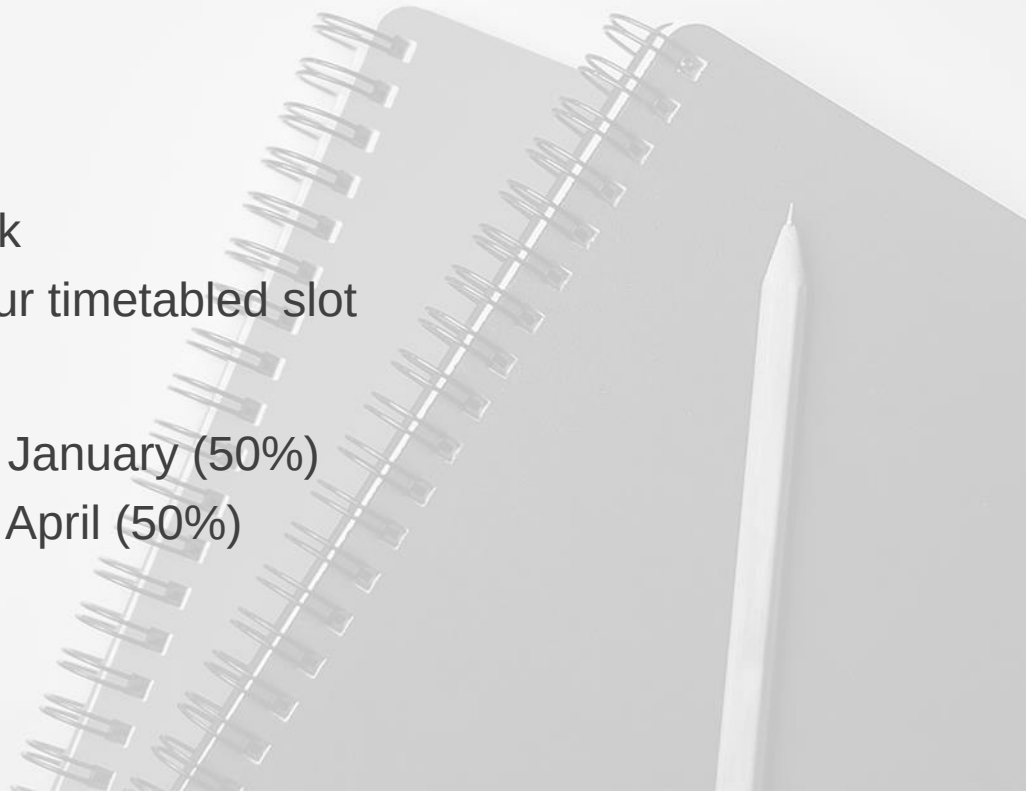
CSY2028
Web Programming

Topic 1 – Introduction / Recap

- Module Overview
- Module Objective
- HTML & CSS Recap

Module Overview

- Module Structure
  - 20 Credits
  - 22 Weeks
    - 3 hour session per week
    - Ensure you come to your timetabled slot
  - Assignments
    - One assignment due in January (50%)
    - One assignment due in April (50%)



Module Objectives

- Develop an understanding of web programming including
  - Development tools and techniques used in industry
  - Best Practices
  - PHP Programing
- You should already have the following:
  - Basic programming skills
  - Basic website building skills (HTML)
  - Basic database skills (MySQL)

How do the sessions work?

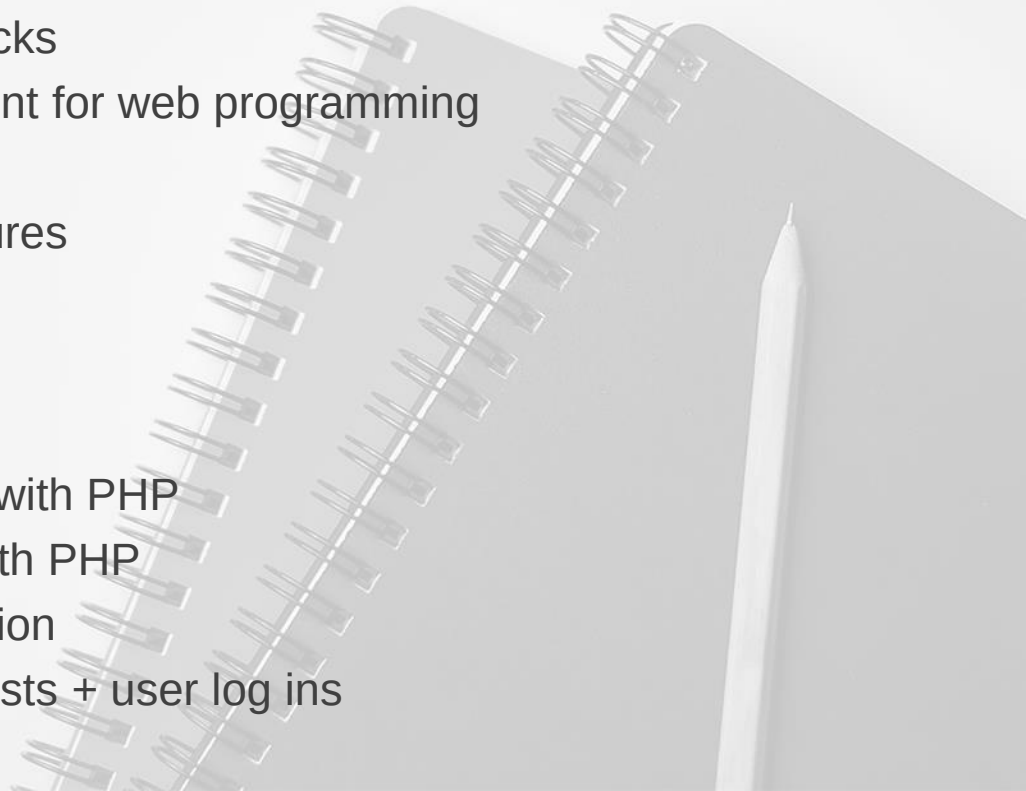
- 2 hour practical session
  - Notes will be broken up into topics
  - Each topic may take one or more lectures
  - Exercises will be mixed in with the notes
  - We will go at the speed of the class, if something takes longer than expected we'll move some of the lecture to the following week
  - Feel free to ask questions
  - Slides will be on NILE

How do the sessions work?

- Exercises
  - Sometimes exercises will be flagged with a difficulty
  - Everyone must do the grade D-C exercises
  - Others are optional and may be a lot more difficult
  - If there is no grade associated with an exercise, everyone should do it
- The exercises may require more time than we have in the seminar, be prepared to work outside the class
- You may work in small groups (Except for assignments!)

