CSY1018
Web Development
Topic 4

Tom Butler
thomas.butler@northampton.ac.uk

Topic 4

- HTML background & HTML5
- Media queries (responsive web design)
- “Hamburger” menu icon
- :target selector

Web Background

- The World Wide Web was created by Tim Berners-Lee in the late early 1990s for sharing academic documents. By the mid-late 1990s it had evolved more closely to what we know it as today: A tool for accessing any kind of information
- When HTML was invented it was very limited. There were tags (now defunct) for things like bold and italic text
  - These kind of things are now handled by CSS!
- Over the years, as the needs of the web grew, new web browsers came along

Web Browsers

- By the mid 90s, there were different competing web browsers (Internet Explorer, Netscape Navigator, among others)
- Each web browser developer added their own tags and supported different technologies.
- This led to a situation where websites normally only worked correctly in one particular browser

Standardisation

- As the web grew, and different browsers started competing with each other, it caused a problem for web developers.
- There was no officially accepted standard for the tags or what they meant
- When pages started becoming more complex and dynamic different browsers started introducing their own tags
- Netscape Navigator introduced the <layer> tag as a generic container, while Internet Explorer introduced the (“division”) tag for a similar purpose

Standardisation

- This made life very difficult for anyone who wanted to develop a website
- And for users
  - Some sites would work some would not!
- As such, developers started building websites specifically for Internet Explorer and users had to use it as nothing else worked correctly.
  - By 2004, Internet Explorer had a 95% market share!

Standardisation

- By 1995, the problems with the lack of any single definition of “HTML” and what was/wasn’t valid code had been realised and Tim Berners-Lee (who created HTML several years earlier) founded an organisation called the W3C (WWW Consortium)
- This organisation set the standards of the HTML language and published specifications for how valid HTML should be formatted, what specific tags should be used for and what those tags should do
- It was then up to browser developers to follow this specification

HTML

- The HTML standard went through 5 major iterations, each one introducing new tags and technologies
- Because browsers had to handle websites that had been published when an older version was the newest, a special tag, the <!DOCTYPE> tag was introduced
- This was to tell the browser which version of HTML the current page is using and it could treat the code differently if it was on an older version.

HTML5

- We are currently on HTML version 5. This version was introduced relatively recently in 2012
- At the time, video and audio multimedia were becoming increasingly common on the web
- Until HTML 5 came along, dated (even at the time) technologies like ~~Macromedia~~ Adobe Flash had to be used to embed video on a web page
- The user had to have Flash installed on their PC or the video on the page would not work, and Flash introduced a lot of privacy and security issues

HTML5

- In addition to better multimedia capabilities, HTML set out to improve the overall structure of web pages
  - For our purposes, this means some new tags available for use on our web pages which weren’t available in the past

HTML 5

- You’ve already seen some of them (header, main, footer, aside…)
- Before these tags were added, there was only one generic container element `<div>`
- `div` is short for “division”

