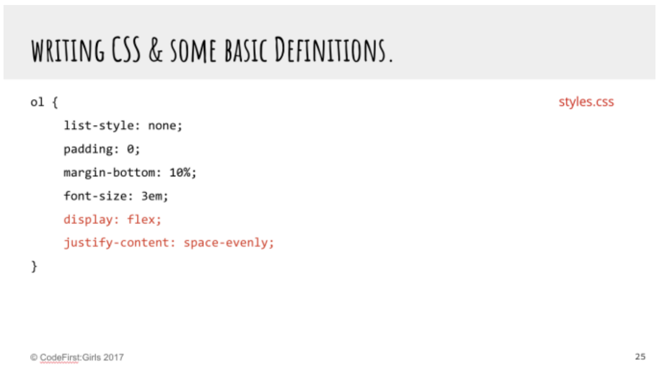
WRITING CSS & SOME BASIC DEFINITIONS.

styles.css

```
ol {
    list-style: none;
    padding: 0;
    margin-bottom: 10%;
    font-size: 3em;
}
```

© CodeFirst:Girls 2017 24

Let's also add a wee bit of margin on the bottom and pump the font size up, to make it slightly more in your face.



WRITING CSS & SOME BASIC DEFINITIONS.

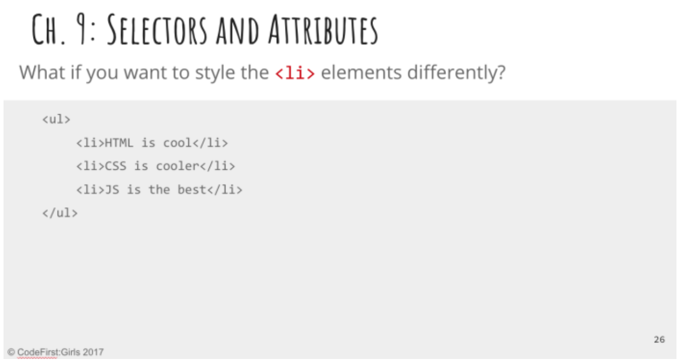
styles.css

```
ol {
    list-style: none;
    padding: 0;
    margin-bottom: 10%;
    font-size: 3em;
    display: flex;
    justify-content: space-evenly;
}
```

© CodeFirst:Girls 2017 25

And here we can use Mr Flexbox again. Again this is a really useful way to getting elements into the right places. Justify-content defines how the browser distributes space between and around content items along the main-axis of a flex container, and the inline axis of a grid container.

Chapter 8 - Selectors and Attributes



Ch. 9: Selectors and Attributes

What if you want to style the `<li>` elements differently?

```
<ul>
    <li>HTML is cool</li>
    <li>CSS is cooler</li>
    <li>JS is the best</li>
</ul>
```

© CodeFirst:Girls 2017

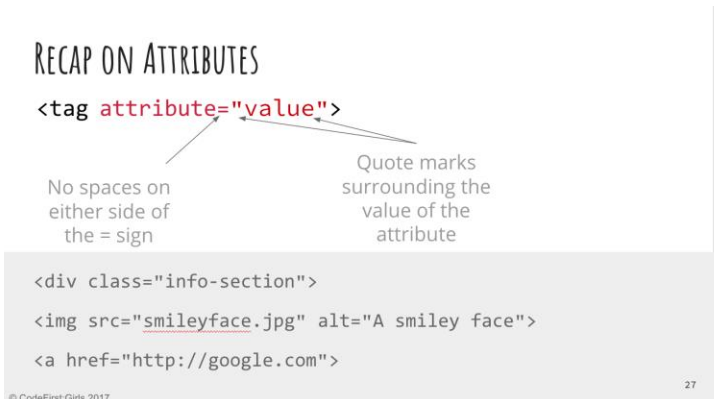
26

So that's the basic styles out of the way, now let's get a little more complex. What is we want to style the individual 'li' elements differently?

We now know all about what selectors are in CSS. So far we have used the names of HTML tag names as selectors. We call these element type selectors.

Even though you can string up several tag names divided by spaces to reach a deeply nested HTML element with CSS, sometimes you may need to be more specific.

For example imagine if you have a webpage with two **`<h2>`** elements and you want each one to look different. Simply using element type selectors will not be possible. To achieve this you can use specific HTML attributes.


Recap on Attributes

```
<tag attribute="value">
```

No spaces on either side of the = sign
Quote marks surrounding the value of the attribute

```
<div class="info-section">
<img src="smileyface.jpg" alt="A smiley face">
<a href="http://google.com">
```

27

© CodeFirst:Girls 2017

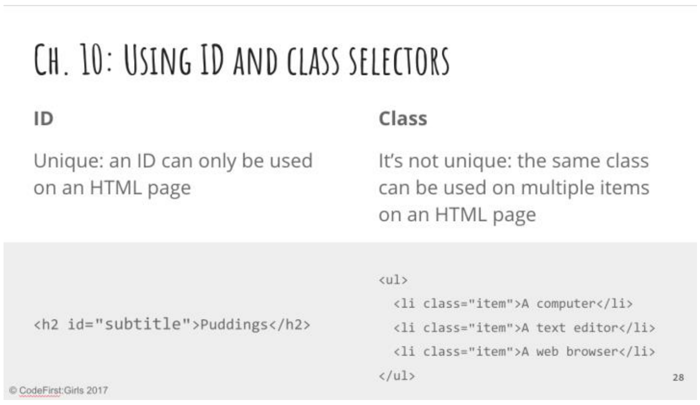
So what is an attribute?

We learned about attributes in HTML.

Attributes are always attached to the opening tag of an HTML element or inside an all-in-one element. They provide additional information about the HTML element.

An HTML attribute is made up of an attribute name and a value. Both are separated by an equal sign and the value is enclosed in double quotes.

Chapter 9 - Using the ID and class selectors



CH. 10: USING ID AND CLASS SELECTORS

ID	Class
Unique: an ID can only be used on an HTML page	It's not unique: the same class can be used on multiple items on an HTML page

```
<h2 id="subtitle">Puddings</h2>
```

```
<ul>
  <li class="item">A computer</li>
  <li class="item">A text editor</li>
  <li class="item">A web browser</li>
</ul>
```

© CodeFirst:Girls 2017

28

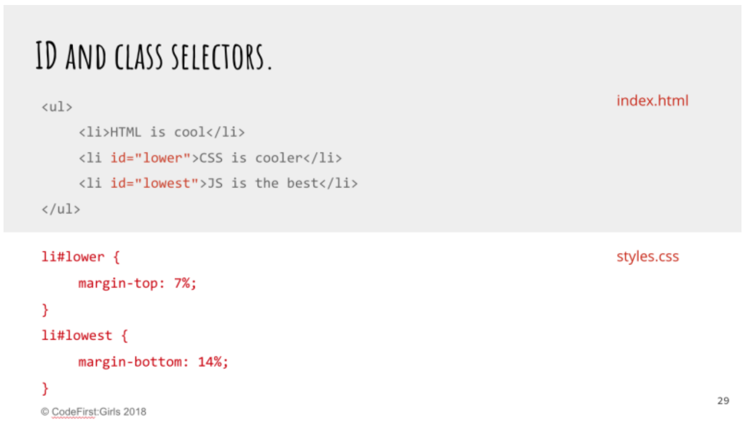
Using the ID and class selectors

There are two attributes you can add to any HTML element. And they are the **id** and **class** attributes. Both are used to add information to the HTML element to which they are added. Both can be used by CSS and JavaScript to target that element for styling or scripting purposes.

The key difference between ID and class is that an ID is unique and can only be added to one HTML element on a page. The same class can be added to multiple elements on a page.

A little analogy: If the world was one bit web page then every person would be an ID, because there is only one of each of us and we are all unique. But our gender would be a class, because there are lots of women and men.

Using the ID attribute on a HTML element will allow you to write CSS that will only affect that particular element. But equally adding the same class name to multiple HTML elements allows you to write the same styles only once for all the elements with that same class name.


ID AND CLASS SELECTORS.

index.html

```
<ul>
    <li>HTML is cool</li>
    <li id="lower">CSS is cooler</li>
    <li id="lowest">JS is the best</li>
</ul>
```

styles.css

```
li#lower {
    margin-top: 7%;
}
li#lowest {
    margin-bottom: 14%;
}
```

© CodeFirst:Girls 2018

29

Using ID selectors in CSS

In your CSS you need to target your ID and class attributes in a special way.

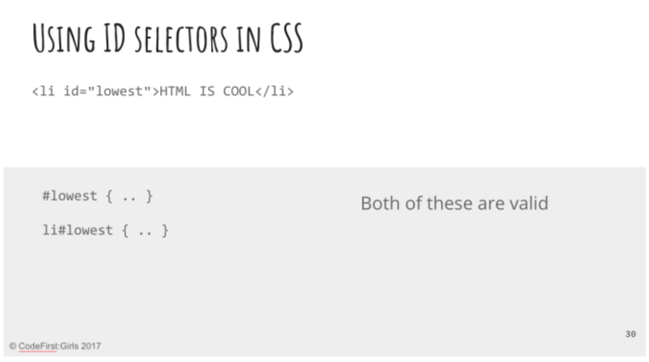
To include an **ID** in your CSS you first write a hashtag # and then the name of the ID.

```
#lower { }
```

You can also include the HTML tag name before the # if you want. In most cases you don't need to do this, but sometimes it can help with being extremely specific.

```
li#lowest{}
```

If you add the HTML tag name to the ID selector make sure there is no space between the tag name and the hash tag. If you leave a space a browser will look for an element with the ID name nested inside an HTML with that tag name.


