Lesson 8: Semantic Web & Fonts

Introduction

The Internet wants to know all about the web-pages you make. Google and other search engines use scrapers, spiders and bots to find, index, and try to make sense of the content you have on your web-pages. The Semantic Web (*semantic* implying meaning or understanding) is all about connecting your web-page information with other web-pages. In other words, it enables data to be linked to perform tasks on our behalf and is concerned with the meaning, and not the structure, of the data. To enable this linking of information requires labeled elements. Therefore, how we label the elements on our web-pages is very important.

Additional topics in the lesson include fonts, text attributes, and measurements. Fonts are an essential component of any web-page. We are going to look at the ways you can affect the fonts on your web-pages.

Learning Outcomes

Topics include:

- Semantic Web
- Creating a Landing Page
- Font manipulation
- Text attributes
- Measurements

Topic 1: Semantic Web

We use **divs** a lot when developing web-pages. Content is placed in a **div**, and then gives it an **id** of ***nav-bar*** and allows the designer to style away. However, how we use the **div** has an impact on the web as a whole.

Labeling **div** elements is haphazard at best. We could label a **div** with an **id** of **nav-bar, myAmazingBar, barbarbar,** whatever you want. The element might have nothing to do with navigation, it could be an image of a chocolate bar. This makes understanding the information on web-pages very difficult for all the robots crawling around.

To help, the **W3C** thought it would be great if we didn't use **divs** for everything and instead used more semantically relevant tags. For example, instead of using: **<div *id* = "navigation">,** simply use **<nav>.** This is the basic principle behind the new semantic tags.

<figure> and <figcaption>

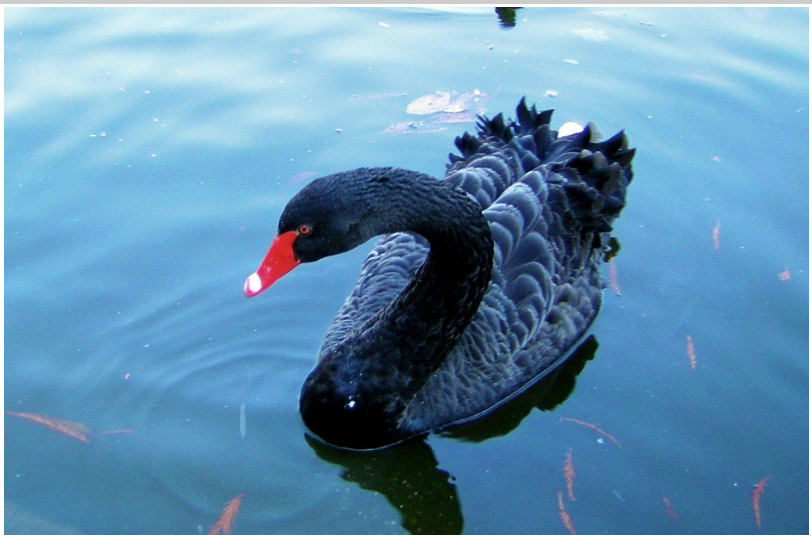
In traditional printed material such as books and magazines, an image or chart, would be accompanied by a caption. Before now, we didn't have a way of semantically marking up this type of content directly in our HTML. Instead we resorted to CSS class names. HTML5 hopes to solve that problem by introducing the **<figure>** and **<figcaption>** elements. Let us explore further and look at examples of the new semantic web <**figure>** and **<figcaption>** tags.

The <**figure>** element represents a self-contained unit of content usually in the form of a diagram, or an illustration of a figure referenced in the main text. A caption is usually optional. While this content is related to the main flow of the page, it can be moved to another section without affecting the main flow of the web-page. You can also use **<figure>** for audio, video, charts, poems, or tables of statistics.

```
<figure>
  <img src="/myImage.jpg" alt="My example of an image">
</figure>
```

The **<figcaption>** element is optional and can appear before or after the content within the **<figure>.**

```
<figure>
  <img src="/myCaptionImage.jpg" alt="My image with a caption">
  <figcaption>My image with a caption.</figcaption>
</figure>
```



My image with a caption.

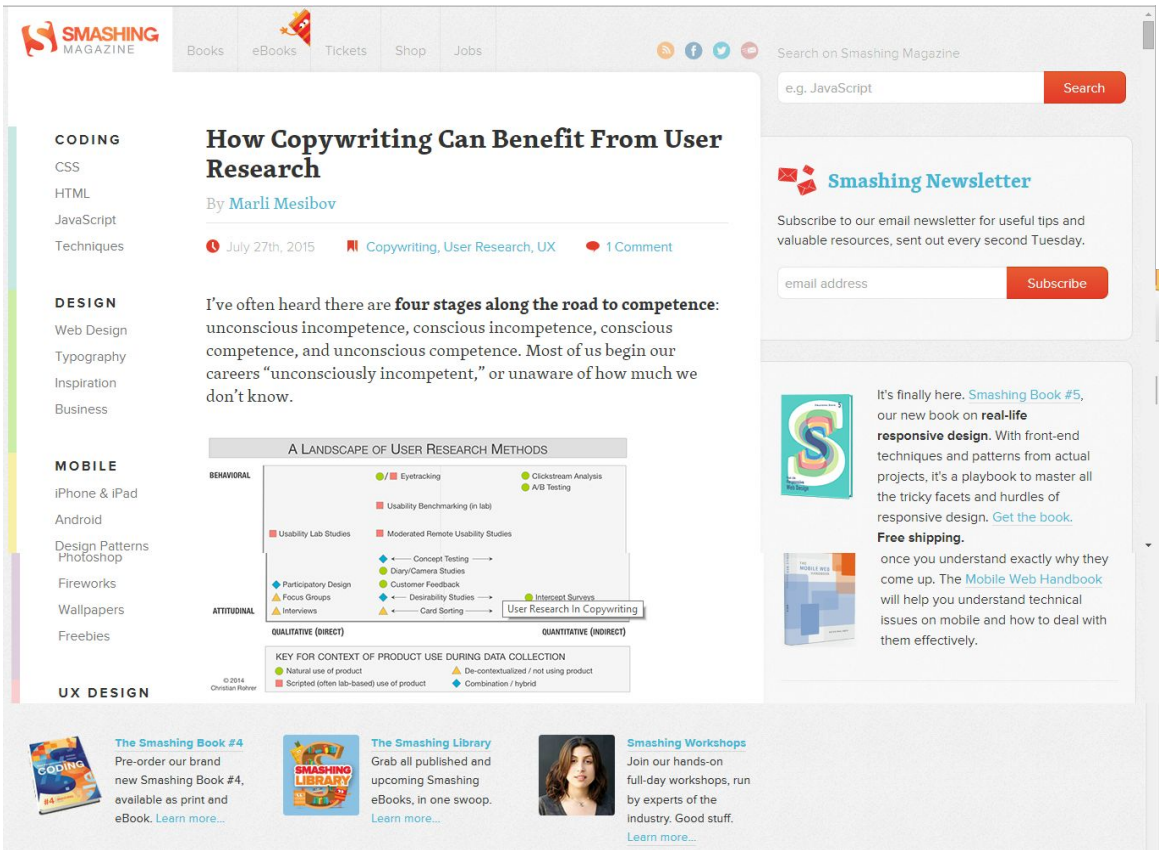
HTML5 Document Structure

Most sites follow a rough structure and include at least some of the following:

- **Header** at the top of the page containing the heading or logo.
- The **main content** which is the principle reason for the site.
- A number of **sidebars**, with content relevant to the page's main content or the site as a whole.
- **Navigation** across the top or down the side of the page.
- A **footer** across the bottom with copyright info and contact details.