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

- 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 2020
  </div>
```

HTML 5

- Because most websites were using the same kind of CSS classes for the same kinds of tasks, the W3C added standardised tags that could be used by everyone.
- Previously some sites had variation:
  -
  -
  -
  - Etc
- This made it difficult for search engines, screen readers and other tools to work out which part of the page was which.
- HTML5 standardised tags for these kind of things

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

- This was designed to replace using elements for everything
- 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>

What not to do

- When developing a web page do not do any of the following:
  - Use <table> elements for layouts (this was already bad in 1999!)
  - Use any of the outdated styling tags:
    - <u>underline</u>
    - bold
    - italic
    - for styling fonts
  - Use any of the positioning elements:
    - for centreing text
    -
 for line breaks (use paragraphs!
    - for aligning text
    - <frame> elements for embedding one page in another (<iframe> should be avoided too!)
  - **These things have all been bad practice since the late 1990s! Use CSS to achieve them instead!**
  - **Doing these on the assignment will likely result in a fail grade!**

HTML5 and the mobile era

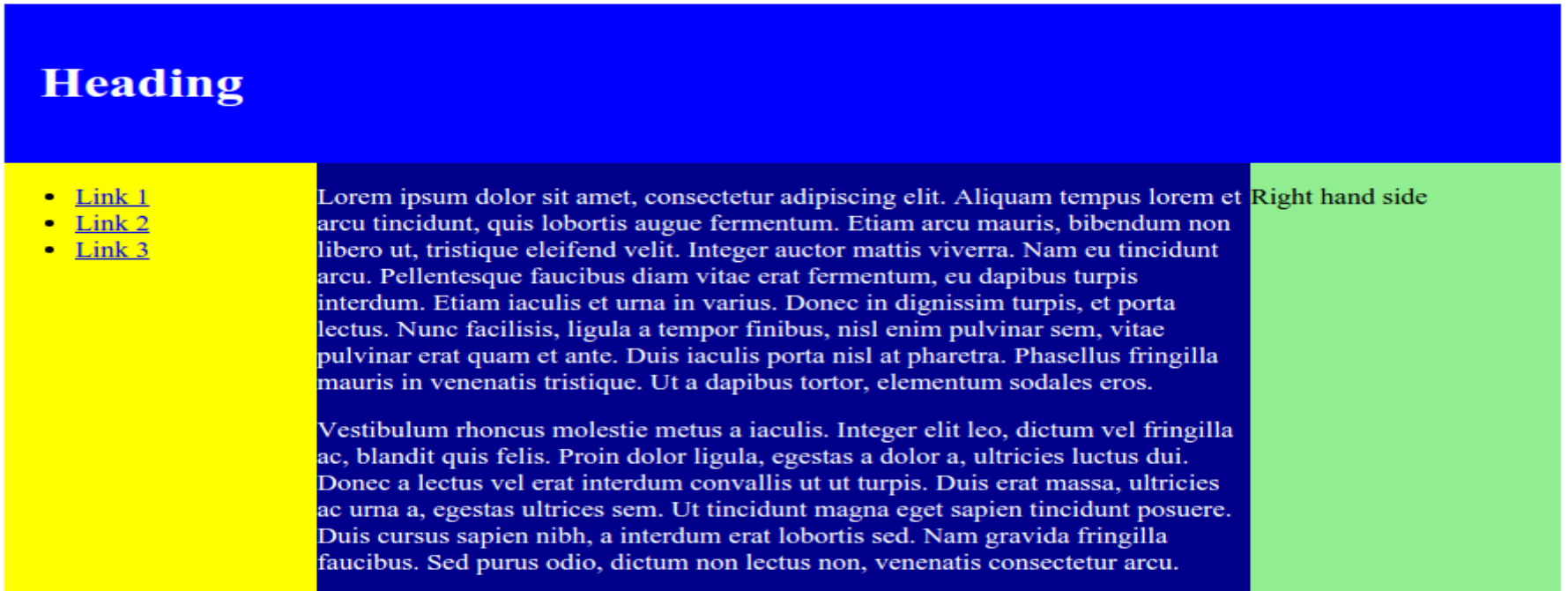
- When the web was created, and for most of its existence, everyone was using a desktop PC to access the internet
- Although different screen sizes (and resolutions) have always existed, websites could, in the past, assume a reasonable minimum width.
  - From 2001 or so onward, websites could assume a screen width of at least 1000 or so pixels (1024x768 was the most common screen resolution in the early 2000s)
- This allowed developers to build sites that could have sidebars, make use of horizontal space in headings for navigation bars, etc.

Smartphones

- By the early 2010s, everyone started getting smart phones and used them to browse the internet.
- Developers could not longer assume that they could display 5-6 links across the top of their website
- In the early days of smartphones, websites that wanted to support phones had two versions of the site:
  - www.website.com for desktops
  - m.website.com for phones
- **Don’t do this now! It’s a very poor solution!**

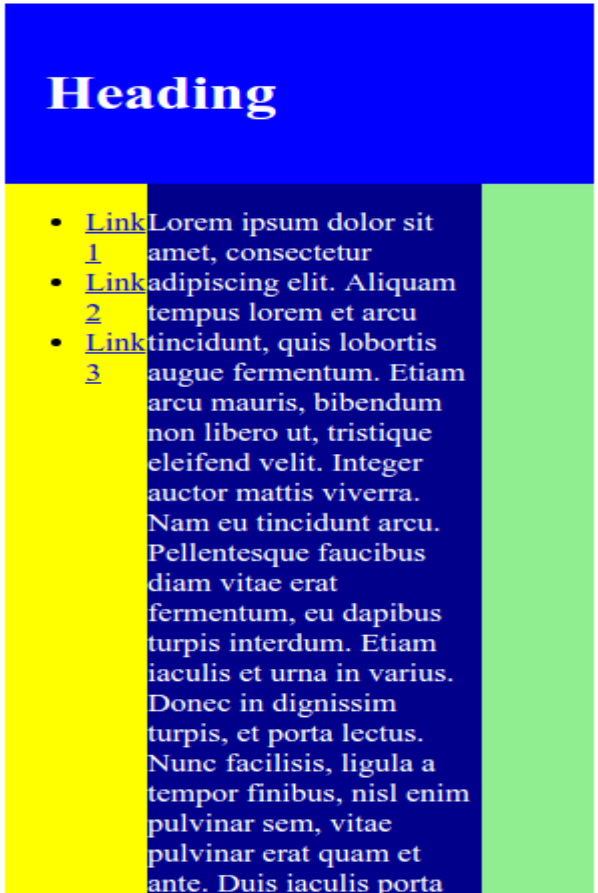
Web Layouts

- Last week you used display: grid to create a layout that looked something like this:
-



Mobile sites

- On a mobile it looks like this:



Everything is squashed,
the links are illegible

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 1000px wide
- 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
- This tells the browser to use the device width as the scale. You can optionally scale the content above or below 1 but don’t do this unless you have a good reason!

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)" />
```