USING ID SELECTORS IN CSS

```
<li id="lowest">HTML IS COOL</li>
```

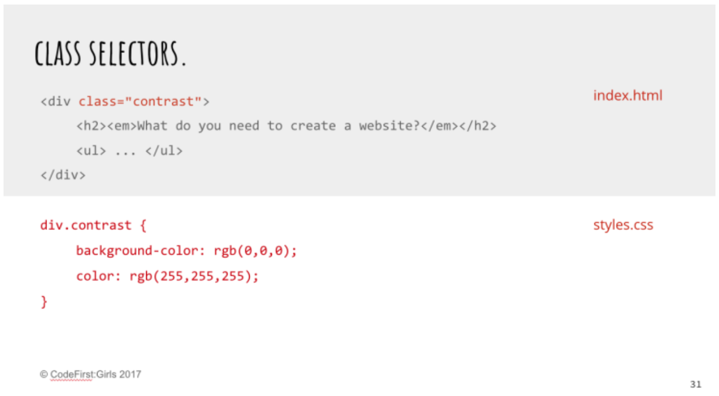
```
#lowest { .. }
li#lowest { .. }
```

Both of these are valid

© CodeFirst:Girls 2017

30

Both of these are valid, but the browser will always give preference to the most specific one.



CLASS SELECTORS.

index.html

```
<div class="contrast">
    <h2><em>What do you need to create a website?</em></h2>
    <ul> ... </ul>
</div>
```

styles.css

```
div.contrast {
    background-color: rgb(0,0,0);
    color: rgb(255,255,255);
}
```

© CodeFirst:Girls 2017

31

Using class selectors in CSS

To include a **class** name in your CSS you first write a full stop . and then the name of your class.

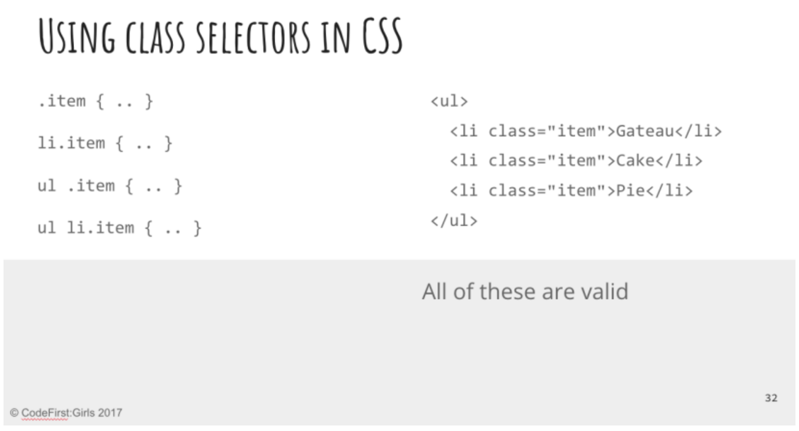
```
.product_item {}
```

Just like the ID you can include the HTML tag name in front of the class name. But again make sure you don't leave a space.

```
li.product_item {}
```

Adding the tag name creates a more specific selector. If you are using a class name on several HTML elements but not all the elements have the same tag name, then adding the tag name to the class will only target the HTML elements with that tag name.

It's up to you how specific you want to make your CSS selectors. Element type selectors using only single HTML tag names can be useful for styling things globally.



Using class selectors in CSS

```
.item { .. }

li.item { .. }

ul .item { .. }

ul li.item { .. }
```

```
<ul>
  <li class="item">Gateau</li>
  <li class="item">Cake</li>
  <li class="item">Pie</li>
</ul>
```

All of these are valid

© CodeFirst:Girls 2017

32

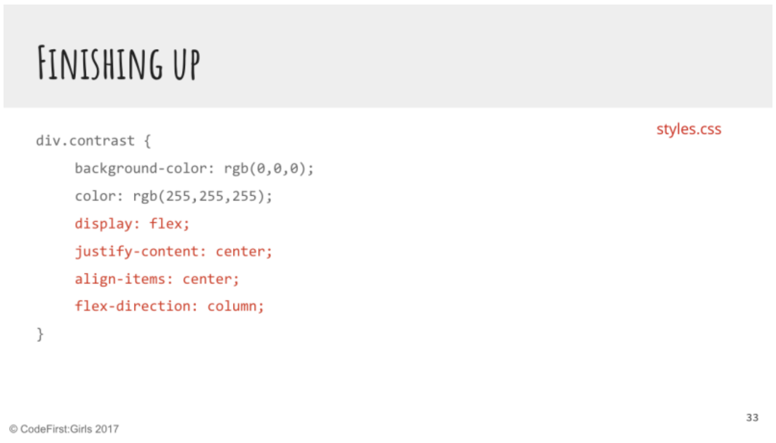
Specificity of selectors

This is a slide to demo css specificity, and is not part of the code along

Similar to the slide with the ID, these are all valid, however the bottom ones are more specific.

Specificity is a weight that is applied to a given CSS declaration, determined by the number of each selector type in the matching selector. When multiple declarations have equal specificity, the last declaration found in the CSS is applied to the element. Specificity only applies when the same element is targeted by multiple declarations. As per CSS rules, directly targeted elements will always take precedence over rules which an element inherits from its ancestor.

Chapter 10 - Finishing Up



FINISHING UP

styles.css

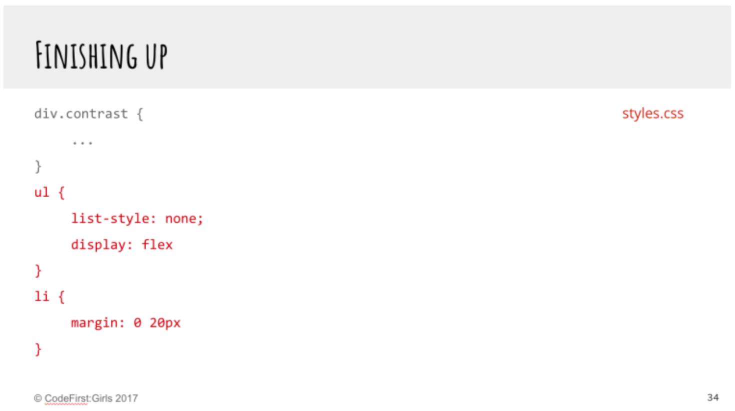
```
div.contrast {
    background-color: rgb(0,0,0);
    color: rgb(255,255,255);
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
}
```

© CodeFirst:Girls 2017

33

To finish up, let's just add a few more things on to get to the finished design.

Let's make the contrast divs a little bit different, by getting all the child elements into columns .



FINISHING UP

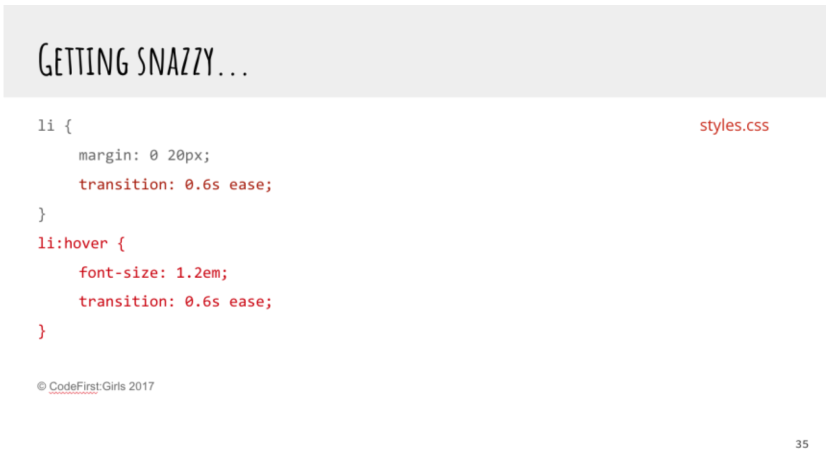
styles.css

```
div.contrast {
    ...
}
ul {
    list-style: none;
    display: flex
}
li {
    margin: 0 20px
}
```

© CodeFirst:Girls 2017

34

Now we can style the unordered list in a different way, and separate out each list element.



GETTING SNAZZY…

styles.css

```
li {
    margin: 0 20px;
    transition: 0.6s ease;
}
li:hover {
    font-size: 1.2em;
    transition: 0.6s ease;
}
```

© CodeFirst:Girls 2017

35

Now let's add some snazzy stuff. This is just and introduction, and please play around with different transitions at home.

The :hover selector specifies that it should apply the style only when the mouse is hovering over it.

CSS transitions allows you to change property values smoothly (from one value to another), over a given duration. Thus, we can make the font larger when we hover over each list element, but include a transition so that it changes from the original to the new font size slowly.

Finishing off and extra tasks:

Quite simple for today, just finish off the code along. Work through the slides at your own pace or check out the solution on github.

Also on the github repo there are several other designs, completely different to the one we did today, that use all the same HTML, but different css. Have a look at them to see how powerful css can be.