Let's take the example of Smashing Magazine website:

Let's see what the front page of the online version of Smashing Magazine looks like. We will then markup the page with the old **div** approach and then again using the new HTML5 semantic tags.



Traditional document structure

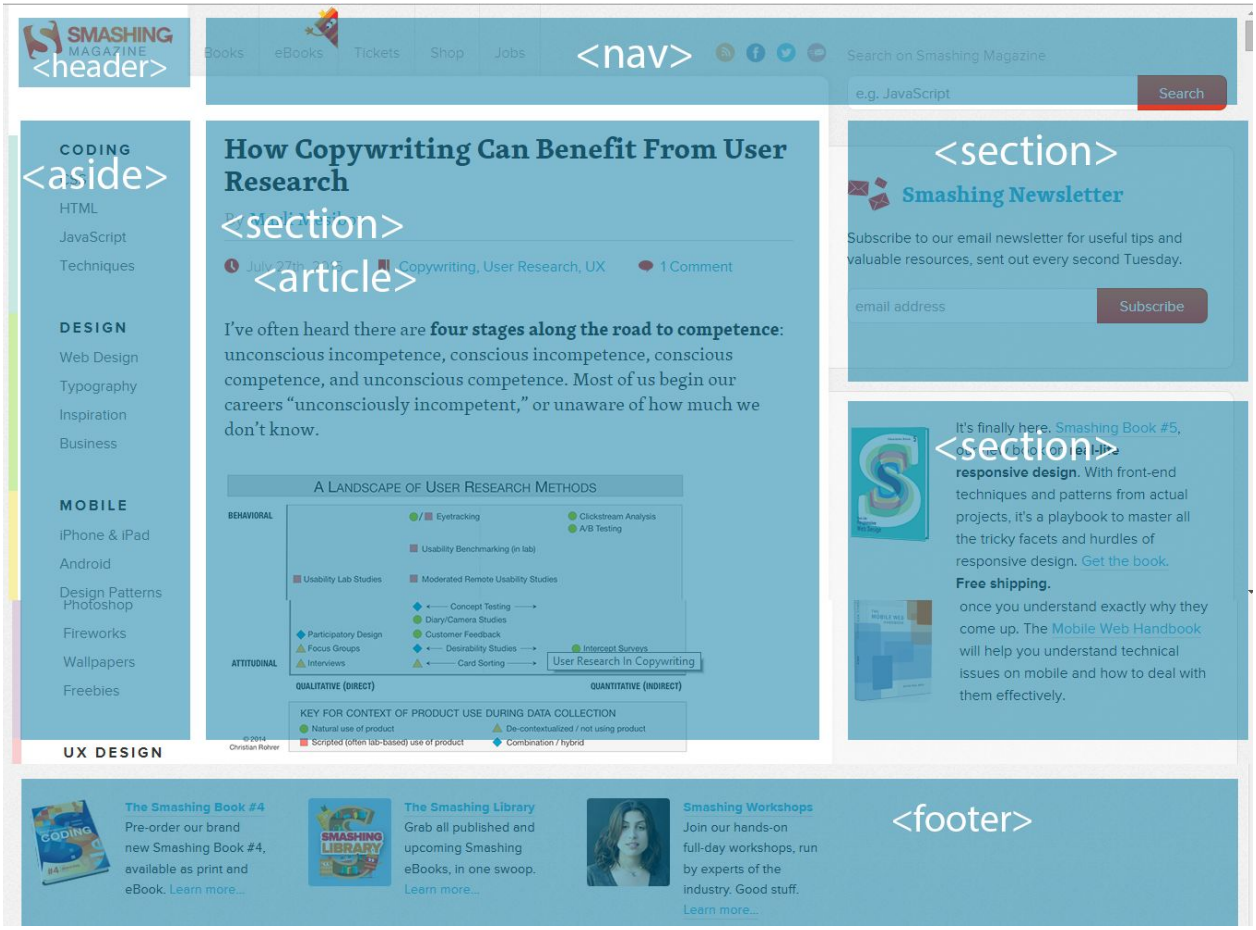
Before HTML5 we would have used the traditional **divs** to divide up the web-page similar to what is shown below:


```
1.  <body>
2.     <div id="header">
3.       <!-- header content goes in here -->
4.     </div>
5.
6.     <div id="navbar">
7.       <!-- navigation menu goes in here -->
8.     </div>
9.
10.    <div id="side">
11.      <!-- sidebar content goes in here -->
12.    </div>
13.
```

```
    <div id="articles">
      <!-- main page content goes in here -->
    </div>

    <div id="ads">
      <!-- sidebar content goes in here -->
    </div>

    <div id="footer">
      <!-- footer content goes in here -->
    </div>

  </body>
```

HTML5 with semantic tags

With HTML5 we can now use semantic tags to divide up the web-page similar to what is shown below:




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

      <section>
          <article>
          </article>

          <article>
          </article>
      </section>
      <footer>
      </footer>
</body>
```

<nav>

The **<nav>** element is a section of a page that links to other pages, or to parts within the page itself. **<nav>** is used for a group of navigational links and this is how the navigation bar was traditionally marked up:

```
<div id="nav">
  <ul>
    <li><a href="#">home</a></li>
    <li><a href="#">blog</a></li>
    <li><a href="#">portfolio</a></li>
    <li><a href="#">about</a></li>
  </ul>
</div>
```

And below we have an example of the new semantic web markup:

```
<nav>
  <ul>
    <li><a href="#">home</a></li>
    <li><a href="#">blog</a></li>
    <li><a href="#">portfolio</a></li>
    <li><a href="#">about</a></li>
  </ul>
</nav>
```

<section>

The **<section>** element is used for containing clear subject areas. It can also be used to break up articles into different sections. Examples of sections would be chapters, or the numbered sections of a thesis. The Home Page could be split into sections for an introduction, news items, contact information etc.

<article>

Where **<section>** is used for grouping sections of content or functionality, **<article>** is for standalone content, such as individual blog posts or news items of

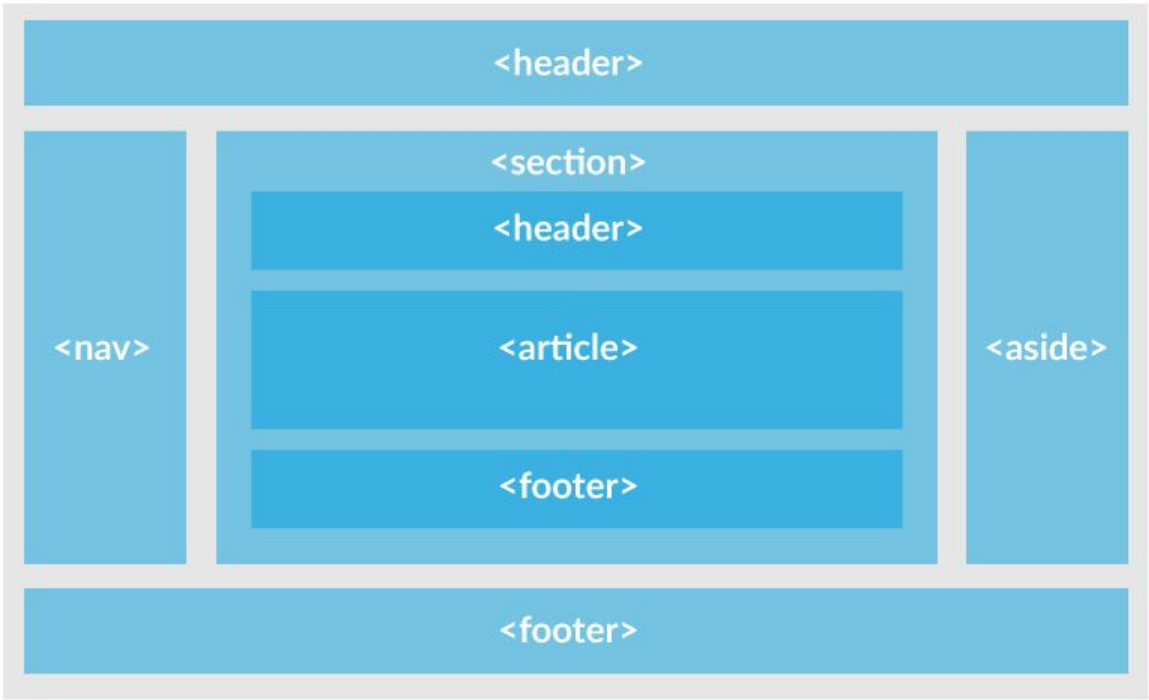
articles! Think of a magazine article, that, if removed from the magazine would still make complete sense to the reader.

```
<section id="main">
    <article>
        <!-- first blog post content goes here -->
    </article>
    <article>
        <!-- second blog post content goes here -->
    </article>
    <article>
        <!-- third blog post content goes here -->
    </article>
</section>
```

Straightforward ? Well think of this. We could have a **section** of our page which contain **articles** but these **articles** themselves may contain **sections (chapters)**. There are no hierarchical rules to say a **sections** are above **articles** in any way. It's all to do with the meaning, as in, the semantics!

See the layout example below. The **<article>** element in the middle of the layout contains the following code:

```
<article>
    <section id="intro">
    </section>
    <section id="content">
    </section>
    <section id="summary">
    </section>
</article>
```



<aside>

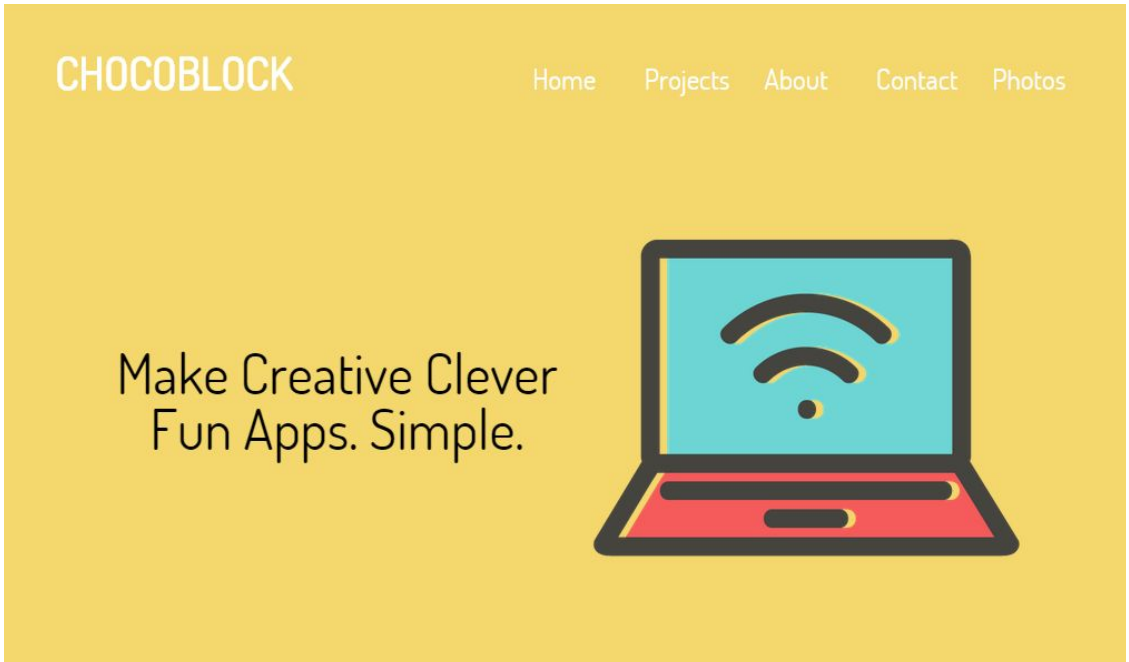
<aside> represents a section of a page that consists of content that is related but is considered separate from that content. They are often represented as sidebars. Examples for the use of <aside> could be notes about the author, or related blog links. In a magazines you often have info-boxes containing related information. These make perfect asides.

<footer>

The **<footer>** element is straightforward. We have all seen footers at the bottom of pages with copyright information, and often a set of links. These links within the footer are perfectly acceptable and do not need a **<nav>** element.

Challenge 1

We are going to continue from or last Challenge and create a mobile first **landing page**. We will use some of this lesson's topics as they come up.



The Prize

Open up the responsive navigation challenge from the last lesson

First a few design changes

(you can change the styling to whatever you wish but best wait until you are finished coding)

- Remove the grey background-color from the **<nav>**
- Give the **<body>** a background color of **#F5D76E**
- Change the colour of the text and border to **white (#fff)**
- Adjust the font size until you are satisfied
- Change the border on <a> for border-bottom only

reset.css

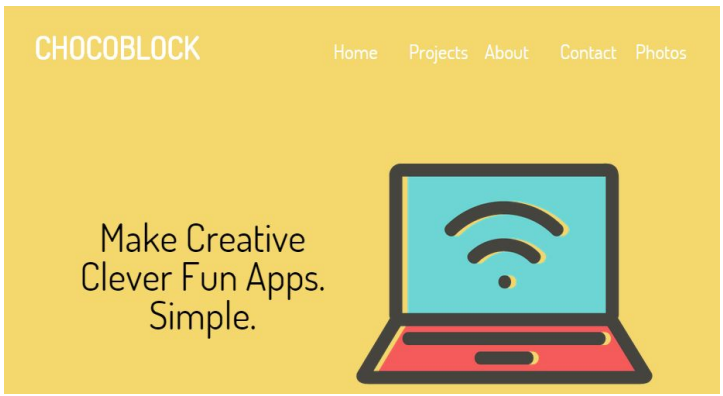
First we are going to include a reset.css file, this will alter some of the default fonts and margins, just resize them after to your preference.

- Download the **reset.css** file (in the lesson folder) and include it it the head of the document.