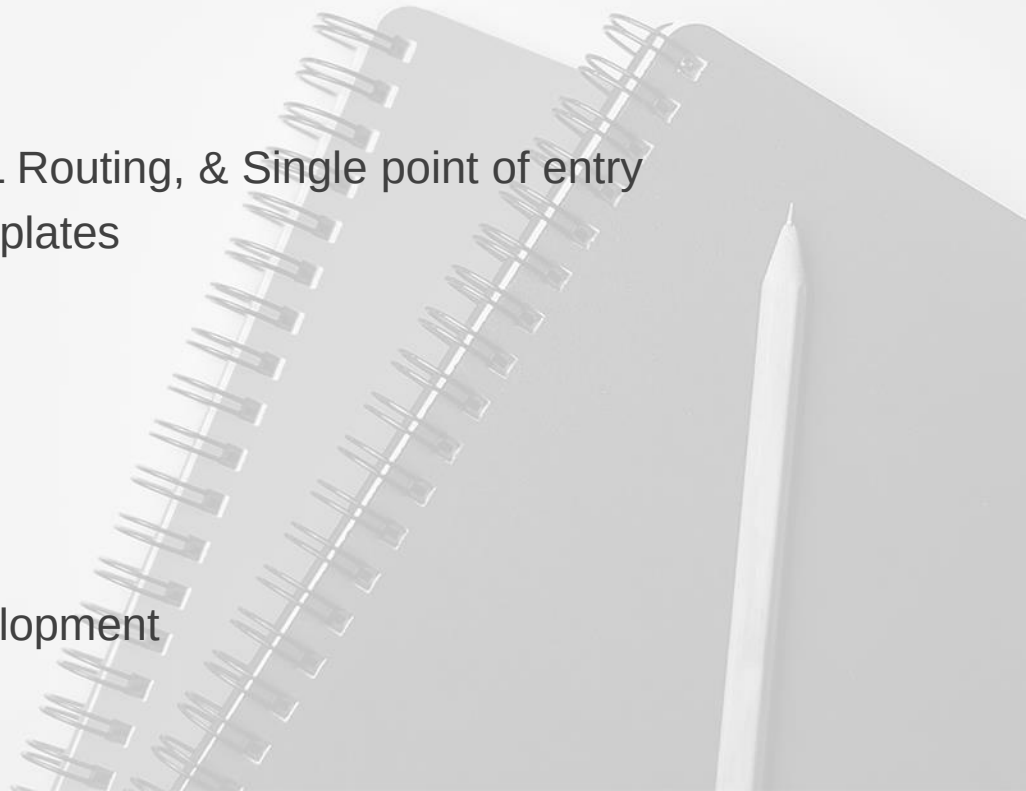
Topics Covered

1. Introduction + Basic HTML/CSS recap
2. HTML5 + Advanced CSS tips and tricks
3. Setting up a development environment for web programming
4. PHP – Introduction
5. PHP – Functions and Control Structures
6. PHP – Arrays and data structures
7. PHP – Handling User Input
8. Object-Oriented PHP Code
9. MySQL – Introduction + Connecting with PHP
10. MySQL – Manipulating databases with PHP
11. PHP, MySQL and Database Abstraction
12. PHP- Maintaining state across requests + user log ins



Topics Covered

13. Revision session
14. Object Oriented PHP part 2
15. Object Oriented PHP part 3
16. URL rewriting with mod_rewrite, URL Routing, & Single point of entry
17. Separation of concerns + HTML Templates
18. Complex HTML forms + automation
19. MVC (Model-View-Controller)
20. Building reusable components
21. Javascript – jQuery
22. Javascript – Ajax
23. PHP – Unit Testing/Test Driven Development



Module Focus

- Learn how to build websites using PHP and MySQL
- Object-Oriented Programming in PHP
- Best Practices- Why should you structure your code in a certain way?
- Making code reusable, removing duplication of effort

Today’s Lecture

- Today's lecture will cover :
  - HTML
  - CSS
  - Creating a basic web page

HTML

- Basic HTML rules:
  - All HTML files should start with <!DOCTYPE html> (This is the only doctype you should use, all others are now very out of date!)
  - All HTML files should have a <html> tag which wraps everything else
  - All HTML should have a <body> tag which wraps all the content
  - Most HTML files should have a <head> tag which contains metadata

HTML

- A basic HTML file looks like this:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
  </head>

  <body>
    <h1>Page heading</h1>
    <p>Page content</p>
  </body>
</html>
```

HTML

- HTML is forgiving
  - You can make mistakes and the browser will try to fix it
- But you shouldn't rely on this behavior!
  - Try to always write valid HTML
  - You can (and should!) check your HTML for errors using the W3C HTML Validator http://validator.w3.org/
  - If your page isn't displaying as you expect, it's probably got an error and the validator will help you find out how to fix it by telling you where to look (e.g. tag name or line number)

Useful HTML Tags

- Link to another page or website
  - Click here to go to google
- Display an image:
  -
  - Always supply an alt attribute, this is used by screen readers and when the image cannot be loaded
- A paragraph of text
  - text
- Headings of various levels (h1 is the top “biggest” header)
  - <h1>Heading</h1>, <h2>Heading 2</h2>,….. <h6>Heading 6</h6>

CSS

- CSS is a separate language which is used to describe how a HTML file looks when it's opened in a browser

- CSS should be in its own file (or multiple files) rather than in the HTML file

CSS Rules

- Each rule is used to declare the way the element should be presented. For example

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

- Targets any H1 element and sets the colour of the text to red
- Notice the American spelling of colour without a u!

External Stylesheet

- To attach a css file to a HTML file, save the file with a .css extension in the same directory as the HTML file and you can reference it in the HTML using a <link> tag inside the <head> tag

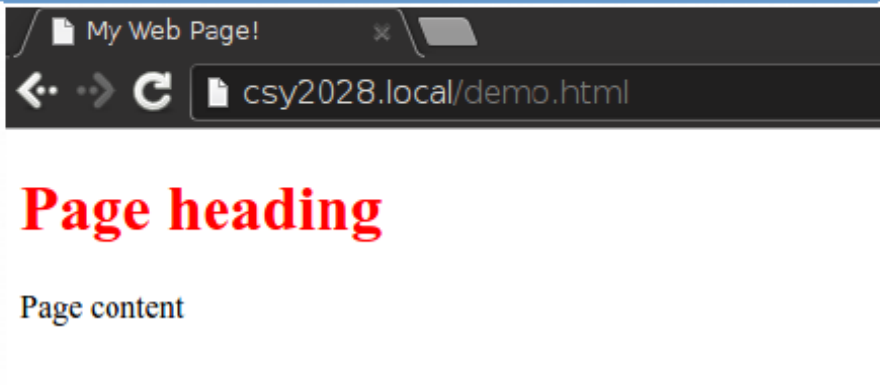
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1>Page heading</h1>
    <p>Page content</p>
  </body>
</html>
```

CSS

- Now, when you open up the page in a browser it will pull in both the HTML and CSS then apply the CSS rules to the HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1>Page heading</h1>
    <p>Page content</p>
  </body>
</html>
```

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

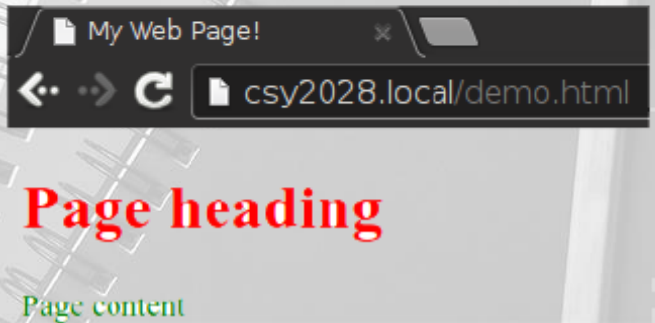


CSS

- You can add as many rules to a CSS file as you like.
- Each rule has a selector, an opening brace, some properties and a closing brace

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1>Page heading</h1>
    <p>Page content</p>
  </body>
</html>
```