- You can combine conditions to apply stylesheets to a width 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 (min-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" />
```

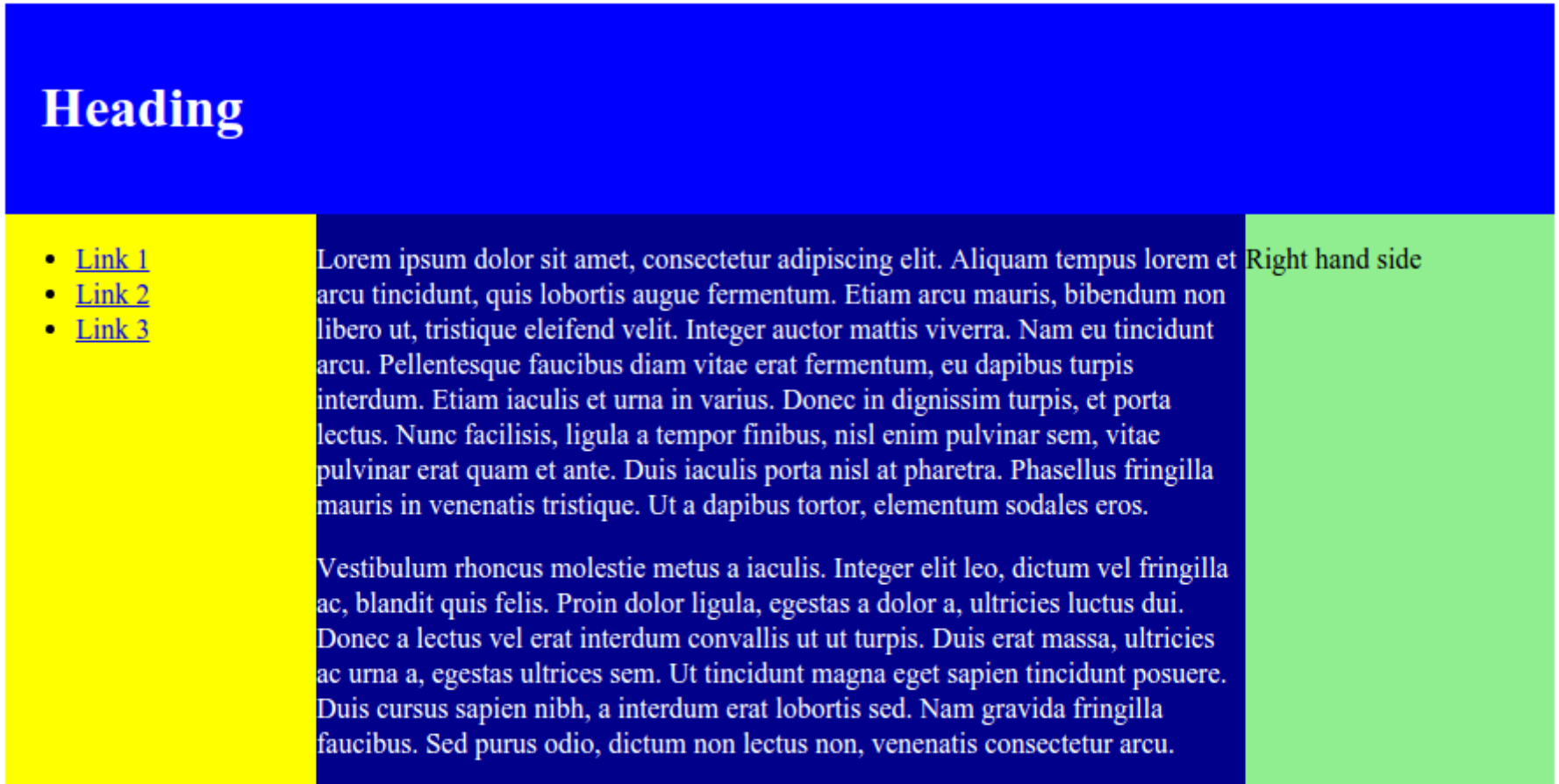
```
h1 {
        color: red;
}
```

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

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

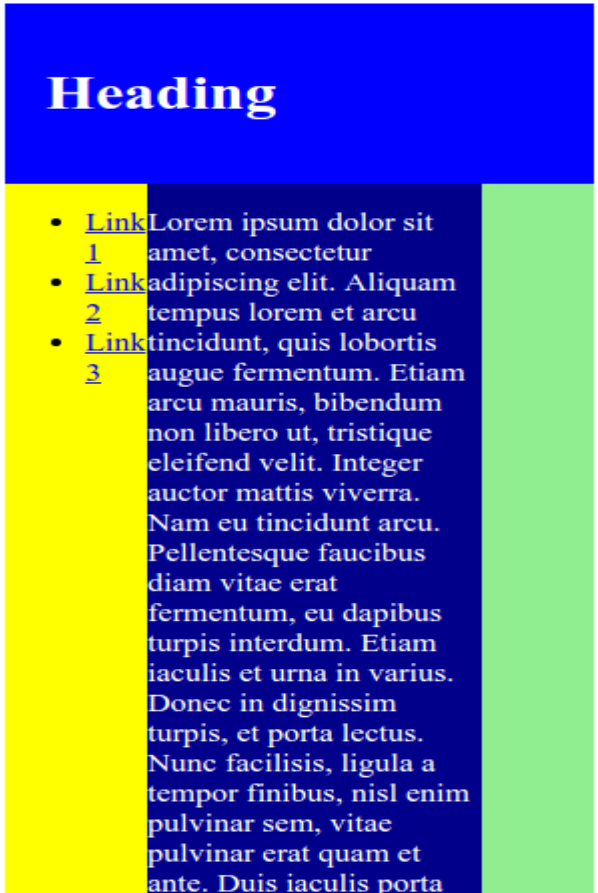