```
<link rel="stylesheet" type="text/css" href="reset.css">
```

The HTML

mobile



desktop

We are going to add a message/tagline and an image after the nav. In the desktop version we will align them side-by-side so we should take that into account. Flexbox will work well here. Let's call this section ***landing***. Inside a landing we will have two other divs.

- **landing**
  - **landing-message**
  - **landing-graphic**

- The landing message can be a <h2> and you can find an image in the lesson folder.
- In this case, the landing div will be the **flexbox container** . The **landing-message** and **landing-image** are the flex-items.
- Style the font as you wish, the **Dosis** font-family was used in the example

- Position the flex-items using **justify-content** and **align-items**.

Media Query

- The desktop version should have the landing-message and landing-image side-by-side. Use Flexbox for this.
- You may want to adjust the landing div.Use margins to center and give it space where it needs.

Topic 2: Fonts

Fonts are one of the most important parts of building a web-page. Font design is a big topic in print design and this carries over to web design. Designers can spend considerable resources and finances in designing fonts and purchasing licensed fonts.

CSS can define the properties of fonts by giving you the ability to define the **font-family**, **size, style, weight,** and **color** of a piece of text.

Font-Family

You can set what font will be displayed in an element with the **font-family** property. If for some reason the browser cannot load the prefered font you can list a number of fallback fonts. Begin with the font you want, and end with a generic family. If you wish to select more than one font-family, comma-separate them in a list. Use quotes around the name if it consists of more than one word e.g. 'Courier New'

```
font-family: lato, Verdana, sans-serif;
```

a set of default fonts can be seen at web fonts available . Note that fonts available are largely determined by the user's operating system which can cause incompatibilities when viewing a web site on a windows system and a mac os. The advantage of using default fonts is that they don't require downloading from the internet. But the downside is that they limit the design choice.

Font-Size

The **font-size** property sets the size of the text on your web-page. Having the ability to set the size of the text is important for the layout of your web-pages. However, always use the correct HTML tags to set up your headings (**<h1>**) and paragraphs (**<p>**) and do not use **font-size** to make paragraphs look like headings or vice versa.

```
1.      font-size: value;
```

There are a lot of choices for your **values** input:

- xx-large
- x-large
- larger
- large
- medium
- small
- smaller
- x-small
- xx-small
- px (pixels)
- % (percent)

Font-Style

You can set the style of text in a element with the **font-style** property.

```
font-style: italic;
```

Possible values are:

- normal
- italic
- oblique

Font-Weight

To control the weight or boldness of the text, use the **font-weight** property.

```
font-weight: bold;
```

Possible values are:

- lighter
- normal
- 100 - 900 (400 is normal)
- bold
- bolder

Color

The **color** property sets the text color of different elements in the web-page.

```
1.      color: #0000ff;
```

Possible values are:

- Color name (red, black…)
- Hex number (#ff0000, #000000)
- RGB color code (rgb(255, 0, 0), rgb(0, 0, 0))

@Font-Face

The **@font-face** rule allows us to load custom fonts on a web-page to display text. Once included in a stylesheet, the browser is instructed to download the font from where it is hosted, then display it as specified by the individual CSS styles. This eliminates the need for the browser to depend on the limited number of fonts that individual users will have installed on their own computers. The **@font-face** rule should be added to the CSS stylesheet before any styles are added.

```
@font-face {
  font-family: 'MyWebFont';
  src: url('myfont.woff2') format('woff2'),
  url('myfont.woff') format('woff'),
  url('myfont.ttf') format('truetype');
}
```

Then use it to style elements like this:

```
body {
  font-family: 'MyWebFont', Fallback, sans-serif;
}
```

Fonts like other assets come in different formats to support the wide range of browsers. The above formats are compatible with most modern browsers including IE9+ . More on font formats .

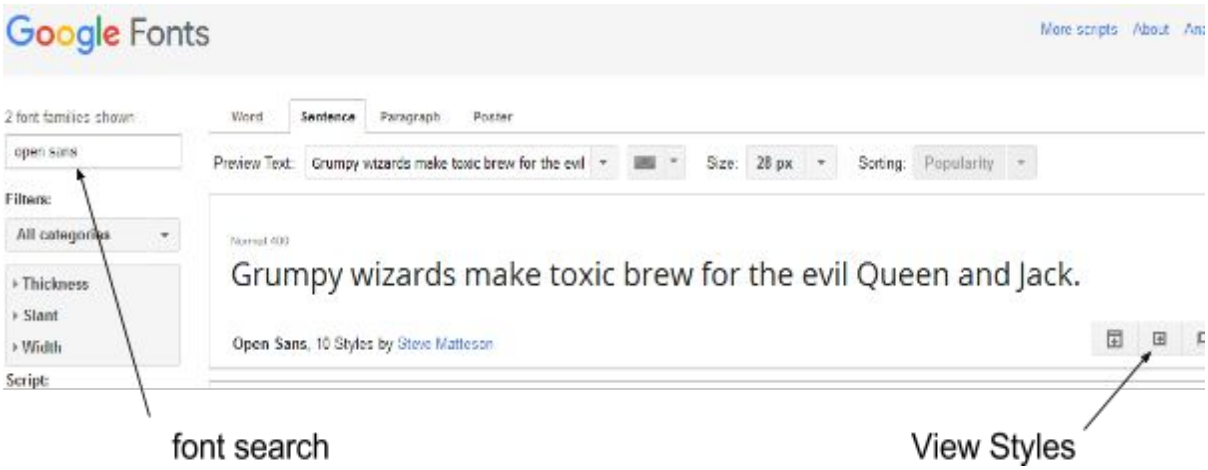
Google Fonts

Using @font-face is a good way of including additional and alternative font choices, but it is complicated. It involves finding suitable fonts that are licence free (if we don't want the expense), installing them and finding them in the correct format.

An alternative is to use Google Fonts. It actually uses @font-face behind the scenes but the fonts are hosted on their servers and makes the process of including new fonts a lot easier.

Google Fonts contains a whole bunch of fonts to choose from which are open source.

The following is an example of using loading the Open Sans font from [Google Fonts]():



We create a link to the 'Open Sans' font from google and include like any other css stylesheet