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

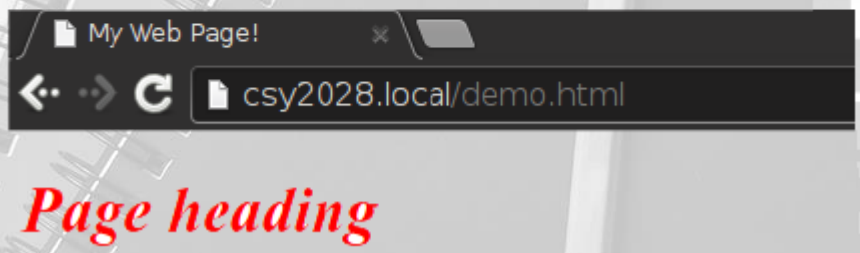


CSS

- Each rule can have one or more properties. For example:
- To change the font to italic you can use the property font-style with the value italic

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1>Page heading</h1>
    <p>Page content</p>
  </body>
</html>
```

```
h1 {
        color: red;
        font-style: italic;
}
p {
        color: green;
}
```



CSS Selector

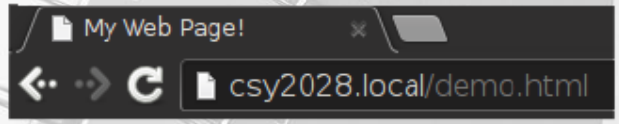
- There are three main CSS selectors
  - Element Name
  - Class Name
  - ID

Element Name Selector

- The element name selector selects all elements with the same tag name

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1>Page heading</h1>
    <p>Paragraph one</p>
    <p>Paragraph two</p>
  </body>
</html>
```

```
h1 {
        color: red;
        font-style: italic;
}
p {
        color: green;
}
```



Page heading

Paragraph one

Paragraph two

Notice that both paragraphs are now green

Class Name Selector

- Any element can be given a class this is an HTML attribute like src in images or href in links
- The class name selector works in the format
  - .name
  - A dot followed by the name of the class you want to match
- This will match any element that has class=”name”
- Note that the attribute does not include the dot in the HTML!

Class Name Selector

- The class name selector is a name prefixed with a dot and selects any element with a class attribute set to that value

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1 class="myclass">Page heading</h1>
    <p class="myclass">Paragraph one</p>
    <p>Paragraph two</p>
  </body>
</html>
```

```
.myclass {
        color: green;
}
```

Page heading

Paragraph one

Paragraph two

Notice that both elements with the class "myclass" are green

ID Selector

- The ID selector works very similarly to the class name selector
  - Uses a # prefix
  - #myId
  - Matches any element with id=”myId” set as an attribute

- Note that the HTML does not contain the # symbol!
- IDs are different to class names. You can only use an ID once: Each ID should apply to a single element on the page

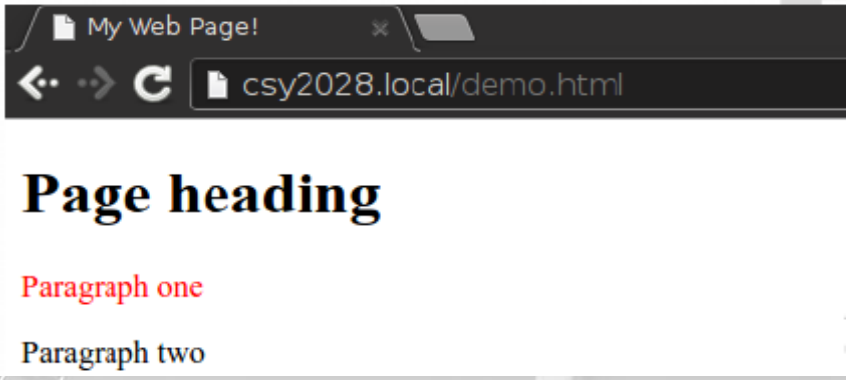
ID Selector

- The ID name selector is a name prefixed with a hash and selects the element with that ID

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <h1>Page heading</h1>
    <p id="myid">Paragraph one</p>
    <p>Paragraph two</p>
  </body>
</html>
```

```
#myid {
        color: red;
}
```

You may only have one element with each ID



Which Should I Use?

- In a lot of cases you can use a class name, or an element name or an ID to achieve exactly the same result!
- You should avoid IDs because you cannot easily reuse the style (if you want to make two elements red, you need to add two css rules and two IDs!)
- You should usually avoid element selectors as they're very imprecise
    - They will affect ALL elements of that type!
    - Sometimes, this is what you want
    - But if you want a specific style on a single part of the page, you might get unexpected results as the page grows and you add more elements

Combining Selectors

- HTML is a nested notation
  - (Some) Elements can exist inside (some) other elements
- You can use css to target nested elements. By combining a selector with a space you can target specific elements in the HTML DOM Tree
  - article h1 {font-weight: bold} – Sets any <h1> element inside an <article> element to bold
  - .myclass span {font-style: italic} – Sets any element inside an element with class=”myclass” to italic
- This will target elements any depth!

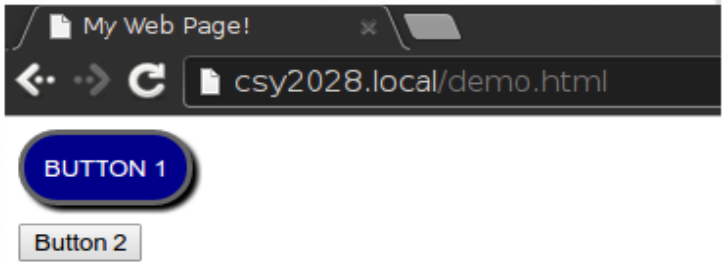
Combining Selectors

- You can combine selectors to be more specific

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <div class="myform">
      <input type="button" value="Button 1" />
    </div>

    <input type="button" value="Button 2" />
  </body>
</html>
```

```
.myform input {
  border-radius: 20px;
  border: 3px solid #666;
  box-shadow: 3px 3px 3px #000;
  margin-bottom: 10px;
  background-color: darkblue;
  padding: 10px;
  text-transform: uppercase;
  color: white;
}
```



Notice that only the button inside the element with the class *myform* has been styled

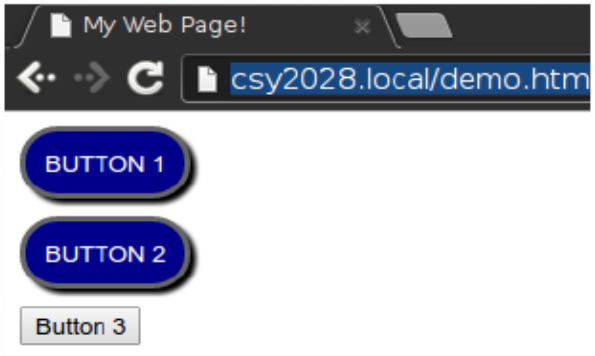
Combining Selectors

- This will target elements of any depth

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <div class="myform">
       <input type="button" value="Button 1" />
      <section>
        <form>
          <input type="button" value="Button 2" />
        </form>
      </section>
    </div>

    <input type="button" value="Button 3" />
  </body>
</html>
```