Also, have a look at flexbox froggy, it's a very good tool to use as a reference for all things flexbox.

Part 3 - User Experience (UX)

Chapter 11 - 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."*

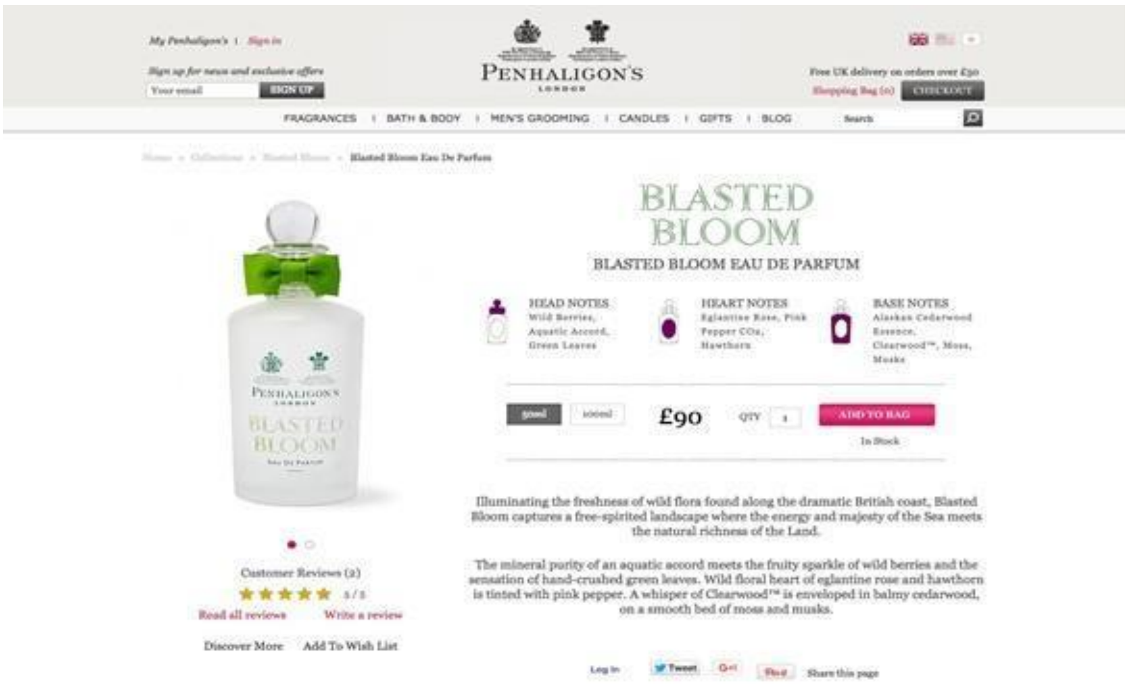
*"There's nothing more **frustrating** when you do a password and it tells you it's wrong, but it doesn't tell you what aspect is wrong"*

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

Chapter 12 - 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.



BLASTED BLOOM

BLASTED BLOOM EAU DE PARFUM

HEAD NOTES Wild Berries, Aquatic Accord, Green Leaves

HEART NOTES Eglantine Rose, Pink Pepper CO2, Hawthorn

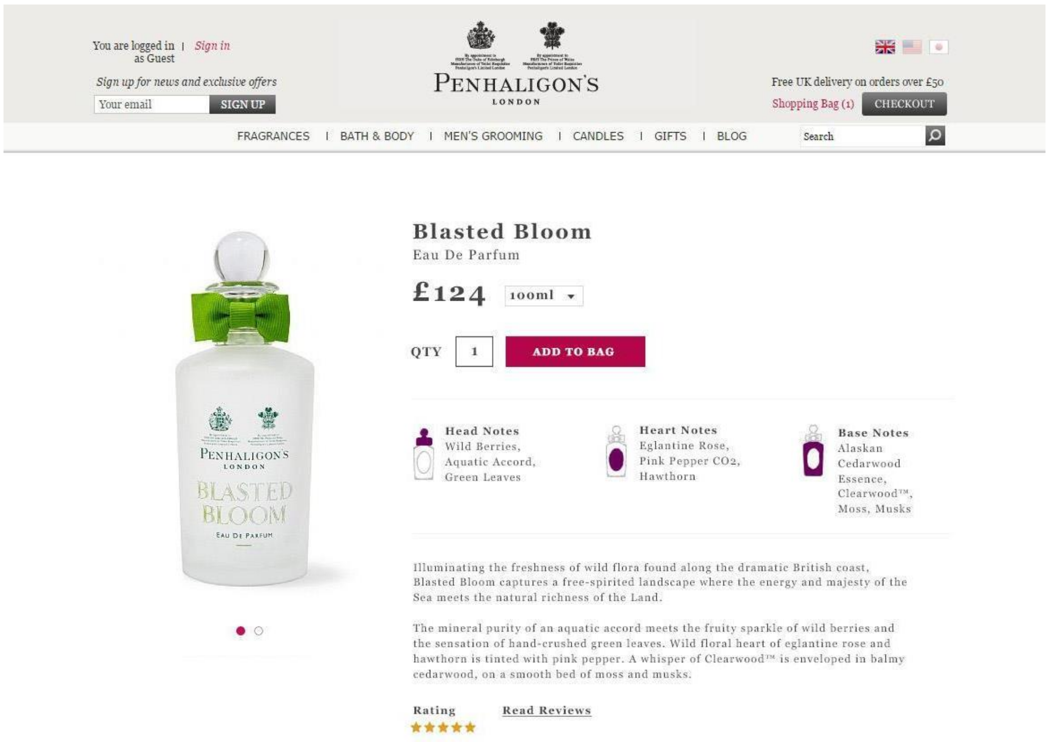
BASE NOTES Alaskan Cedarwood Essence, Clearwood™, Moss, Musks

£90

Illuminating the freshness of wild flora found along the dramatic British coast, Blasted Bloom captures a free-spirited landscape where the energy and majesty of the Sea meets the natural richness of the Land.

The mineral purity of an aquatic accord meets the fruity sparkle of wild berries and the sensation of hand-crushed green leaves. Wild floral heart of eglantine rose and hawthorn is tinted with pink pepper. A whisper of Clearwood™ is enveloped in balmy cedarwood, on a smooth bed of moss and musks.

Control version - original project page design


You are logged in as Guest | Sign in
Sign up for news and exclusive offers
Your email
SIGN UP
PENHALIGON'S
LONDON
Free UK delivery on orders over £50
Shopping Bag (1)
CHECKOUT
FRAGRANCES | BATH & BODY | MEN'S GROOMING | CANDLES | GIFTS | BLOG
Search
100ml
QTY
1
ADD TO BAG
Rating
Read Reviews

Blasted Bloom

Eau De Parfum

£124

Head Notes Wild Berries, Aquatic Accord, Green Leaves

Heart Notes Eglantine Rose, Pink Pepper CO2, Hawthorn

Base Notes Alaskan Cedarwood Essence, Clearwood™, Moss, Musks

Illuminating the freshness of wild flora found along the dramatic British coast, Blasted Bloom captures a free-spirited landscape where the energy and majesty of the Sea meets the natural richness of the Land.

The mineral purity of an aquatic accord meets the fruity sparkle of wild berries and the sensation of hand-crushed green leaves. Wild floral heart of eglantine rose and hawthorn is tinted with pink pepper. A whisper of Clearwood™ is enveloped in balmy cedarwood, on a smooth bed of moss and musks.

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

Part 4 - Course competition

Chapter 13 - 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)
    - Code 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, 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. 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 can create teams using our Group Generator!

Part 5 - GitHub , Version Control, Git

Chapter 14 - 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.

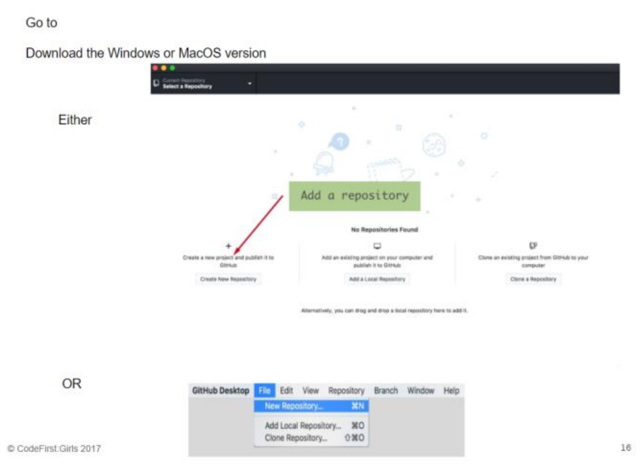
The below are some of the main elements of version control, check through them and talk them through with your partner or instructor if you're unclear:

- 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 15 - Creating your first repository

Task: Creating a GitHub repository


Go to

Download the Windows or MacOS version

Either

Add a repository

No Repositories Found

Create a new project and publish it to GitHub

Add an existing project on your computer and publish it to GitHub

Clone an existing project from GitHub to your computer

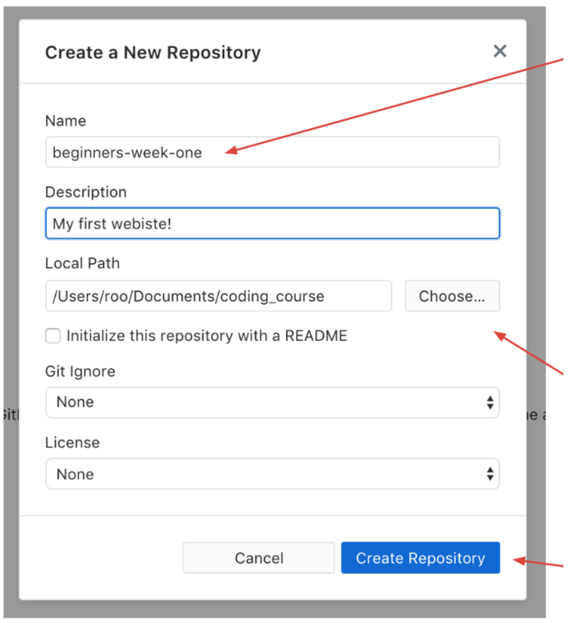
Alternatively, you can drag and drop a local repository here to add it.