Mobile sites

- This site is not mobile friendly



Mobile sites

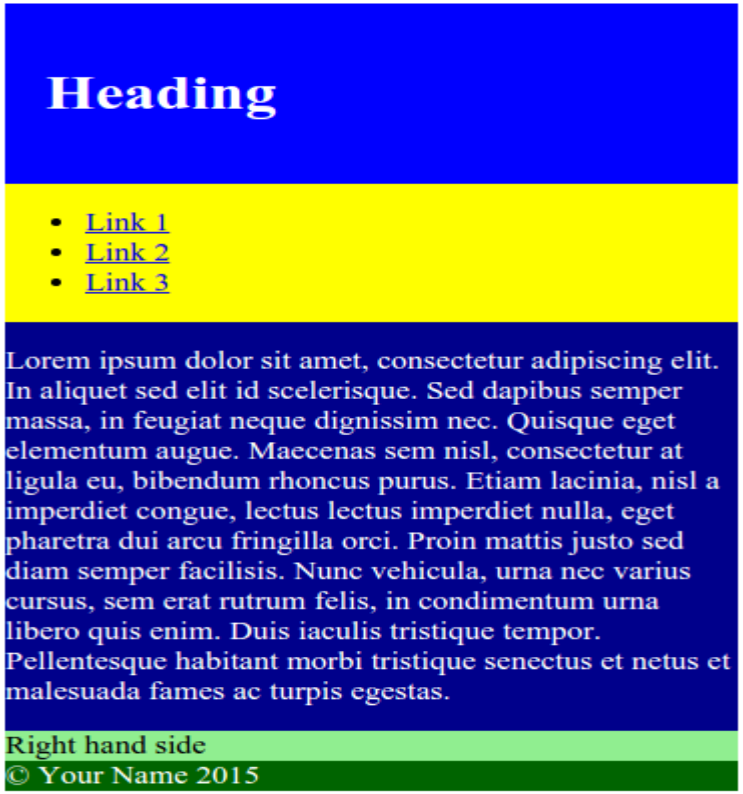
- On a mobile it looks like this:



Everything is squashed,
the links are illegible

Mobile site

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



Today's exercise!

“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) Give it a top and bottom border and a height