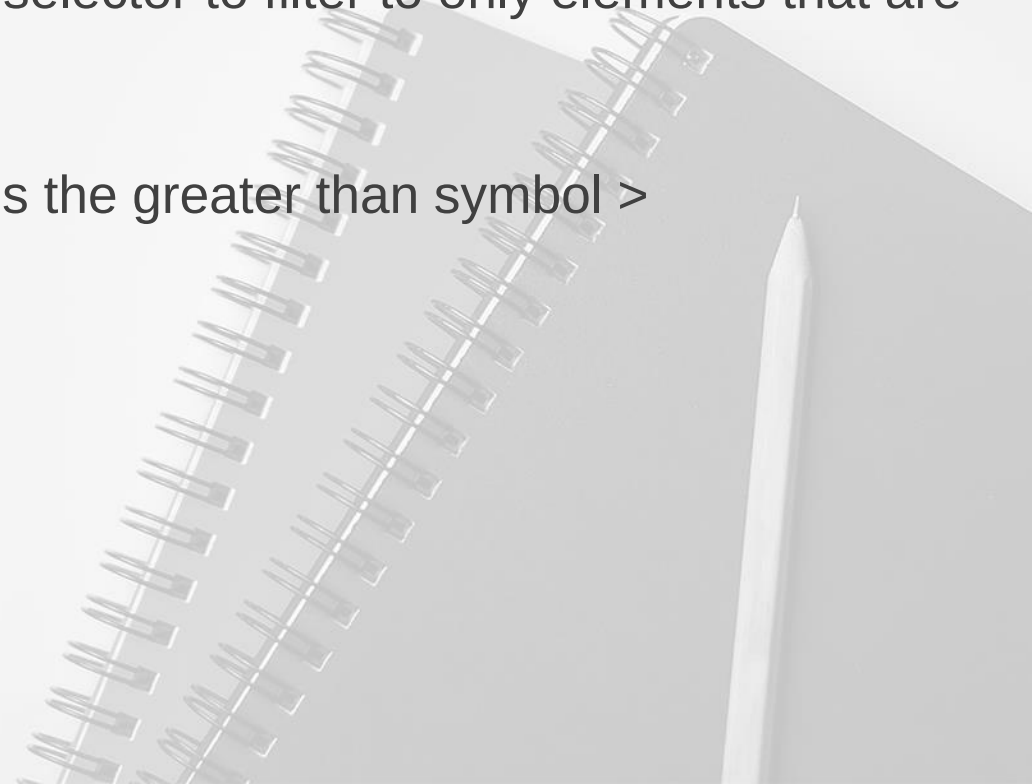
```
.myform input {
  border-radius: 20px;
  border: 3px solid #666;
  box-shadow: 3px 3px 3px #000;
  margin-bottom: 10px;
  background-color: darkblue;
  padding: 10px;
  text-transform: uppercase;
  color: white;
}
```



Notice that only the button inside the element with the class *myform*

Combining Selectors

- You can use the direct decedent selector to filter to only elements that are only one level down
- The direct decedent selector uses the greater than symbol >



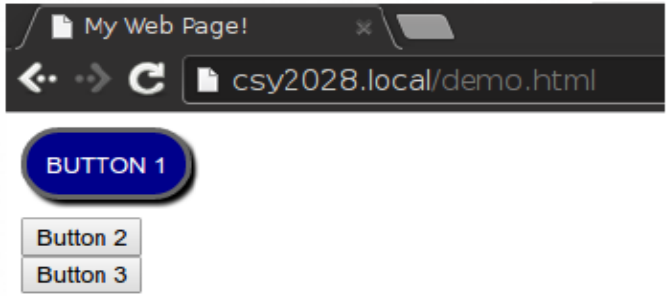
Combining Selectors

- This will target elements only one level down

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <link rel="stylesheet" href="demo.css" />
  </head>
  <body>
    <div class="myform">
       <input type="button" value="Button 1" />
      <section>
        <form>
          <input type="button" value="Button 2" />
        </form>
      </section>
    </div>

    <input type="button" value="Button 3" />
  </body>
</html>
```

```
.myform > input {
  border-radius: 20px;
  border: 3px solid #666;
  box-shadow: 3px 3px 3px #000;
  margin-bottom: 10px;
  background-color: darkblue;
  padding: 10px;
  text-transform: uppercase;
  color: white;
}
```



Notice that only the button inside the element with the class *myform*

Combining Selectors

- You can combine selectors into very complex expressions e.g.:
- #myelement > .myclass article section header h1
- However, this is generally discouraged
- We call this tightly coupled to the HTML: A slight change to the HTML will stop the h1 tag being styled
- As a rule of thumb you shouldn't need more than three levels of selector

Layouts

- CSS is used for describing how elements look on the screen:
  - Background colours, font colours, font sizes, font faces, spacing (paddings/margins)

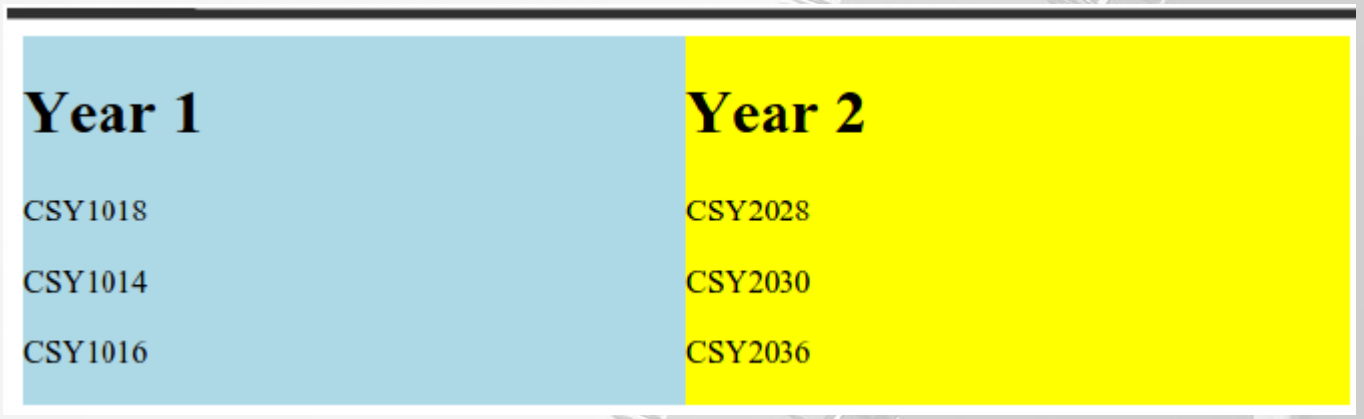
- It's other use is describing how the page is laid out: where on the screen the element should appear and how large it is

Layouts

- Don't use tables for layouts!
- Back in 1999 when browser support for CSS was limited, tables were the best available choice for positioning elements on the screen
- Using tables for this purpose has been labelled bad practice since the early 2000s!
- Don't do it!
- I will mark you down for doing this!
- Further reading:
  - https://www.hotdesign.com/seybold/everything.html
  - http://phrogz.net/css/WhyTablesAreBadForLayout.html
  - http://webdesign.about.com/od/layout/a/aa111102a.htm
  - http://www.htmlgoodies.com/beyond/css/article.php/3642151

Exercise 1

- Create a web page with two columns with a width of 50% each. In column 1 put all your first year modules, in column 2 put all your second year modules. You should give each column a background colour. It should look something like this:



Year 1	Year 2
CSY1018	CSY2028
CSY1014	CSY2030
CSY1016	CSY2036

Hint: You can use 50% widths on the Elements!

Exercise 1 Solution