OR

© CodeFirst:Girls 2017

16

1. Open up the GitHub Desktop client on your computer
2. You should see something like this
3. Click on the **plus sign** in the top left hand corner of the screen


Create a New Repository

Name: beginners-week-one

Description: My first webiste!

Local Path: /Users/roo/Documents/coding_course

☐ Initialize this repository with a README

Git Ignore: None

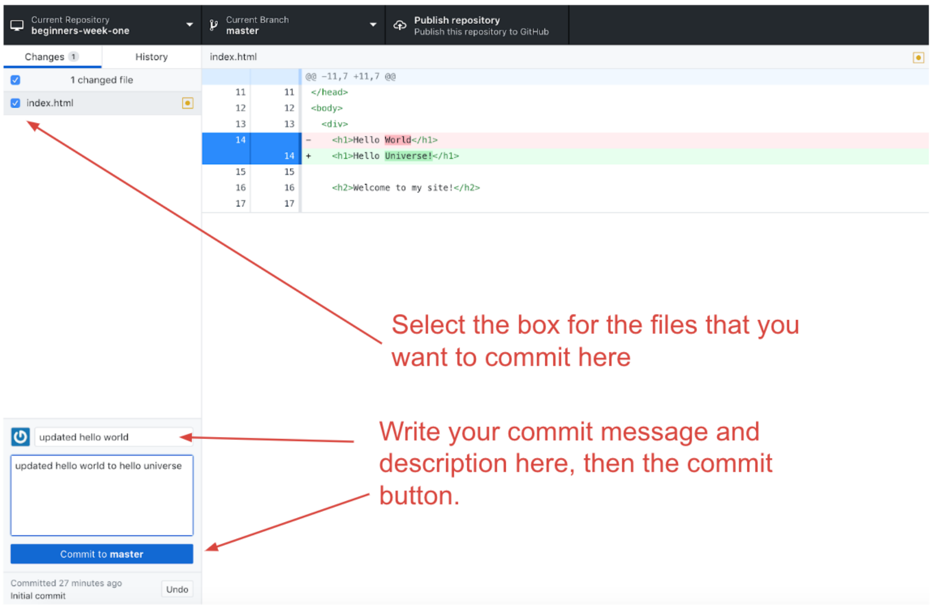
License: None

1. On the **Create** tab of the pop-up window type the name of your repository. In this case, because we are initialising a folder that is already there, we need to write in the folder name **exactly** as it is.
2. Check in the **Local Path** box that it has the correct path name for your Part 1 project.
3. Click the **Create Repository** button.
4. The GitHub Desktop client will now create your folder at the location you wanted and, turn it into a git repository.

You now have a repository! You can look at all the changes you have made in the 'changes' tab, and you can see the history of your changes in the 'history' tab.

Task: First commit

1. Make a small change in your index.html file, this can be as small as changing 'hello world' to 'hello universe'. You should now be able to see these in the desktop.
2. Tick the check box for the files that you want to commit (here we only have one line change in one file) but you can have multiple
3. Then write a commit message and a description for the commit (if you want) and press the commit button



```
@@ -11,7 +11,7 @@
  </head>
  <body>
    <div>
-     <h1>Hello World</h1>
+     <h1>Hello Universe!</h1>

      <h2>Welcome to my site!</h2>

```

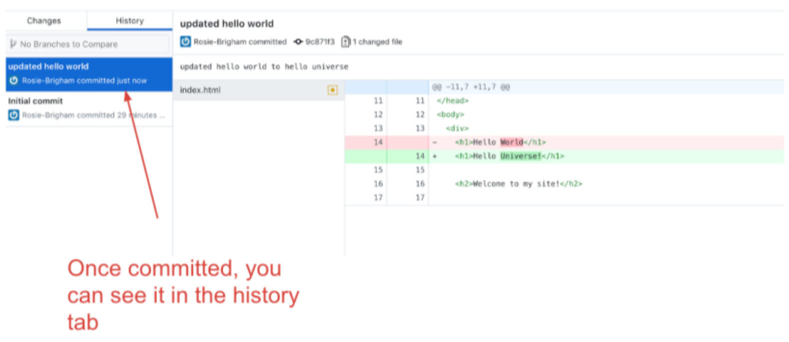
Commit summary: updated hello world

Description: updated hello world to hello universe

Committed 27 minutes ago
Initial commit

Current Repository
beginners-week-one
Current Branch
master
Publish repository
Publish this repository to GitHub
Changes 1
History
1 changed file
index.html
index.html
Commit to **master**
Undo
Select the box for the files that you want to commit here
Write your commit message and description here, then the commit button.

Make you commit messages descriptive, as when you look at the history of commits it will help you understand what you did. (‘added a thing’ is never helpful a month on when trying to find out where a bug may have been introduced!) Some developers also recommend to write them in the present tense. This makes it easier to read when you go through the history of a project later.



updated hello world
Rosie-Brigham committed just now

Initial commit
Rosie-Brigham committed 29 minutes ...

updated hello world
Rosie-Brigham committed 9c871f3 1 changed file

updated hello world to hello universe

index.html

```
@@ -11,7 +11,7 @@
 </head>
 <body>
   <div>
-    <h1>Hello World</h1>
+    <h1>Hello Universe!</h1>

     <h2>Welcome to my site!</h2>
```

You will then be able to see the commit in the ‘history’ tab!

Undoing a commit

One of the benefits of using a version control system is that you can go back to a previous version of your project if you need to. This can be because of a bug that needs fixing, or maybe you made a mess of a file when trying something clever, or maybe the client you are working for decides that all the work you did in the past month is not what he wanted after all.

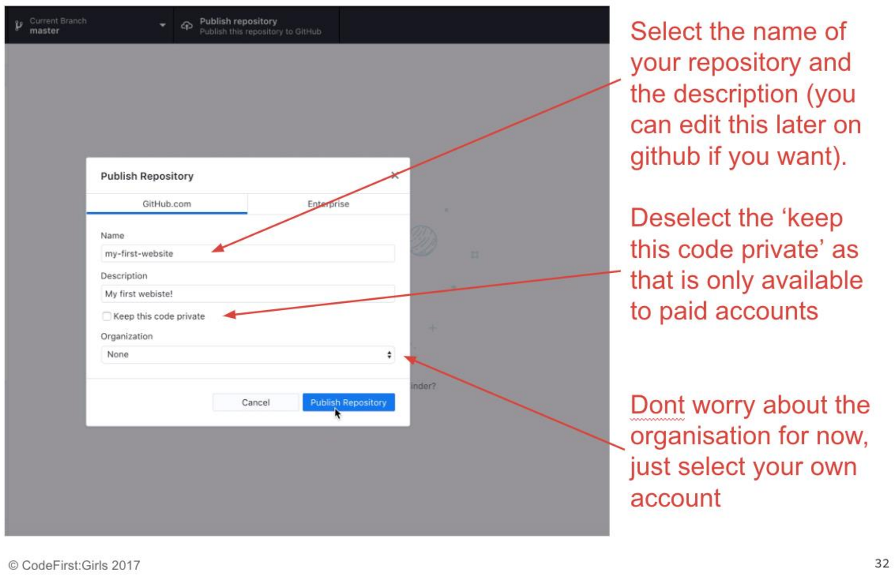
Whatever the reason, you’ll be glad to be using version control when you are in a situation like that. There are two ways of reverting to an older version. Let’s start with the easy way.

If you have just done a commit and realised you didn’t want to do it, or maybe you didn’t include all the files you needed to (or included the wrong ones) then undoing it is simple. Simply press the ‘undo’ button! When you click the Undo button it will put your repository back in the state it was just before you did the commit. Meaning that your changes will still be there, but they are listed as being uncommitted changes. Make whatever you need to do and then commit as usual.

However, if you have already pushed the changes up to github ‘undo’ will get you in a muddle and you will need to ‘revert’ it instead. More on that later...

Chapter 16 - Publishing on Github

What we have done so far is initialising a current project with git on your local machine. We need to publish it to our GitHub account. That's what we'll do next.
Simply click on the **Publish** button in the top right corner of the screen.



Name: my-first-website
Description: My first webiste!
Keep this code private
Organization: None

Select the name of your repository and the description (you can edit this later on github if you want).