```
<link href='//fonts.googleapis.com/css?family=Open+Sans'
rel='stylesheet' type='text/css'>
```

Then, we can style our elements like the other methods:

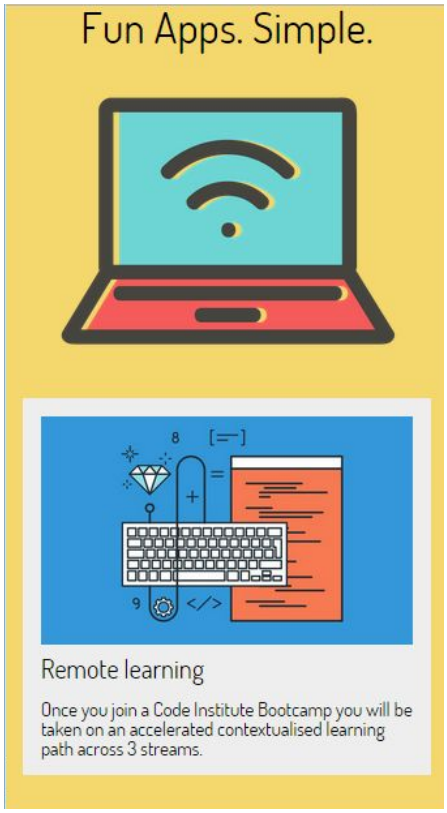
```
body {
  font-family: 'Open Sans', sans-serif;
}
```

To explore how to use Google Fonts spend some time at [how to get started]() .

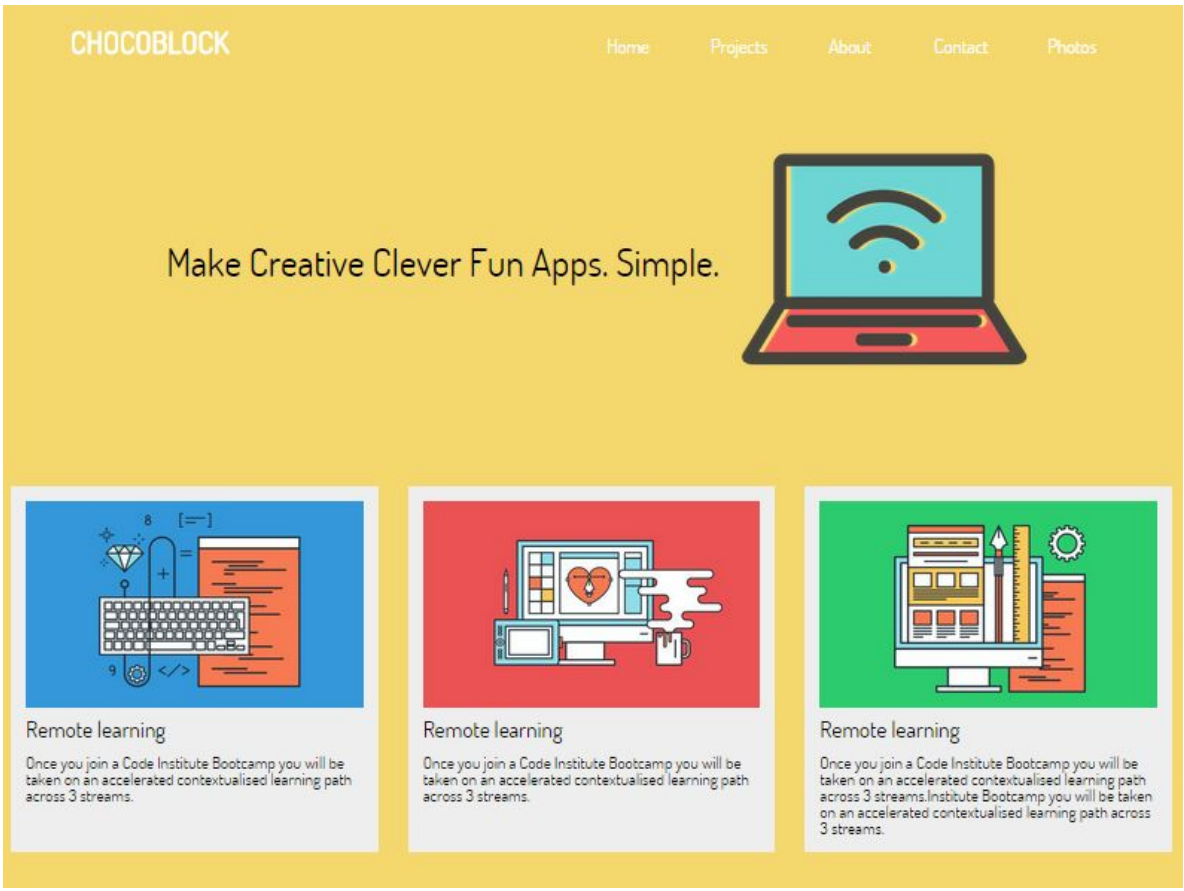
Search for the ‘dosis’ and ‘Roboto’ fonts - can you find them ?

Challenge 2

Continuing from the previous Challenge , we will add a series of design element in the form of cards. Each will have an image, heading and paragraph.



mobile



desktop

The HTML for a single **card** is as follows(images in lesson folder):

```
<section class="card-container">
        <div class="card">
             <img src="img/graphic1.png" alt="design pic">
             <h2>Web Info Sessions</h2>
             <p>If you have questions about your career in...</p>
        </div>
</section>
```

Firstly, we should create a mobile version of the cards using CSS.

Below is a shell for the CSS classes. The challenge is to replace ***code here*** with CSS rules:

```
.card-container{
        /*code here*/
}
.card{
        /*code here*/
}
.card h2{
        /*code here*/
}
.card img{
        width:100%;
        }
.card p{
        font-family: dosis;
        font-size: 22px;
}
```

Hint: **.card-container** is a perfect flexbox container

- Add two more cards.

Topic 3: Text Attributes

Text attributes are different to **font** attributes. They include properties such as **alignment** and **indentation**. Let us have a look at some of the differences.

Text-Align

You can horizontally align text with the following syntax:

```
text-align: value;
```

Possible values are:

- left
- right
- center
- justify

Text-Decoration

You can decorate text with the following syntax:

```
text-decoration: value;
```

Possible values are:

- none
- underline
- line through

Examples:

This text is underlined.

~~This text has a line through it.~~

Note:

The **<a>** tags have **text-decoration: underline** set by default. A common procedure is to set **text-decoration: none;** in the code.

Text-Indent

You can specify how much you would like to indent the first line of text, by using the following syntax:

```
text-indent: 20px;
```

Text -Transform

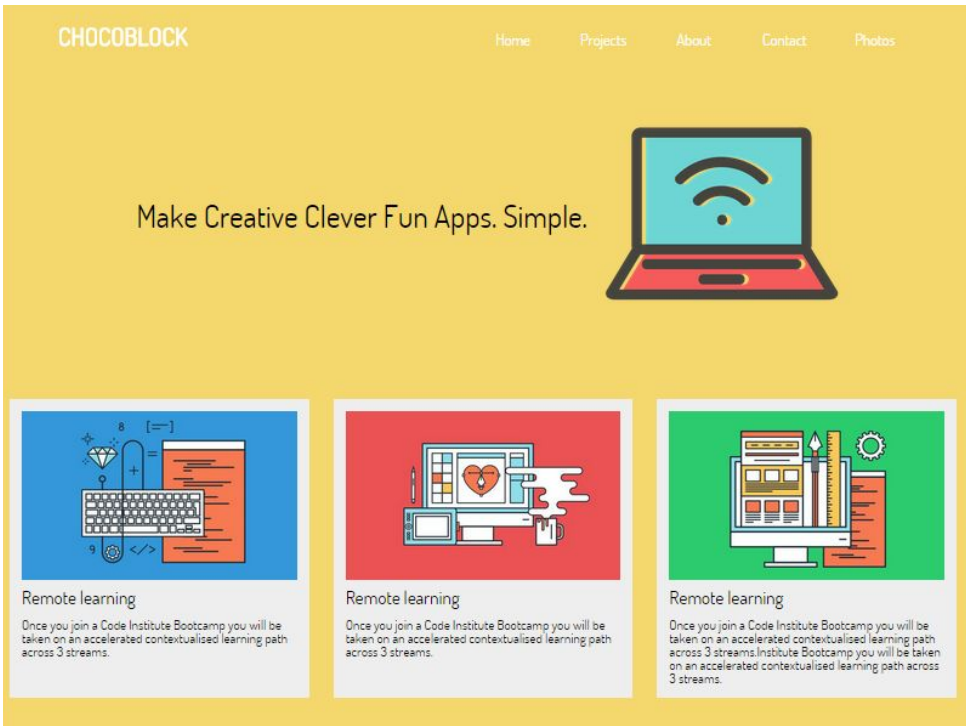
You can control the case (uppercase or lowercase) of text by using the following syntax:

```
 text-transform: value;
```

Possible values are:

- none
- capitalise (capitalises the first letter of each word)
- lowercase
- uppercase

Challenge 3



Continuing from the previous challenge. Let's create the layout design for desktop. We can do this using the media query here.

We need to change a few flexbox properties.

```
@media screen and (min-width: 700px){
      ...