```html
<!doctype html>
<html>
    <head>
        <title>Exercise 1</title>
        <link rel="stylesheet" href="ex1.css" />
    </head>

    <body>
        <div class="left">
            <h2>Year 1</h2>
            <p>CSY1018</p>
            <p>CSY1014</p>
            <p>CSY1016</p>
        </div>

        <div class="right">
            <h2>Year 2</h2>
            <p>CSY2028</p>
            <p>CSY2030</p>
            <p>CSY2036</p>
        </div>

    </body>
</html>
```

```css
.left {
    float: left;
    width: 50%;
    background-color: lightblue;
}
.right {
    float: left;
    width: 50%;
    background-color: yellow;
}
```

HTML5

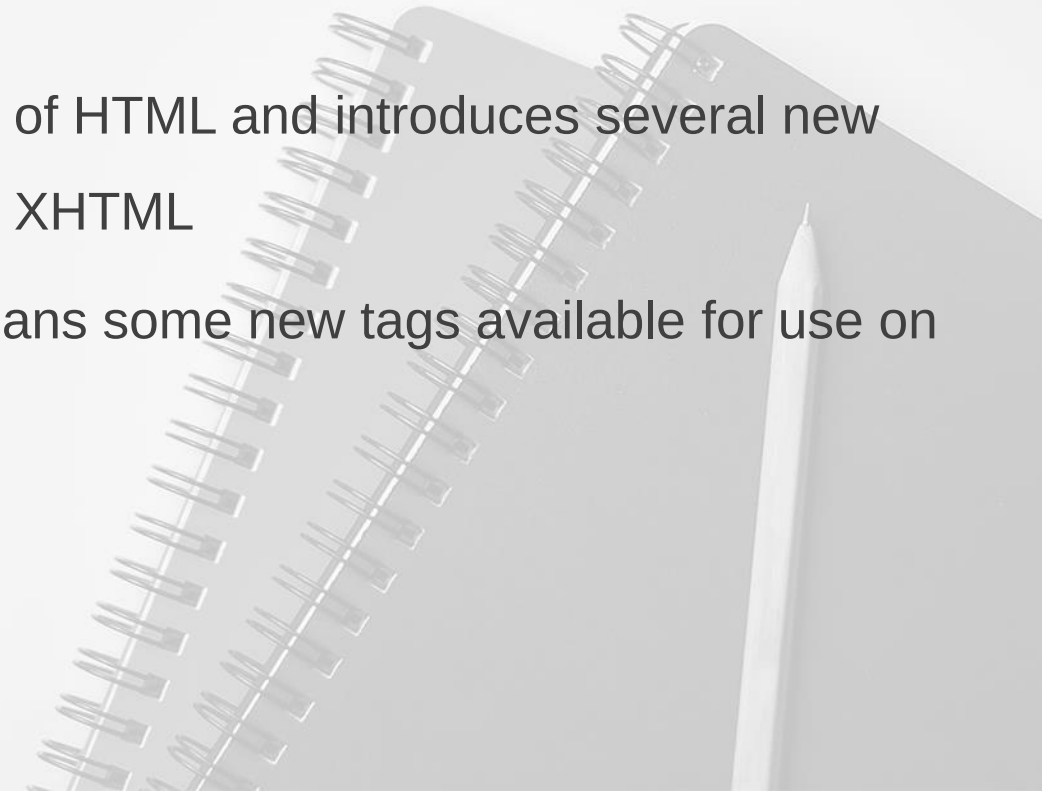
- HTML5 is the current version of HTML
- You should use HTML5 and not XHTML or HTML4
- A HTML5 page starts with the doctype

```
<!DOCTYPE html>
```

- If you are using any other doctype (or no doctype) browsers will not see your page as HTML5

HTML5

- What is HTML5?
  - HTML5 is the fifth version of HTML and introduces several new features over HTML4 and XHTML
  - For our purposes, this means some new tags available for use on our web pages



HTML5 Tags

- When the W3C (Worldwide Web Consortium) were drafting HTML5 and deciding what to implement they looked at the most commonly used CSS classes. E.g. pages like this:

```
<div cass="header">
    Header
  </div>
  <div class="navigation">
    <ul>
      <li>
        <a href="#">Link 1</a>
      </li>
    </ul>
  </div>