Deselect the 'keep this code private' as that is only available to paid accounts

Dont worry about the organisation for now, just select your own account

© CodeFirst:Girls 2017

32

In the pop-up window you should see the name of your repository and your Github account name. If you want to you can write a description for your repository.

Click the **Publish Repository** button. You should now get a message saying it is publishing to GitHub.

1. Go to your account on https://github.com
2. Click on the arrow next to your profile image in the top right corner and select Your profile from the drop down menu.
3. Then click on the Repositories tab and you should see your new repository at the top of the page.

Chapter 17 - Branching and Merging

Up until now we have been using a version control system to commit our changes and to fix commits we should not have committed. But as we saw earlier, using a version control system is also a great way of adding new features in isolation to a project without jeopardising the current code.

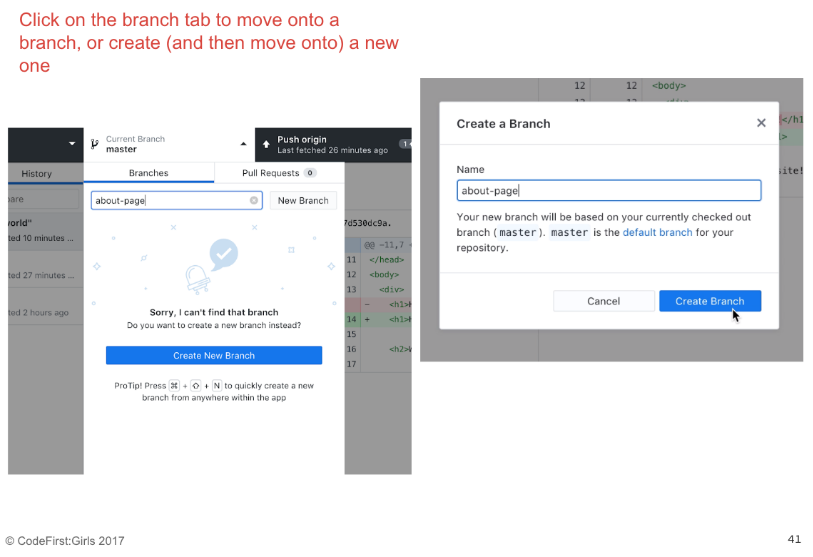
To do this you need to create a separate branch on your project.

Remember when you create a git repository it will also create your first branch with the name master. If you want to add a new feature to your project then you first create a different branch, do all your work, test it out thoroughly and then merge the feature branch back into the master branch.

When you create a new branch, git will take a copy of the current branch you are in, usually the master branch. This new branch will sit alongside the main branch until you are ready to merge it back into the mainstream.

Let's create a branch on our repository.

1. First remember to pull or push any changes up before you add a branch
2. Click on the 'branch' tab, you can type in a branch name to search for it. If there is no branch found, as in this case, you can simply create a new one.


Click on the branch tab to move onto a branch, or create (and then move onto) a new one

Sorry, I can't find that branch
Do you want to create a new branch instead?

ProTip! Press ⌘ + ⇧ + N to quickly create a new branch from anywhere within the app

Create a Branch

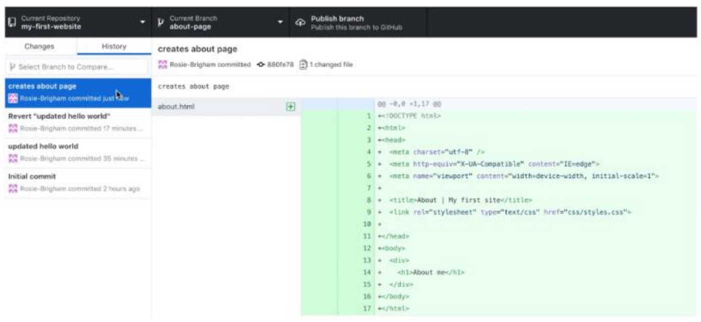
Name: about-page

Your new branch will be based on your currently checked out branch (master). master is the default branch for your repository.

© CodeFirst:Girls 2017 41

Here we have added a new page, about.html.
You can check what branch you are in on the branch tab, and when commiting new changes you will see that the button tells you what branch you are committing to.

Let’s commit this new page onto the about page branch.


Current Repository
my-first-website
Current Branch
about-page
Publish branch
Publish this branch to GitHub
Changes
History
Select Branch to Compare...
creates about page
Rosie-Brigham committed just now
Revert "updated hello world"
Rosie-Brigham committed 17 minutes ...
updated hello world
Rosie-Brigham committed 35 minutes ...
Initial commit
Rosie-Brigham committed 2 hours ago
creates about page
Rosie-Brigham committed
880fe78
1 changed file
creates about page
about.html

```
@@ -0,0 +1,17 @@
+<!DOCTYPE html>
+<html>
+<head>
+  <meta charset="utf-8" />
+  <meta http-equiv="X-UA-Compatible" content="IE=edge">
+  <meta name="viewport" content="width=device-width, initial-scale=1">
+
+  <title>About | My first site</title>
+  <link rel="stylesheet" type="text/css" href="css/styles.css">
+
+</head>
+<body>
+  <div>
+    <h1>About me</h1>
+  </div>
+</body>
+</html>
```

The history on this branch is now different to the master branch. Press the **publish** button to get the new branch, and your changes, up on github.

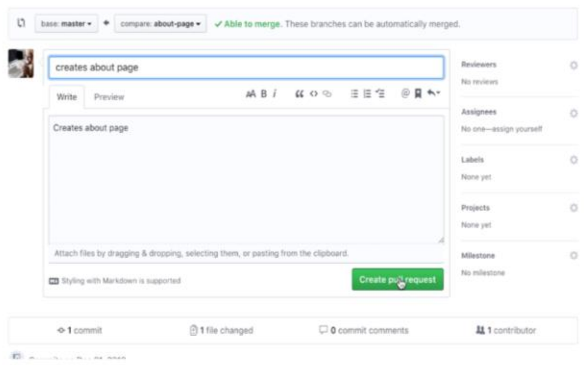
44

Here we can see the commit made onto the about page branch. The history on the about-page branch is now different to that on master. Let’s push the **publish** button to push our changes, and the new branch, onto github.
But how do we get these changes back onto the master branch? We do this using **pull requests.**

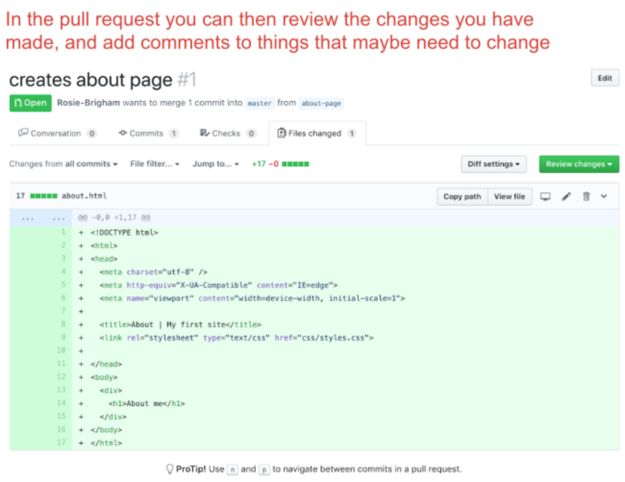
Pull requests

Go to your repo on github and navigate to the ‘branches’ tab.
Here you will be able to see a list of all the branches that you have pushed to github. If any of the branches histories are different to that of master, there will be a ‘new pull request’ button. Click on this for our about page to create a new pull request.

Add a title and description for your pull request to describe the changes you’ve made



Here you will be able to review all the changes you have made, alongside adding a title and description to your pull request. For large complicated ones, these are important so that someone who is reviewing your code will know exactly what changes you are making and why.


In the pull request you can then review the changes you have made, and add comments to things that maybe need to change

creates about page #1

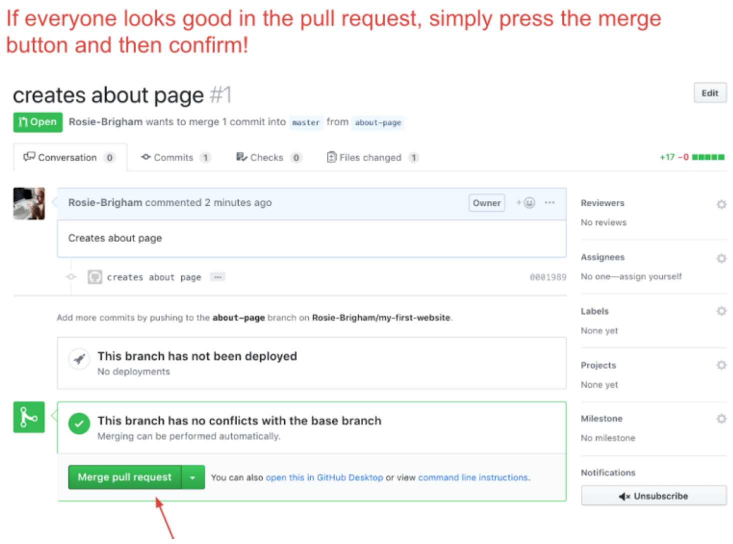
Rosie-Brigham wants to merge 1 commit into `master` from `about-page`

17 about.html

```
@@ -0,0 +1,17 @@
+ <!DOCTYPE html>
+ <html>
+ <head>
+   <meta charset="utf-8" />
+   <meta http-equiv="X-UA-Compatible" content="IE=edge">
+   <meta name="viewport" content="width=device-width, initial-scale=1">
+
+   <title>About | My first site</title>
+   <link rel="stylesheet" type="text/css" href="css/styles.css">
+
+ </head>
+ <body>
+   <div>
+     <h1>About me</h1>
+   </div>
+ </body>
+ </html>
```

ProTip! Use n and p to navigate between commits in a pull request.