```
.shownav {
    height: 0.8em;
    width: 1.25em;
    display: block;
    border-top: 0.2em solid white;
    border-bottom: 0.2em solid white;
}
```

Create a Hamburger Icon

- 3) Add border-radius: 0.1em if you want it curved

```
.shownav {
    height: 0.8em;
    width: 1.25em;
    display: block;
    border-top: 0.2em solid white;
    border-bottom: 0.2em solid white;
    border-radius: 0.1em;
}
```

Hamburger icon

CSS supports :before to draw an extra element (using CSS, not HTML!) before the targetted element.

- This can also be used to draw an extra element *on top* of an existing element.

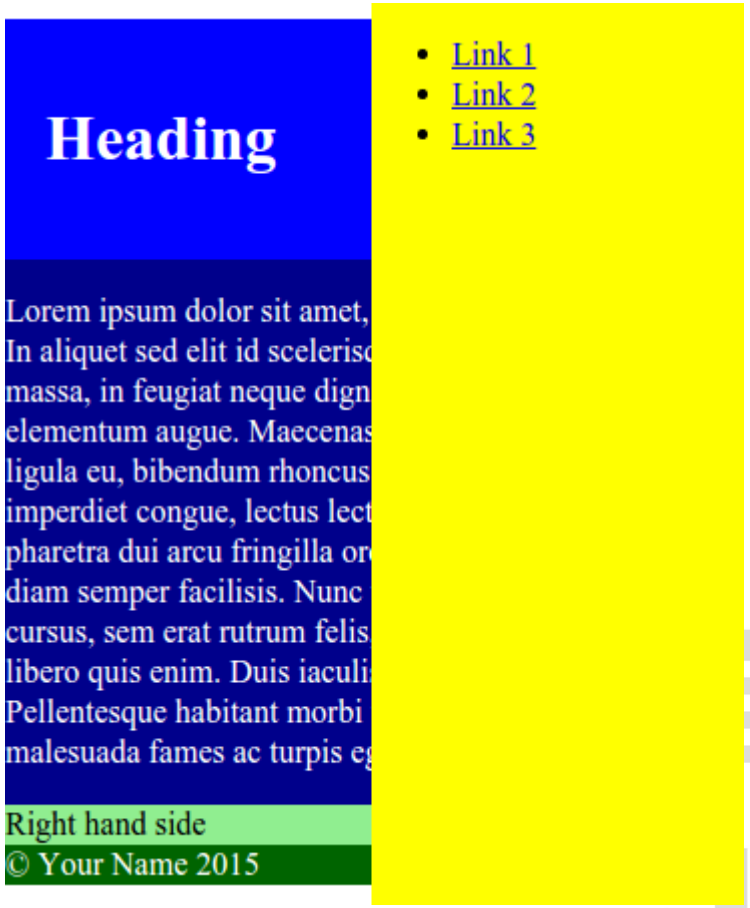
```
.shownav:before {
    content: '';
    position: absolute;
    width: 100%;
    top: 0.3em;
    left: 0px;
    border-top: 0.2em solid #fff;
}
```

Moving the menu

- Instead of having the menu visible on the page all the time, it's possible to move it to use the hamburger icon to toggle it. To do this, you can use the following CSS:
  - position: fixed which lets you position it on the screen using x/y co-ordinates
  - Setting top to 0
  - Setting left to 50% so it covers half the screen
  - Setting height to 100vh (100 vertical height)
  - Setting the width to 50% so it covers half the screen

Mobile menus