  <div class="main">
    <p>Lorem ipsum....</p>
  </div>

  <div class="right">
    Right hand side
  </div>

  <div class="footer">
  &copy; Your name 2015
  </div>
```

HTML5 Tags

- Using CSS classes to highlight common areas of the page caused several problems:
  - Extra typing for developers (duplication of effort)
  - Difficult for non-humans to understand which part of the page is which. E.g. search engines and screen readers
- The common CSS classes were turned into their own tags

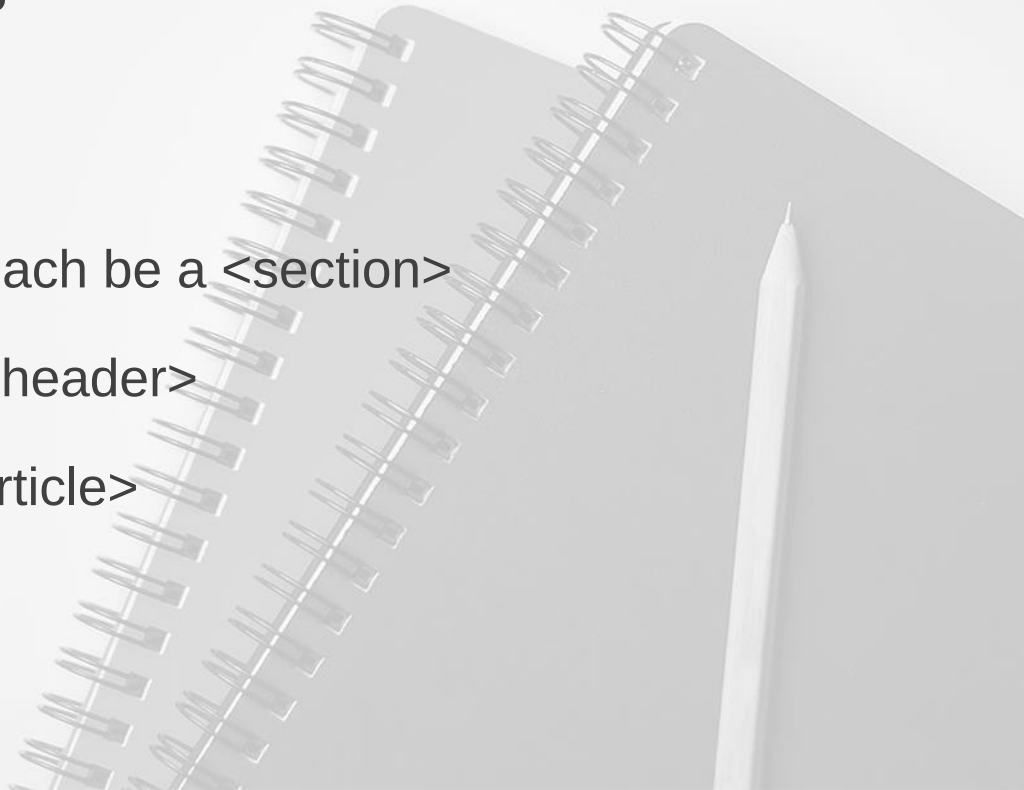
HTML5 Tags

- New tags available in HTML5

<header>	Describes a header (can be of a page or a section)
<footer>	Describes a footer (can be of a page or a section)
<main>	The main content of a page (this is useful for screen readers, they can skip straight to the content). You can only have one main element per page!
<article>	Describes a section which would make sense on its own. A paragraph would not but a topic on a forum would.
<section>	Describes part of an article, e.g. an individual post on a forum
<nav>	Contains the navigation for the web page. You can have more than one <nav> tag. This is also useful for screen readers as they can jump straight to the navigation
<aside>	Describes content related to a section/article that will be displayed differently/elsewhere e.g. sidebars

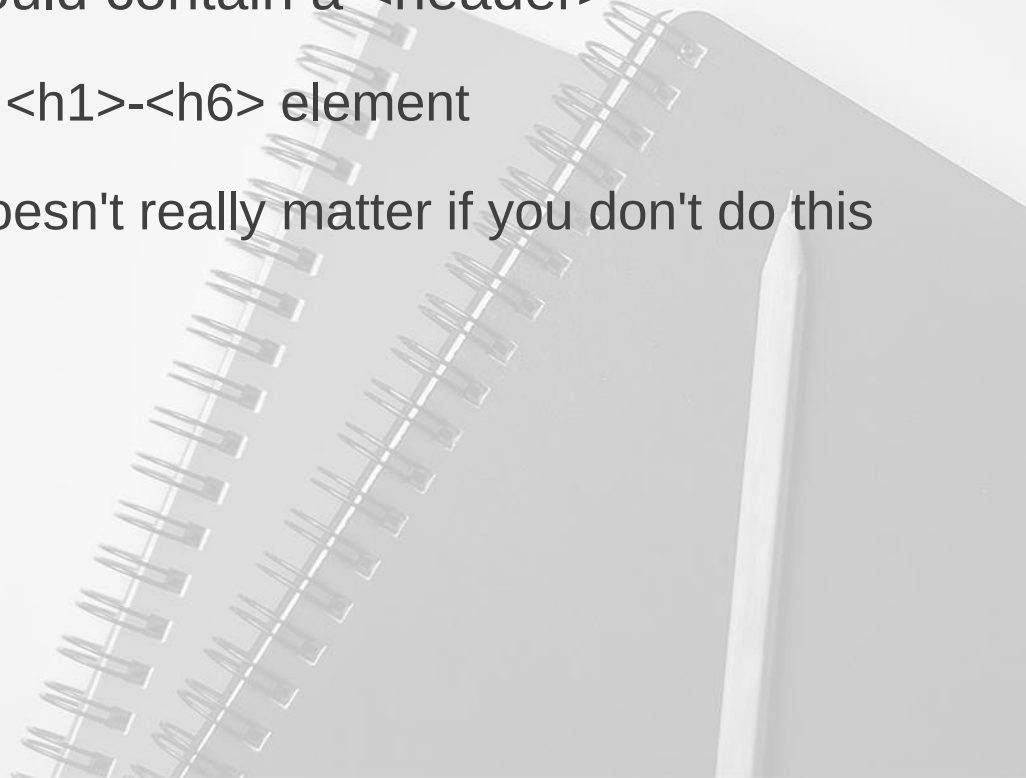
HTML5

- You should use an HTML5 tag rather than a CSS class if it's applicable
- For example:
  - Article comments should each be a <section>
  - Page headers should be <header>
  - Blog content should be <article>



HTML5 Headers

- Each <section> in HTML5 should contain a <header>
- Each <header> should contain a <h1>-<h6> element
- Although HTML isn't strict so it doesn't really matter if you don't do this



Responsive Web Design

- Responsive Web Design is a term for designing a web page that works in browsers of any size
- From widescreen desktop monitors to narrow phones
- This is done by having more than one stylesheet that is applied to the page:
    - e.g. One for mobile phones, one for tablets, one for desktops

Responsive Web Design

- You can apply different style sheets for different browser widths
- A desktop browser is usually over 1000px wide
- A mobile browser is usually under 500px wide
- Tablets are in between
- You can use these figures to apply different styles for different type of devices

Mobile Site Considerations

- You cannot get as much on the screen
- Fingers are big and clumsy compared to cursors, make buttons and links easy to press
- Navigation should be accessible via a button rather than a always visible using up valuable space
- You may want to hide some content entirely for mobiles to reduce clutter

Mobile Sites

- When creating a site for mobiles you should set the viewport with. This is almost always:

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

- This goes in the <head> tag

Conditional Stylesheets

- The <link> tag where you include your css has an optional media attribute
- This can be
  - all (default - is always applied)
  - screen (only gets applied when viewed on screen)
  - print (a stylesheet used when the web page is printed)

```
<link rel="stylesheet" href="demo.css" media="print" />
```

Conditional Stylesheets

- Additionally, stylesheets can be applied based on a screen width
- There are two main options
- max-width
- min-width

```
<link rel="stylesheet" href="mobile.css" media="screen and (max-width: 1000px)" />
```

- This will only apply mobile.css if the browser width is less than 1000px

Conditional Stylesheets

- This will only apply desktop.css to devices over 1000px

```
<link rel="stylesheet" href="desktop.css" media="screen and (min-width: 1000px)" />
```

- range (>300 and < 800px)

```
<link rel="stylesheet" href="mobile.css"
    media="screen and (min-width: 300px) and (max-width: 800px)" />
```

Multiple Stylesheets

- You can add more than one <link> tag to a page
- Each one can have a different media attribute
- It's a good idea to have at least three stylesheets:
  - One for styles that are shared by both desktop and mobile, e.g. fonts, background-colours, etc
  - One for styles relevant only to desktop browsers
  - One for styles relevant only to mobile browsers

Multiple Stylesheets

- For example

```
<link rel="stylesheet" href="main.css"
    media="screen" />

<link rel="stylesheet" href="mobile.css"
    media="screen and (max-width: 800px)" />

<link rel="stylesheet" href="desktop.css"
    media="screen and (max-width: 800px)" />
```

Stylesheet Precedence

- When you have more than one stylesheet, it's possible that they could contradict. For example
- Main.css

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

- Mobile.css

```
h1 {
        color: green;
}
```

Stylesheet Precedence

- When this happens, the last stylesheet included will override the first one