Once created, you, or a colleague, can review the changes you have made. At this point, if there are any things you think that needs to be changed before it's ready to be merged in, you can comment and request for changes.


If everyone looks good in the pull request, simply press the merge button and then confirm!

creates about page #1

Rosie-Brigham wants to merge 1 commit into `master` from `about-page`

Rosie-Brigham commented 2 minutes ago

Creates about page

creates about page 0001989

Add more commits by pushing to the **about-page** branch on **Rosie-Brigham/my-first-website**.

This branch has not been deployed
No deployments

This branch has no conflicts with the base branch
Merging can be performed automatically.

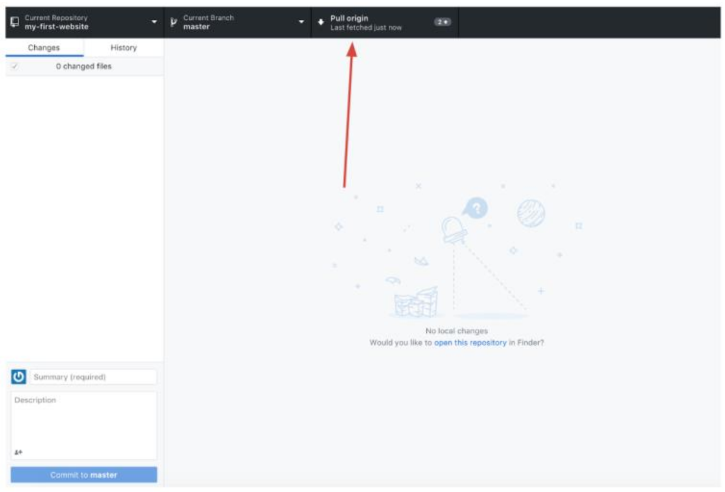
You can also open this in GitHub Desktop or view command line instructions.

Reviewers: No reviews
Assignees: No one—assign yourself
Labels: None yet
Projects: None yet
Milestone: No milestone

After that, it should be all merged in! Make sure that you pull these changes down on your local machine using the 'pull origin' setting.

A little bit of good advice

Whenever you have things on your remote repo that is not on your local one (such as merged branches) it is crucial that you pull them down. You can do this by pressing the 'pull origin' button.



It is really easy to make a right mess of things when working with repositories. In a moment we'll do this deliberately so that you can learn how to resolve conflicts. But the best way of working with git is to avoid conflicts all together.

So the first rule of git is as follows:

Always pull down new changes

Chapter 18 - Publishing on GitHub Pages

Git and GitHub isn't only about version control. Another great feature that GitHub has is free hosting for your website projects. The hosting can only be used for repositories stored on GitHub accounts, and is only suitable for static websites where there is no requirement for a database. Despite that, it's a good way of having a website online at no extra cost.

In order to publish your website through GitHub pages you will need to create a branch in your repository called gh-pages. GitHub will look for this branch and use the contents of this repository to create a website from .

Any code you want to show on your website will need to be added to this branch. If it isn't on this branch you will not see it on your online website. In other words, if you for example added a page to your website on the master branch, and you haven't merged your master branch into your gh-pages branch, then you online website will not have this extra page.

Once you have published your website to GitHub Pages your site will show up at **[your-github-username].github.io/[repository-name]**

On the settings page of your repo on github, select the 'settings' tab and scroll down to the 'github pages' section and select 'enable'. Select the master branch to be the branch that it is built from.

Click on that link and you should now have your website up and running, you will need to commit and push to it first so that it can build.

Chapter 19 - Conflict scenario!

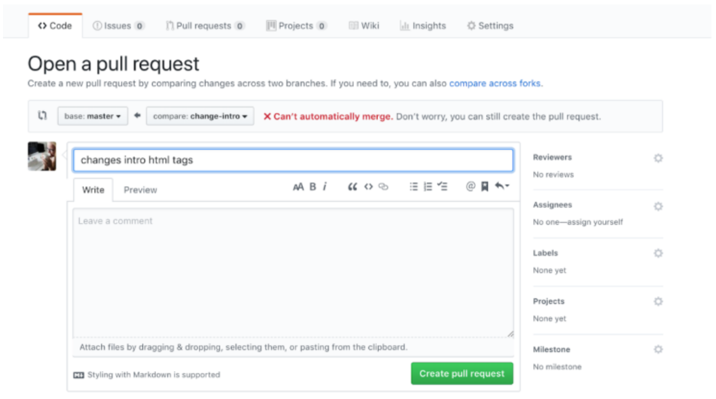
When you are working on repositories with more than one collaborator it is likely that at some point a conflict will arise. And I don't mean conflicts in terms of a disagreement between collaborators on an aspect of the project. But conflicts where one collaborator is working on an older version of the repository that has already been updated.

Remember the first rule of git? Always pull down changes first will avoid most conflicts, but sometimes they still slip through the net. The most likely time when conflicts will happen is when collaborators are working on the same repository at exactly the same time.

Let's set up a conflict scenario so that we can learn how to resolve it.

Line 14 of index.html was changed in a change-intro and master branch. When we then try to merge the branch back in, it won't as it cannot automatically work out which line has precedence. So it requires you to tell it manually. We can do this by opening a pull request.

You can still open a pull request with a merge conflict, but you have to manually fix the conflicting changes


Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

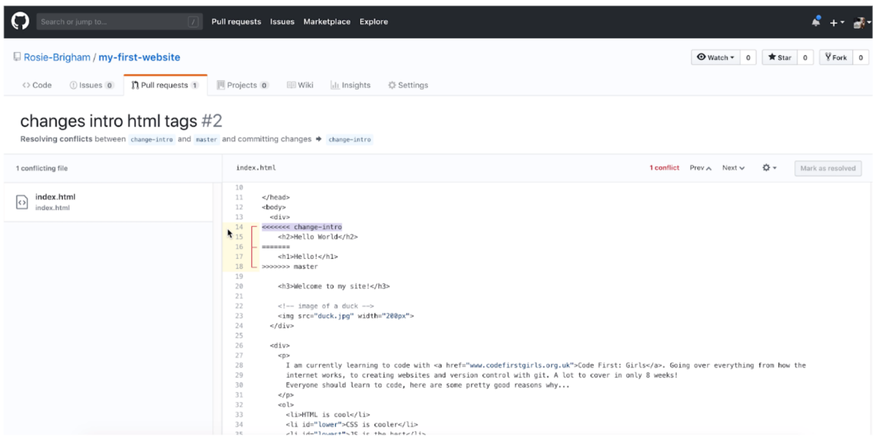
base: master ← compare: change-intro ✕ Can't automatically merge. Don't worry, you can still create the pull request.

changes intro html tags

Here is the pull request to merge the 'change-intro' branch into the master. As you can see, we can still open the pull request even though it cannot be automatically merged.

Handily, github gives us the option to 'resolve conflicts' manually and opens a web text editor. This is what we will do here.

This opens the web editor on github, simply delete the changes that you do not want and select 'mark as resolved'


Rosie-Brigham / my-first-website

changes intro html tags #2

Resolving conflicts between `change-intro` and `master` and committing changes → `change-intro`

1 conflicting file: index.html

```
</head>
<body>
  <div>
<<<<<<< change-intro
    <h2>Hello World</h2>
=======
    <h1>Hello!</h1>
>>>>>>> master

    <h3>Welcome to my site!</h3>

    <!-- image of a duck -->
    <img src="duck.jpg" width="200px">
  </div>

  <div>
    <p>
      I am currently learning to code with <a href="www.codefirstgirls.org.uk">Code First: Girls</a>. Going over everything from how the
      internet works, to creating websites and version control with git. A lot to cover in only 8 weeks!
      Everyone should learn to code, here are some pretty good reasons why...
    </p>
    <ol>
      <li>HTML is cool</li>
      <li id="lower">CSS is cooler</li>
```

© CodeFirst:Girls 2017 78

Simply delete the version of the line, or lines, that you do not want. Make sure you also delete the chevrons as well, otherwise they will appear in your code. When you're done, hit the 'mark as resolved' button.This creates a new merge commit, and the conflict has been resolved!

Make sure you then pull this down onto your machine.

How to avoid and reduce merge conflicts

- Anytime you want make changes to code in a shared repository make sure that you pull down new changes 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.

Finishing off:

1. Make sure both your first website and group project is on github, and your first site is published on github pages

Part 6 - Bootstrap

Chapter 20 - 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 with all browsers.

We have learned quite a bit of CSS by now and it all seems fairly straightforward. You write some CSS, tweak it here and there until it looks the way you want, and you're done! In theory that is exactly how CSS is meant to work and why it is brilliant.
As designs get more and more complex, for larger and larger sites - you will need to write more and more CSS code. For simple designs like navbar, tables and modals, lots of other people will have written css to create them on their website, so it is counter-productive for everyone to write their own version.