      .card-container{
            /* code here*/
      }
}
```

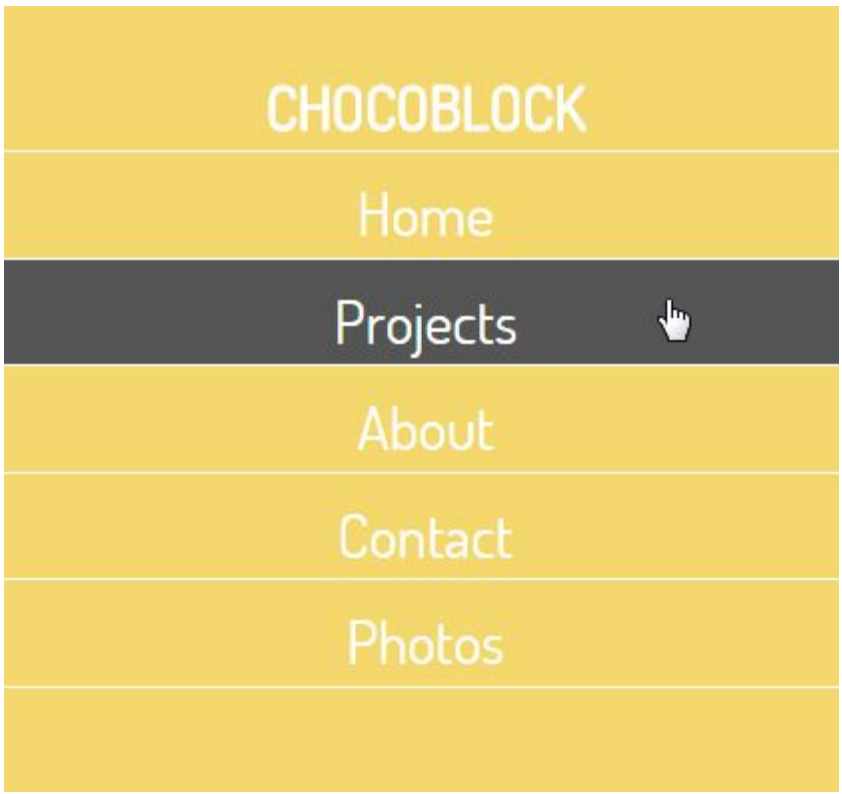
Hint: One way to get the layout involves using **flex-wrap**

If you got this far, well done, flexbox can be tricky to begin with but with a little practice it make laying out responsive pages quick and relatively painless.

Challenge 4

Part A)

- Firstly give your navbar a hover pseudo-class and change the properties for mobile only.

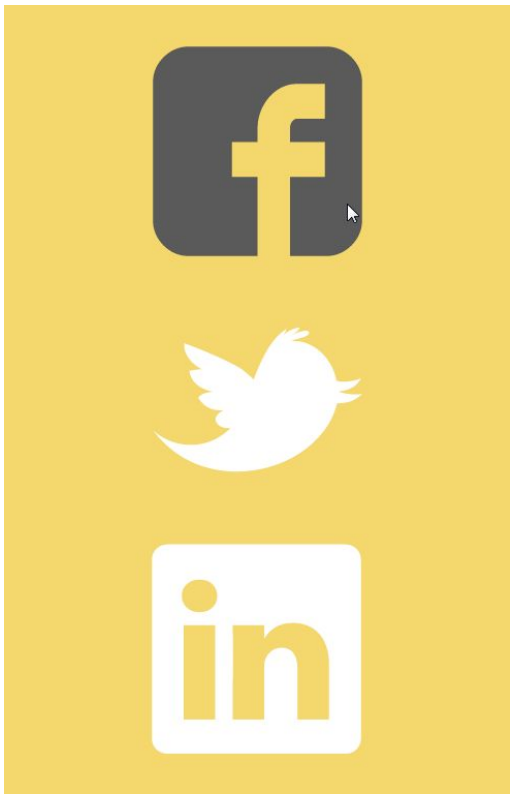


- Apply a transition to the nav items

Part B)

- Add a footer which contains social media icons (images in lesson folder).
- The images as svgs so they scale without loss of quality.

- Taking a mobile first approach and using flexbox create the following footer layouts (use semantic tag for the footer element)
- The original svgs are black in colour, we will change the colour later.



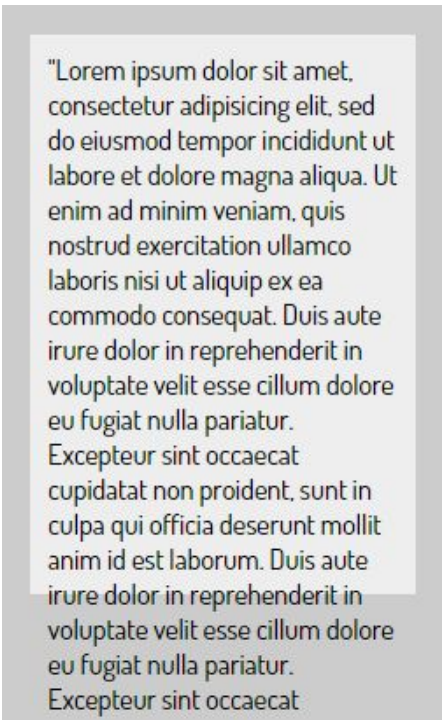
mobile

desktop

- Add a hover pseudo-class (use css filters to change the image colour) **Hint:** filter: invert() will work well
- Add a transition effect

Topic 4: Overflow

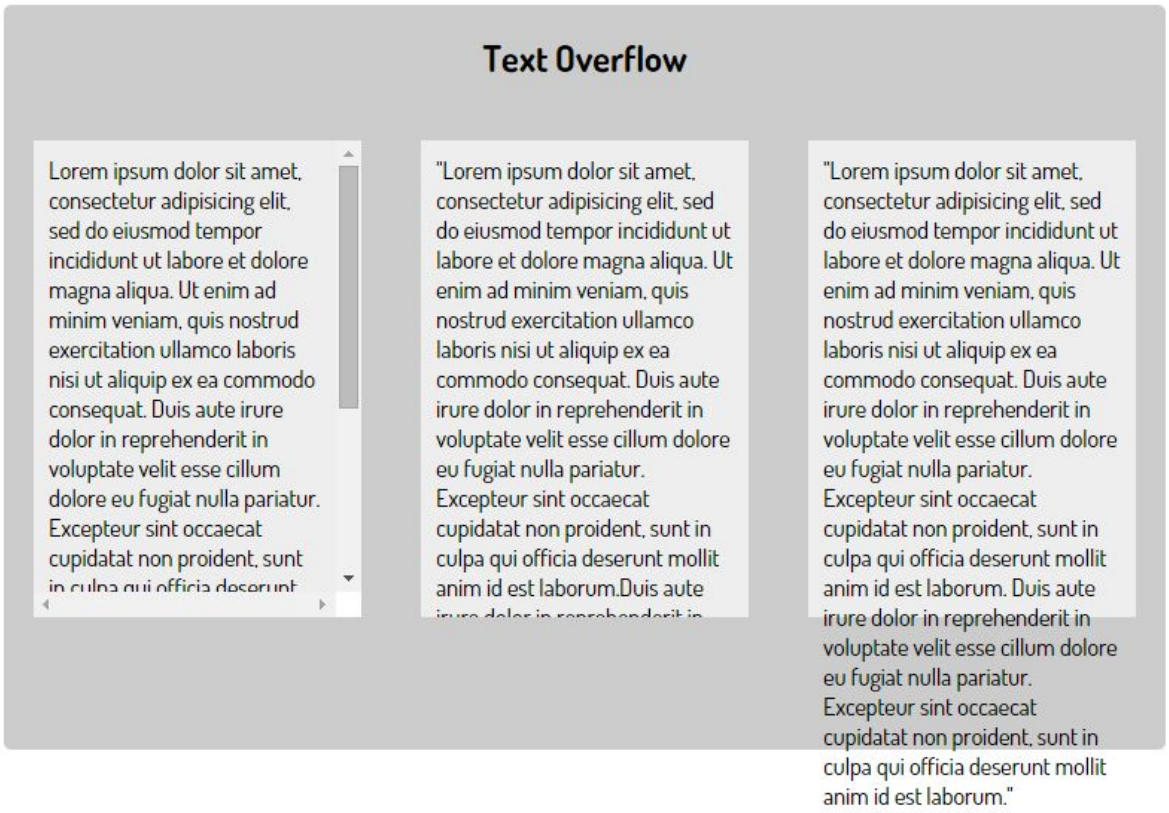
Overflow issues occur when your container has a fixed height but it's contents can no longer fit inside.



Flexbox deals with overflow by automatically increasing the height of the flex item. However what if your element is not using flexbox.

There are a few options we can consider:

1. **Scrolling overflow** creates a scrollbar to navigate the content.
2. **Hidden overflow** makes overflow invisible.
3. **Visible overflow** spills over the container.



overflow property

Values:

overflow:scroll;

overflow:visible;

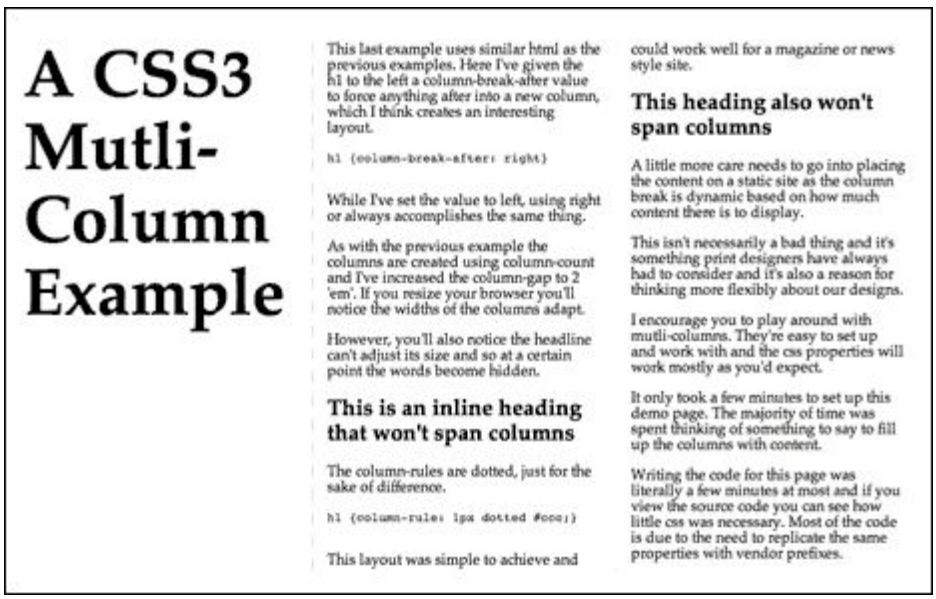
overflow:hidden;