```
<link rel="stylesheet" href="one.css"
    media="screen" />

<link rel="stylesheet" href="two.css"
    media="screen" />
```

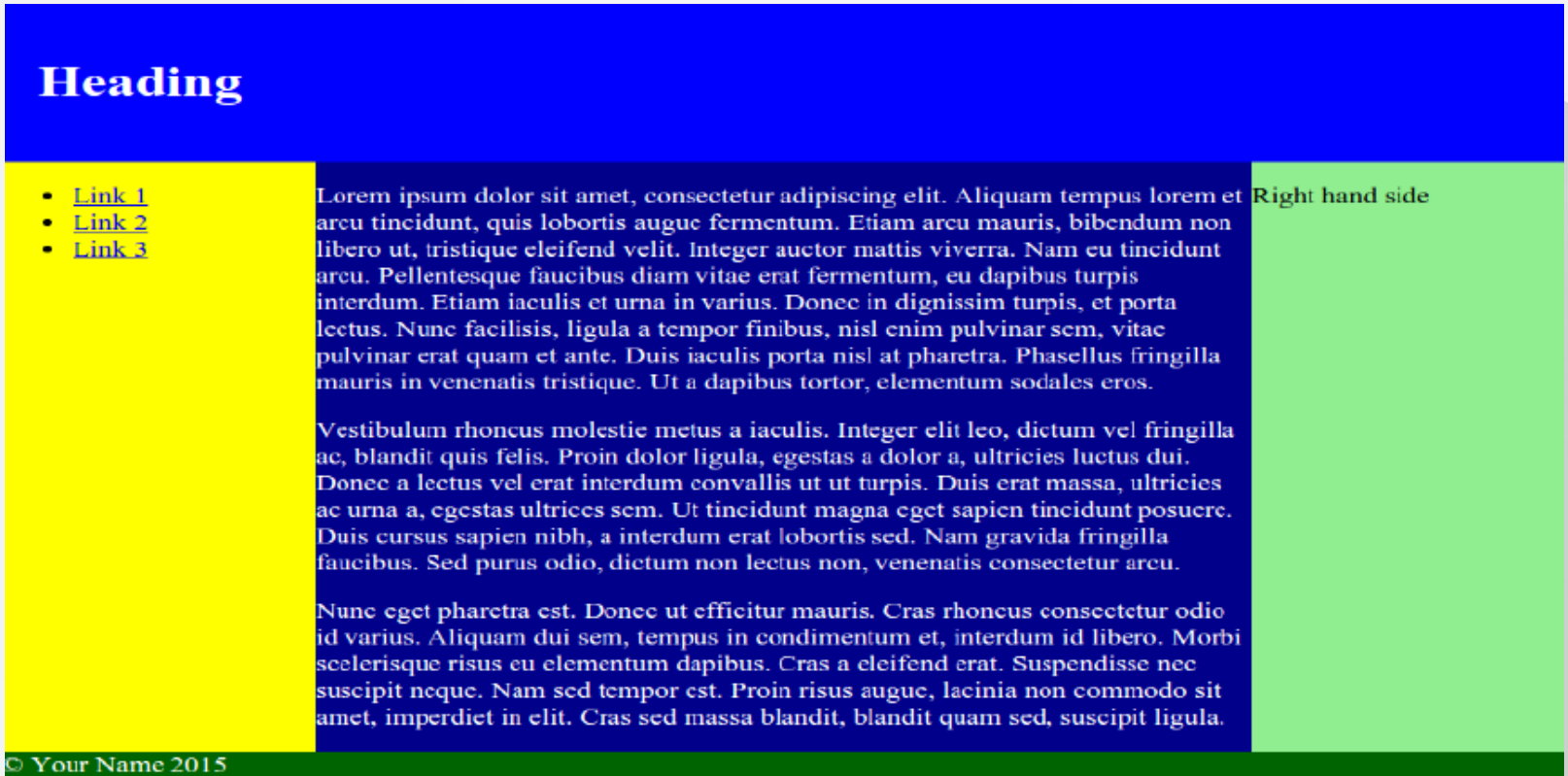
```
h1 {
        color: red;
}
```

```
h1 {
        color: green;
}
```

In this instance, the h1 wil be green because two.css was included last

Mobile Sites

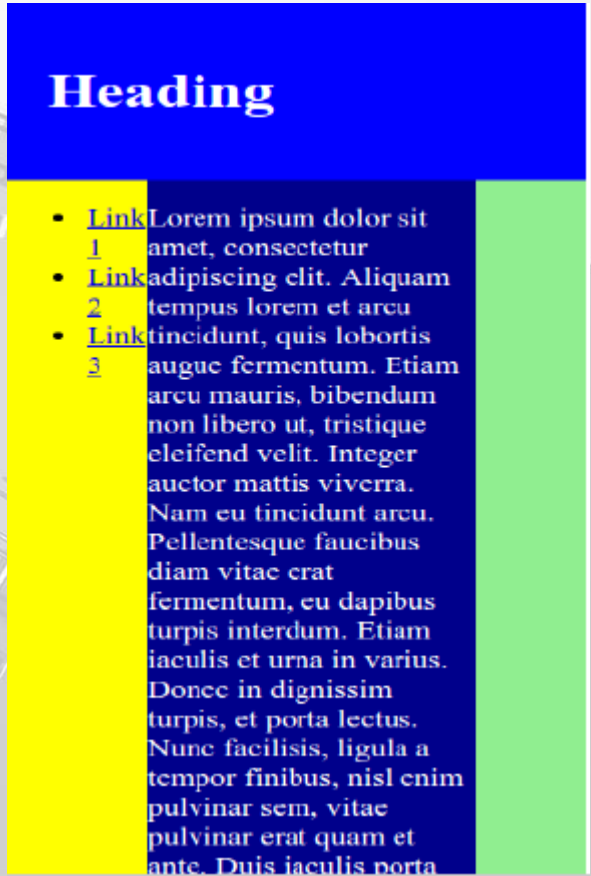
- This site is not mobile friendly



Mobile Sites

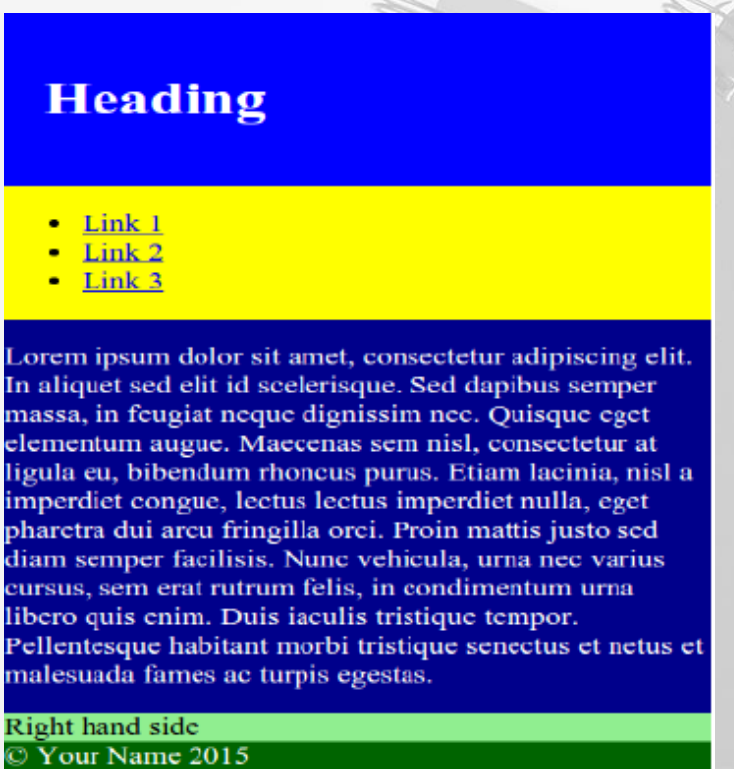
- On a mobile it looks like this:

Everything is squashed, the links are illegible



Mobile Sites

- By using a different stylesheet for mobile sites it's possible to “stack” the elements to look like this:



“Hamburger” Icon

- The “hamburger” icon is commonly used as an icon to display the menu on a mobile website
- This is so commonly used it's always good to follow convention and use the icon
- You can use an image for this, but it's possible to draw it using purely CSS

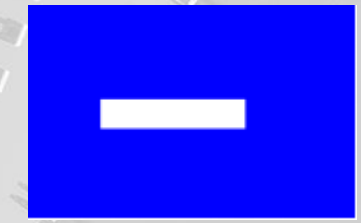
Create a Hamburger Icon

1. Create a link <a> element in the html which will be used as the button (this doesn't need any text)