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 21 - Bootstrap.

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 Bootstrap, you need to do two things:

1. Link to the Bootstrap stylesheet in the head of your html page.
2. Attach the relevant 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**.

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 fluid responsive design - change the size of your browser to see the difference.

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 be using the CDN
Task:

1. Fork the following into your github account: https://github.com/CodeFirstGirls/beginners-part-five
2. Click on the **fork** icon, to copy it onto your github account
3. On your copy, Click **clone or download**
4. Select the **Open in Desktop** option.
5. Clone the repo into your coding_course folder
6. Open the folder in Atom

Chapter 22 - 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

There many different bootstrap components, but the most common (and therefore the ones we will focus on today) are nav bars, grid layouts, buttons, modals and carousels.

Page margins

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

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

Columns

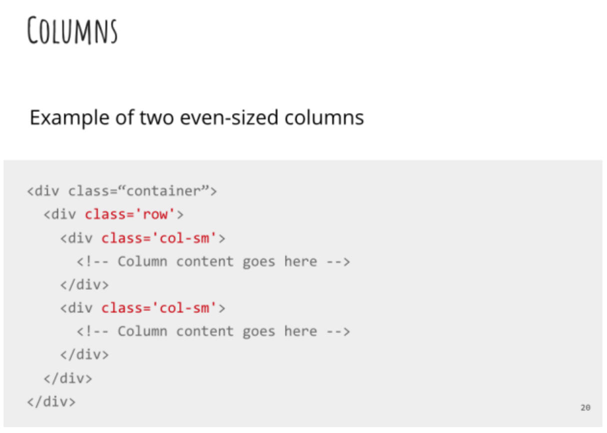
Inside a `.container` you might want to divide things up further and maybe create columns. Bootstrap works on a grid layout with 12 columns. To create a column layout you use a combination of **`.row`** and **`.col`** classes

1. First you need to create a .row which is going to contain your columns. You do this by adding a **`<div>`** 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

e. If you leave it without - it will apply it to all devices



Columns

Example of two even-sized columns

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

20

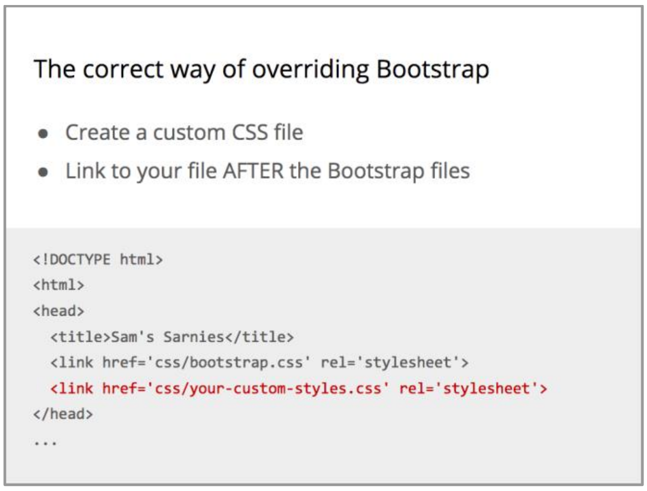
This is an example of two even sized columns would look like in your HTML.

Chapter 23 - Modifying Bootstrap

Our site looks pretty good now by only adding a couple of things, but maybe you don’t like how Bootstrap makes some things look? Also, imagine if all websites in the world used 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

- Create a custom CSS file
- Link to your file AFTER the Bootstrap files

```
<!DOCTYPE html>
<html>
<head>
  <title>Sam's Sarnies</title>
  <link href='css/bootstrap.css' rel='stylesheet'>
  <link href='css/your-custom-styles.css' rel='stylesheet'>
</head>
...
```

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

Chapter 24 - Time to play!

Task: Carry out the instructions using the bootstrap template we downloaded earlier in the class. If in doubt, reference bootstrap for help! Try the following things:

- A Carousel
- A modal
- Overwrite the styles to make the links different colours

Part 7 - JavaScript & JQuery

Chapter 25 – 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 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 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**.

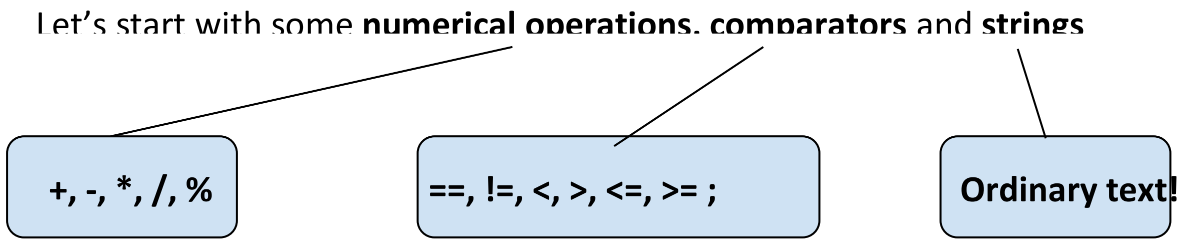
3. Programming languages are organised into **paradigms**, that is - ways of thinking and communicating with the computer.
4. JavaScript is powerful because it is:
a. **multi-paradigm** (it can accommodate a range of thinking/programming styles),
b. **prototype-based** (things that you have defined can 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 26 –Getting started with 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!


Let's start with some **numerical operations**, **comparators** and **strings**
+, -, *, /, %
==, !=, <, >, <=, >= ;
Ordinary text!

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 **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

Task: in your console, try out the following. Try and guess what the answer will be:

1. `5 + 2`
2. `"Hello" + " " + "world"`
3. `25/6`
4. `1 == 1`
5. `1 + 1 == 2`
6. `360 > 70`
7. `10 * 34`
8. `10 != 10`

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.