```
nav {
    display: block;
    background-color: yellow;
    position: fixed;
    top: 0;
    left: 50%;
    width: 50%;
    height: 100vh;
}
```



:target selector

- The CSS :target selector is a special CSS selector that's very useful for links:
  - It allows you to apply CSS *after* a link has been clicked

Anchors

- Before using the :target element you must define an *anchor*
- This is an element on the page with an ID
- You can link to this ID, and when the link is clicked the browser will scroll to the element
- To link to an element use Click me

Anchors

```
<!DOCTYPE html>
<html>
  <head>
    <title>anchor example</title>
    <link rel="stylesheet" href="demo2.css" />
  </head>

  <body>

    <a href="#clicked">Click me</a>

    <div id="clicked">
      Link has been clicked
    </div>

  </body>

</html>
```

Anchors

- When a link to an anchor is clicked, it will do two things:
- 1) Scroll down to the element (or scroll to the top if the element is at the top of the page)
- 2) Append the anchor name to the URI

:target

- The CSS :target selector allows you to apply styles to an element only when it has been targeted (e.g. when #clicked is present in the URI)
-

```
<!DOCTYPE html>
<html>
  <head>
    <title>anchor example</title>
    <link rel="stylesheet" href="demo3.css" />
  </head>

  <body id="clicked">
    <a href="#clicked">Click me</a>
  </body>

</html>
```

```
/* Sets the background to red when the body
 isn't targeted */
body {
    background-color: red;
}

/* Sets the background to green when the body
 is targeted */
body:target {
    background-color: green;
}
```

:target

- You can use CSS to select elements inside the targeted element
- This is useful if you want to have a toggle button
- Add two links:
  - One that sets the target
  - One that clears the target
- Hide one link at a time so only one is visible

:target

```html
<!DOCTYPE html>
<html>
  <head>
    <title>:target example</title>
    <link rel="stylesheet" href="demo4.css" />
  </head>

  <body id="clicked">
    <a href="#clicked" class="on">Target on</a>
    <a href="#" class="off">Target off</a>
  </body>

</html>
```

```css
/*
When the target isn't set, hide the "off"
button and show the "on" button
*/
#clicked .on {display: block;}
#clicked .off {display: none;}

/*
When the target is set, hide the "on"
button and show the "off" button
*/
#clicked:target .on {display: none;}
#clicked:target .off {display: block;}
```

:target

- :target is a very useful way of toggling on/off the menu on a mobile site
- Using :target, position: fixed and the hamburger icon allows you to create a mobile menu purely in HTML/CSS

Exercise 1

- 1. Make your page from last week mobile friendly by applying a different stylesheet when it is viewed on a narrower device (recommended 1000px+ for desktop, < 1000px for mobile layout)
  - Hint: You can re-arrange the grid for the mobile site!
- 2. Add a hamburger icon
- 3. Hide the menu
- 4. Make it so clicking the hamburger icon toggles the menu
- 5. Remember to validate your CSS!