```
<a class="shownav"></a>
```

2. Style it as a line with display: block, a background-color and a width/height

```
.shownav {
    background-color: white;
    height: 5px;
    width: 20px;
    display: block;
}
```



Create a Hamburger Icon

3. Add border-radius: 5px if you want it curved

```
.shownav {
    background-color: white;
    height: 5px;
    width: 20px;
    display: block;
    border-radius: 5px;
}
```

4. You can use box-shadow to make an element cast a shadow. Syntax is:
    - Box-shadow: [X offset] [Y offset] [Blur amount] [colour]
    - e.g. box-shadow: 10px 10px 5px black will cast a black shadow 5px across, 10px down with a 5px blur

Hamburger Icon

```
.shownav {
    background-color: white;
    height: 5px;
    width: 20px;
    display: block;
    border-radius: 5px;
    box-shadow: 10px 10px 5px black;
}
```

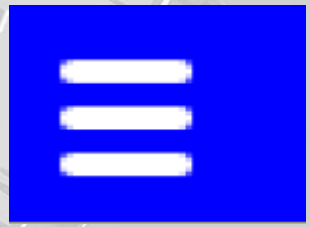
- By adding a shadow without a blur, below the line it's possible to draw the bottom part of the icon

```
.shownav {
    background-color: white;
    height: 5px;
    width: 20px;
    display: block;
    border-radius: 5px;
    box-shadow: 0px 10px 0px white;
}
```

Hamburger Icon

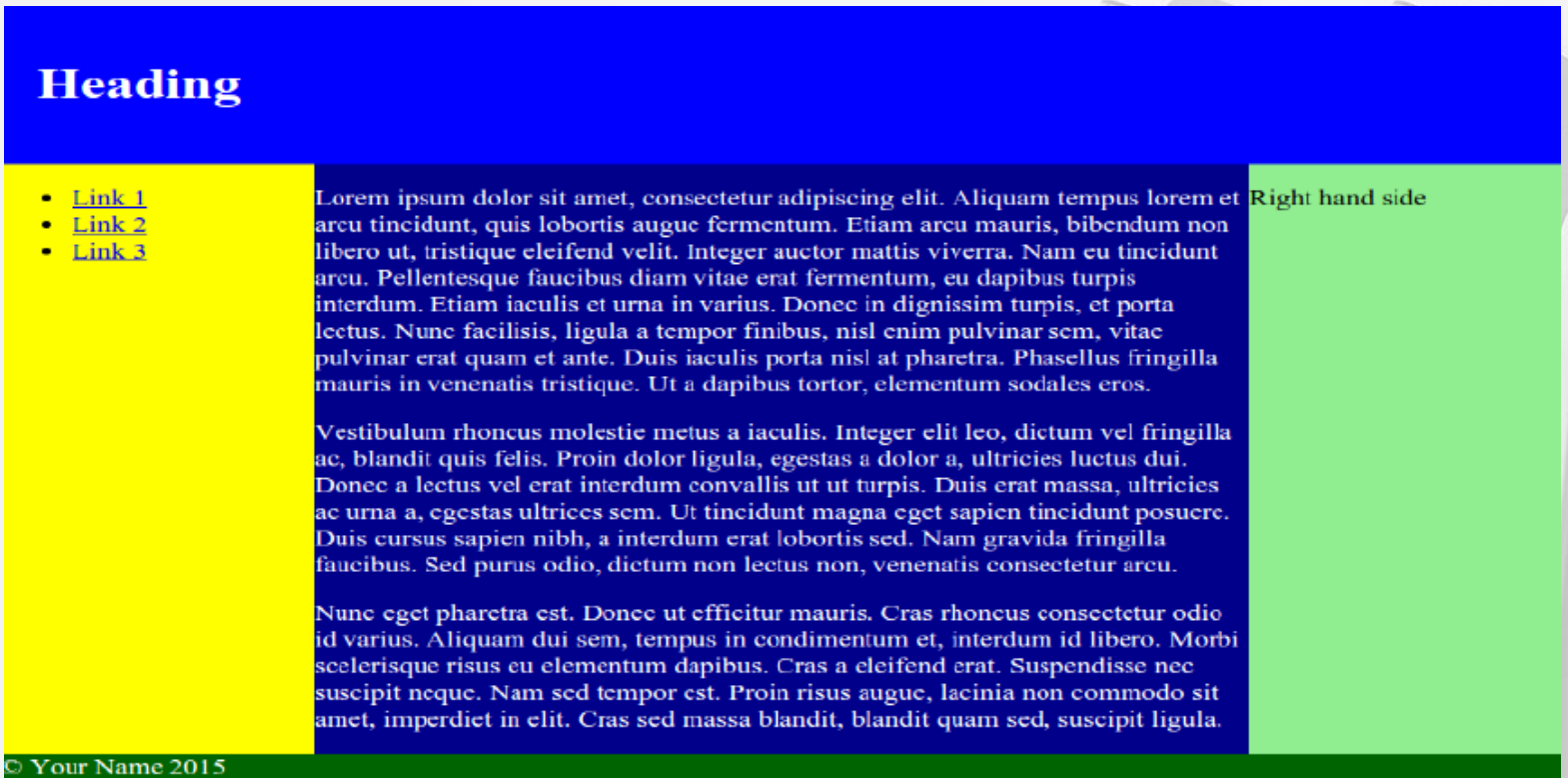
- You can add more than one shadow to an element. The syntax is:
  - box-shadow: [SHADOW 1], [SHADOW 2]
- By drawing a second shadow above the line you can complete the hamburger icon

```
.shownav {
    background-color: white;
    height: 5px;
    width: 20px;
    display: block;
    border-radius: 5px;
    box-shadow: 0px 10px 0px white,
                0px -10px 0px white;
}
```



Exercise 2

1. Design a responsive web page



Exercise 2

2. Add the hamburger icon to the mobile website
    - Hint: You can use display: none to hide navigation from the desktop version!
3. Add a transition to the menu so that it slides in/out