You include the JQuery library in your site by adding the following in between your <head></head> tags:

```
<script src="http://code.jquery.com/jquery-3.2.1.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.

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

Task:

1. Fork the project for ‘week 6’ and clone it into your coding folder
2. Add jquery using a cdn: make sure it is called above the other js files
3. Read through the code on the ‘background.js’, try and complete the function which will change the background when the button is clicked
4. Extension task: using variables, change the text in the span to be the name of the artist when the background changes

   Remember to ask google your questions, and if you get stuck, check out the solution branch

Useful resources

- https://stackoverflow.com/
- https://www.w3schools.com/
- https://css-tricks.com/
- Ben Howdle talks about different JS Frameworks

CodePen/jsFiddle

CodePen and jsFiddle are sites which allows you to try out small bits of HTML, CSS and Javascript. It’s a really useful tool for getting good help with JavaScript (and HTML/CSS) online: If you’re having a problem:

1. Create a jsFiddle or CodePen showing what you’ve tried.
2. Post on StackOverflow describing the problem, with a link to the jsFiddle/CodePen.

People will be able to help you better if they can see the code themselves. Often they will respond with a working jsFiddle. (as in this example)

CodePen is used by many front-end devs to showcase their portfolios - it even has a “hire me” button.

Finishing off

Task:

Add JS to your website. Continue to work on your first_site until it's something you can be proud of!

Group Project

Task:

Meet outside of class to work on your project!

Additional suggestions you could think about for your project are here.

Part 8 - Optional extras

Students can pick a couple of these subjects and run through them (in 30 mins)

Chapter 27: Google Analytics

Google Analytics is an analytics service provided for free by Google. It allows you get an overview of how many people are visiting your site, where they come from, what they do on your site, and much more.

How it works

To use Google Analytics you need to place JS Plug-In, a snippet of JavaScript, (that they provide) on each of the HTML pages on your site. When a user visits the page, the javascript sends a message to the Google Analytics site logging the visit.

Task:

1. Set up a Google Analytics account - You want to choose the default 'Universal Analytics' option.
2. Go to the **Admin** section, create a new account for your personal site.
3. Click on the **Tracking Info** under the **Property** section, click on "**Tracking Code**" and install the analytics code on all the pages of your site.

Chapter 28: 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.

Chapter 29: Domain names for GitHub pages

Remember in Session 1 we previously introduced you to Domain Names? In case you need a quick recap, in order 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

We previously gave you some examples of web hosting and domain registrars you could contact, feel free to recap this from Session 1. As you've been working with Github and publishing your sites via Github pages we wanted to give you some more guidance on setting up a custom domain using Github Pages.

Task (if you have bought a personal domain name):

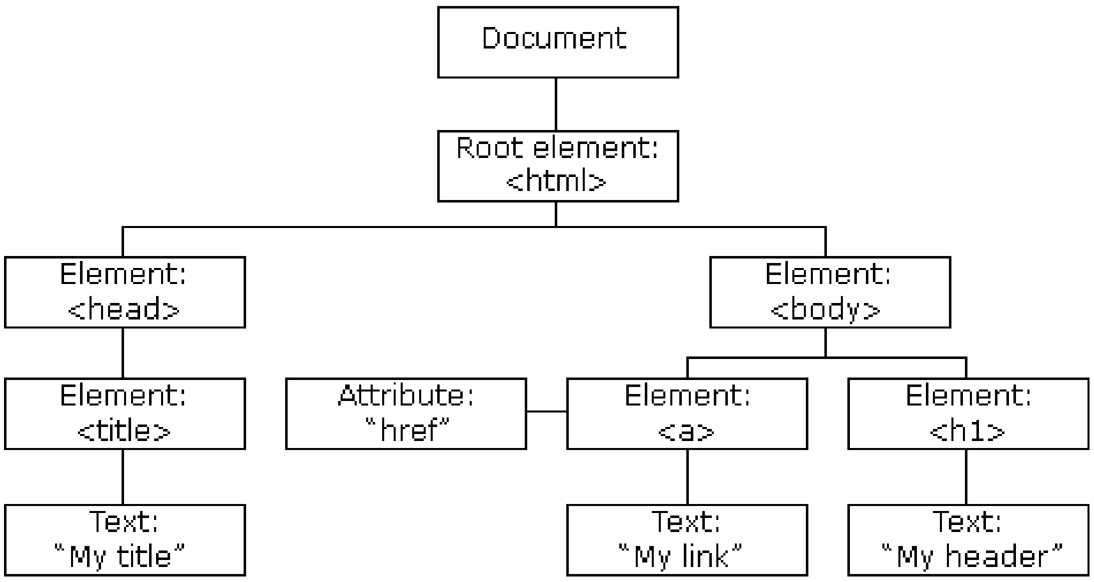
For Github pages there are three main stages to setting up a custom domain:

1. Pick a custom domain and register it with a DNS provider/Domain Registrar/DNS host (three names for the same service). In Session 1 we provided some examples of these: 123-reg.co.uk, godaddy.com and namecheap.com.
2. Add your custom domain to your Github Pages site. Follow these instructions here.
3. Set up your custom domain with your DNS provider. For more information on how this should look find some examples from Github pages here and here.

Chapter 30: HTML DOM

As we have covered in this course, HTML, CSS and JavaScript are the three main components that make almost every website. How do browsers comprehend our code?

When they receive code, browsers start building websites by converting all of the HTML



they receive into a JavaScript object called the Document Object Model (DOM).

In fact, from the Developer Tools console, you can examine the DOM for any website.

The DOM is demonstrated by the node-tree visual representation in your developer tools, identifying relevant attributes (and their respective values), CSS styles, JavaScript event listeners, breakpoints, and properties of each HTML element. This is important because this is how we create dynamic websites; Through the DOM, JavaScript is able to manipulate all aspects of a web page, from HTML elements and their attributes, CSS styles, adding or removing new HTML elements & their attributes, reacting to existing HTML events on the page, to finally, creating new HTML events on the page.

Formally, the HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** (values you can obtain or set) of all HTML elements
- The **methods** (actions you can take) to access all HTML elements
- The **events** (scenarios which actions lead to) for all HTML elements

It is a W3C Standard for how to manipulate HTML elements. A lot of functions we have used so far in jQuery have made use of the DOM to tell the computer what to do when users interact with the page.

Chapter 31: Interacting with APIs, working with company APIs

What is an API?

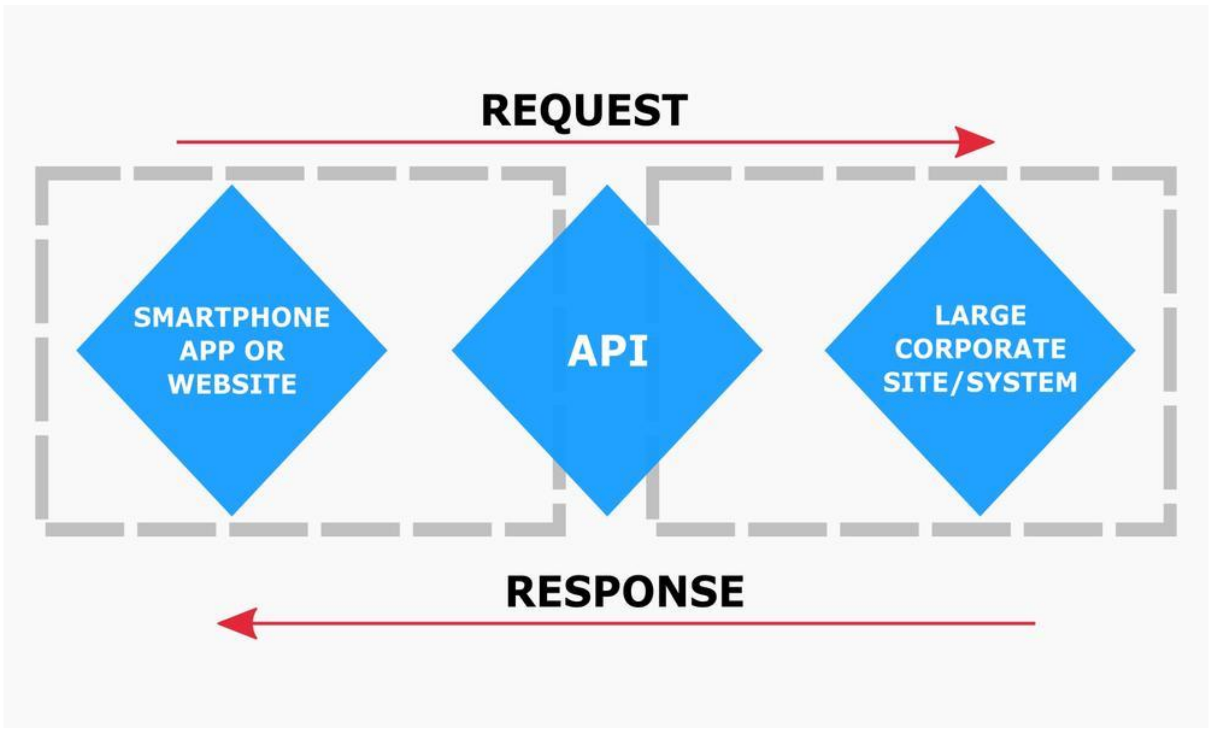
The web is made up of a large network of servers, all connected together. Every web page on the internet is stored on a remote server. A remote server is part of a remotely located computer that is "optimized" to process requests

When you type www.facebook.com into your browser, a request goes to Facebook's remote server. Once your browser receives the response, it interprets the code and displays the page

To the browser, also known as the *client*, Facebook's server is an API. This means that every time you visit a page on the Web, you interact with some remote server's API

An API isn't the same as the remote server — rather **it is the part of the server that receives requests and sends responses**.

Or, to put it another way:



Interacting with APIs

The DOM is of the utmost importance, next to the ability to make requests to a server (XMLHTTPRequest*) when we're thinking about interacting with APIs, because they come together to set the standards for **AJAX (Asynchronous JavaScript and XML*)**

AJAX is a technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.

What are REST and AJAX? How are they different?

To fully understand AJAX, we need to understand how information on most websites is communicated over the standard web protocol, HTTP (HTTPS is similar with a layer of security) - using **REST (Representational State Transfer).**

AJAX is not the same as REST. AJAX is a programming interface that allows us to implement a set of client-side data handling techniques to **retrieve and access data** from a server, whereas...

REST is an architectural style - it is a **standard way** of handling, sending and responding to **HTTP (Hyper-Text Transfer Protocol)** requests.

JSON

So now we know (roughly) how clients fetch data (AJAX), and how applications are built on the server-side to listen for our requests for data (REST architectures), so…how is the data received?

JSON request example (from Facebook)

This Facebook example demonstrates how their Graph API is structured: This GET request:

```
GET graph.facebook.com
  /facebook/picture?
    redirect=false
```

Returns this JSON:

```
{
  data:
    {
      is_silhouette: false,
      url:       "https://scontent.xx.fbcdn.net/v/t1.0-
1/p100x100/12006203_10154088276211729_2432197377106462187_n.png?oh
=d1b6b18e1846c1adefd157a45c4d384d&oe=58226457"
    }
}
```

Facebook's Graph API was built to handle requests with specific parameters, in order to return their logo to us in JSON form.

Working with company APIs

What are some good APIs for us to try out on our websites?

Let's go with the well-established, public facing APIs of some big tech cos:

- Twitter for Websites
- Google Maps
- Facebook Social Sharing Plugins
- Facebook Graph API

As Developers, we love tools that make our lives easier, so let's get a useful tool for interacting and testing interactions with APIs. Postman is a Chrome extension that does just what we need it to.

Some companies provide their own consoles, which allow us to try out HTTP requests in browser too; pretty handy!

Finishing off:

Task:

Add plugins and metrics to your project.

Watch this video and this video to consolidate your knowledge of RESTful web APIs and how to use them.

Refresh your memory on AJAX and how it is used in websites and web applications to interact with APIs with this video.

Group Project

Task:

Meet outside of class to work on your project!

Additional suggestions you could think about for your project are here.

(link to document including suggesting for working with Bootstrap, JavaScript and jQuery, drop-down menus, pop-up boxes, image slideshows / carousels, a form to send an email, hover effects, and more!)

Appendix

Awesome features of Bootstrap

Drop-down menus

Check you've included the Bootstrap and JQuery stuff above, then look at this section: https://getbootstrap.com/docs/4.1/components/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: https://getbootstrap.com/docs/4.1/components/modal/

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: https://getbootstrap.com/docs/4.1/components/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 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

A simple to do 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!

Continue your learning

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!

Charlotte, who used to work for Code First Girls, has set up a fantastic repo which is kept up to date with new resources for continuing learning, check it out here.

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