```
#box1 {
  overflow: scroll;
}

#box2 {
  overflow: hidden;
}
```

```
   #box3 {
    overflow: visible;
}
```

Multi-column Layout

With a multi-column layout we can divide text across several columns. Replicates a newspaper layout.

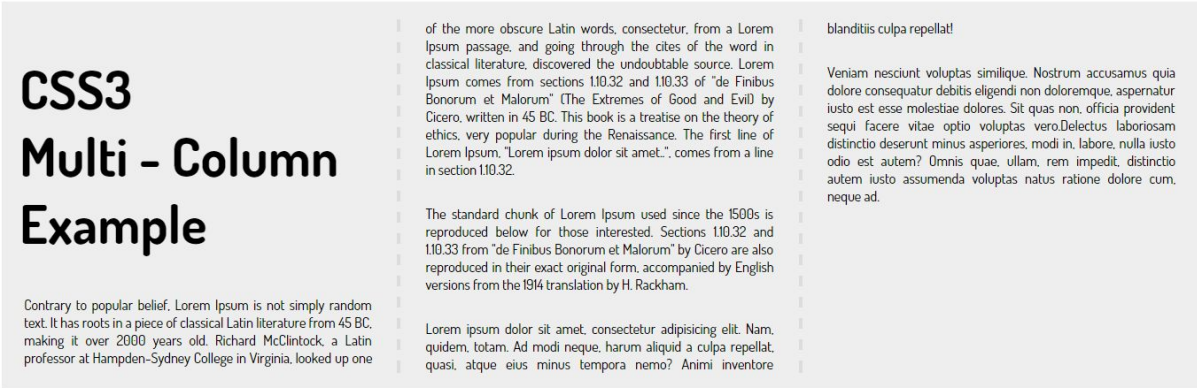


We can specify the amount of space that appears between columns (the gap) and create vertical lines (rules) between columns.

There are three CSS properties we can use to create this layout:

- **column-count**: Sets the number of columns.
- **column-gap:** Specifies the gap between the columns, known as the gutter or alley.

- **column-rule:** Creates a vertical line between columns and sets the width, style (single or double line, solid, dashed, etc.) and color.



Note:

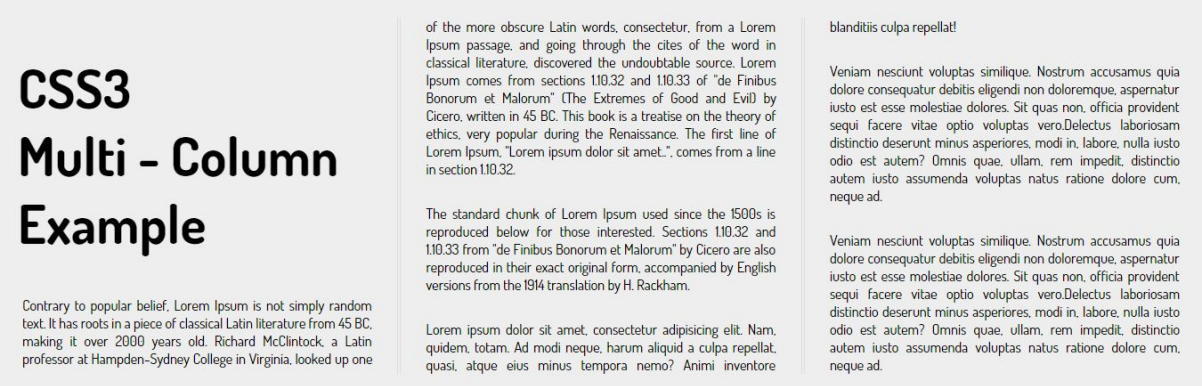
Multicolumn layouts are still not fully implemented and need vendor prefixes.

```
-webkit-column-count:3;
-webkit-column-gap:50px;
-webkit-column-rule:4px dashed #ddd;

-moz-column-count:3;
-moz-column-gap:50px;
-moz-column-rule:4px dashed #ddd;

column-count:3;
column-gap:50px;
column-rule:4px dashed #ddd;
```

Challenge 6

- Create a similar multi-column layout to the above and add it to your page.

- Use **overflow** to contain the contents



Topic 5: Measurements

CSS3 has (brace yourself) sixteen different units by which you can measure the size of text with **font-size** attributes. Luckily, many of them are remnants from the print world and are not widely used on the web.

There are two types of measurement units that you need to consider when using **font-size** attributes: **relative lengths** and **absolute lengths.**

- **Relative lengths:** These units can change their size according to the size of the screen. The relative units we will look at are: **percent(%)**, **em**, and **rem**. They are mainly useful when the output media has dynamic sizing like mobile devices, or tablets. Therefore, they have become very important for responsive design.
- **Absolute lengths:** These units are fixed in relation to each other and are anchored to a physical measurement. The most common unit is: **px**. They are mainly used when the output format is fixed.

As mentioned above, relative measurements are prefered if you wish your pages to dynamically respond to the device being used. Let's have a look at our choices:

- **%:** Percentage values are always relative to the parent container. They often used to size container elements.
- **em:** These values are relative to the font size of the element itself.
- **rem (root em):** These values are relative to the font size of the root element (For example, **<html>** is the root element of a HTML document)

em Examples:

If 1 em is set to 14 pixels:

- .5 em = 7 pixels
- 1.5 em = 21 pixels
- 2 em = 28 pixels

So what is best option for using font sizes, in practice? There is a lot of debate around this, but for now let us keep it as simple as possible. One relatively straightforward way is to declare the root element (**<html>**) in **pixels (px)** and then use **rem** throughout the page. This should be sufficient for most projects. Again, this is illustrated best with an example:

```
1.  html {
2.    font-size: 12px; /*declaring root element in pixels*/
3.  }
4.
5.  h1 {
6.    font-size: 2rem; /*displayed at 24px*/
7.  }
8.
9.  h2 {
10.   font-size: 1.5rem; /*displayed at 18px*/
11. }
12.
13. h3 {
14.       font-size: 1.25rem; /*displayed at 15px*/
15. }
16.
17. h4 {
18.       font-size: 1rem; /*displayed at 12px*/
19. }
```

Summary

The basic concept behind the Semantic Web is that it connects facts. Rather than linking to a specific document or application, you can instead refer to a specific piece of information contained in that document or application. If that information is ever updated, you can automatically take advantage of the update.

Creating a responsive landing page is a huge step in front-end development. Once known as the holy grail layout and it was so difficult to achieve in the past.

The ability to control how the text appears on a web-page is an essential component of any web-page. In this lesson, we have shown how text can be manipulated using font properties and text attributes. In addition, you will have learned how to manipulate the size of text using absolute and relative measurements.