Lab 3: CSS Positioning

Objectives

1. Understand the CSS properties related to the Box Model.
2. See how changing the display type can affect content positioning.
3. Understand how flexbox can be used to create a page layout.

Setting up your webspace

Download the ZIP file for this week's lab. Extract the files and folders.

To add the labs files for this week drag the folder required into the Visual Studio Code workspace.

The files in the *lab3* folder will now appear on the left-hand side.

View the files

Your file structure should appear as follows:

```
someDrive:\year1\web-dev\css-positioning\
someDrive:\year1\web-dev\css-positioning\images
someDrive:\year1\web-dev\css-positioning\styles
```

As with the last lab there will be a number of HTML files created for you: *index.html*, *qualifications.html*, *skill-set.html* and *work-experience.html*. All four HTML pages are attached to the stylesheet *styles/main.css* which was done using the following:

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

There is also a file called *flexcontainer.html* that we'll use to experiment with. Apart from when experimenting with *flexcontainer.html* the CSS in this lab should be added to *styles/main.css*.

Using More Semantic HTML5 Elements

Our previous designs were very dependent the on `<div>` tags. Such dependency can lead to what is known as 'div soup' where HTML is dominated by many nested `<div>` tags. With HTML5 we can reduce this dependency on `<div>` and use more semantic tags. This technique will make our pages easier to read for both developers as well as automated systems as the HTML will now be more descriptive.

Open the *index.html* page and notice the use of `<header>`, `<nav>`, `<main>`, `<section>` and `<footer>` to replace `<div>` elements from the previous lab files. For example:

```
<div id="header">
…
</div>
```

… has been replaced with the more semantic `<header>` element.

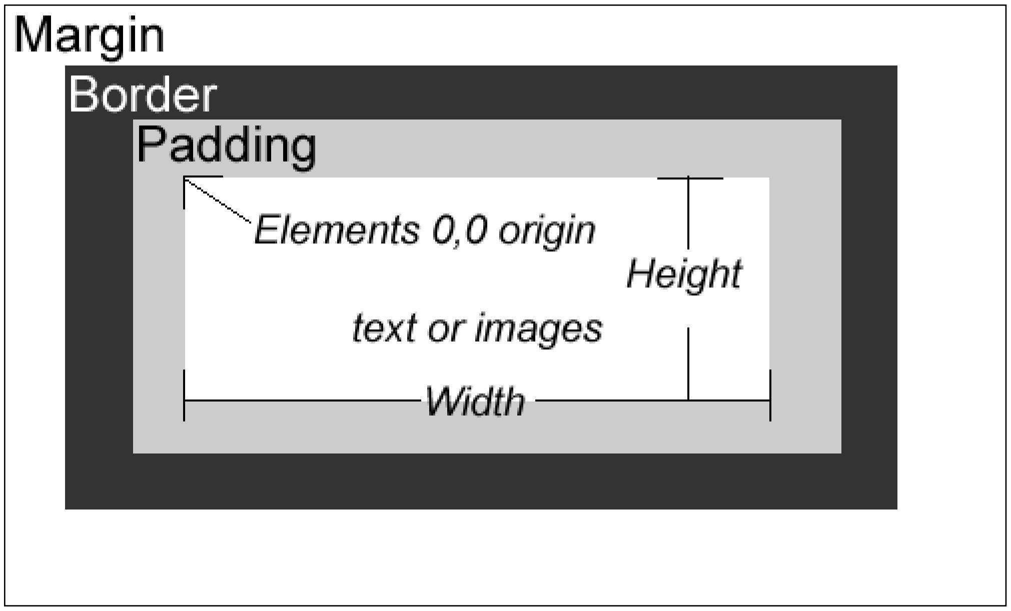
```
<header>
…
</header>
```

Chrome Inspector

In this lab you'll find it very useful to use Google Chrome Inspector. To access the Inspector *right-click* anywhere in the page and choose *Inspect*. Under the elements tab you can see the HTML and CSS that is currently in play.

The Box Model

Every HTML element creates a box. With CSS, the properties of this box can be manipulated. These are the box's margins, borders, padding, width and height.



The box model explains how the various properties interrelate. The element itself is enclosed by three settable properties. Padding is closest to the element, the border is between padding and the margin, and the margin is placed around the border. Each of these three has its own properties and all are optional. In some instances, the box properties cannot be set. Most notably, inline elements (such as `<a>` and `<span>`) cannot have their widths set (unless their display type is changed via CSS).

Box Model Margin

Margin is the space outside the border and adjacent to other content in the page.

Margin values can be set individually Ie:

```
margin-left: 50px;
margin-right: 150px;
margin-top: 10px;
margin-bottom: 150px;
```

The same value can be set to all four sides using the shorthand:

```
margin:10px;
```

If different values are required for all four sides then the following shorthand can be used:

```
/* Clockwise from the top left corner */
/* i.e top right bottom left */
margin: 10px 50px 20px 30px;
```

The same value can be assigned to the top/bottom and left/right using the shorthand:

```
/* top/bottom left/right */
margin: 10px 50px;
```

Margin has a special value of `auto` that can only be applied to left/right margins. When an element has a width value, `margin: auto` will allocate whatever margin is available evenly between the left and right hand sides – this has the effect of centering the container.

Box Model Padding

Padding is the space inside the border.

As with `margin`, `padding` values can be set individually ie:

```
padding-left: 50px;
padding-right: 150px;
padding-top: 10px;
padding-bottom: 150px;
```

The same value can be set to all four sides using the shorthand:

```
padding: 10px;
```

If different values are required for all four sides then the following shorthand can be used:

```
/* Clockwise from the top left corner*/
/* i.e. top right bottom left */
padding: 10px 50px 20px 30px;
```

The same value can be assigned to the top/bottom and left/right using the shorthand:

```
/* top/bottom left/right */
padding: 10px 50px;
```

Box Model Border

This is the line drawn around an element and is separated from the element by any padding values set. The border properties are:

border-width	The width of the border using either a specific value in a specific unit or the relative values `thin`, `medium` or `thick`.
border-style	The style of the border can be `dotted`, `dashed`, `solid`, `groove`, `ridge`, `inset`, `outset` or `none`.
border-color	The border colour as a hexadecimal or RGB value

Properties can be set specifically for each of the borders by using a rule that specifies the border as left, right, top or bottom. For example:

```
border-top-style: solid;
border-top-color: #006666;
border-right-style: dotted;
border-right-width: 0.25em;
```

The most popular way to set a border is with the shorthand `border` property. This sets the three properties outlined above in one go. The rule can be written using the following syntax:

```
border: width style colour;
```

This shorthand can also be applied to individual sides ie:

```
border-top: 6pt dotted #FF0000;
```

Box Model width and height

These values can be set in pixels or as percentages. When percentages are used the element's dimensions are relative to any parent container.

Notice that the `width` and `height` properties are calculated inside any margin, border or padding values – this needs to be taken into account when creating a page layout.

Viewing the Box Model

Experiment with the box model by setting values for the *div.container* E.g:

```
.container{
     border:4px solid #ffff00;
     margin:50px;
     padding:50px;
}
```

We'll use `margin:auto` to center the *div.container*.

The `width` of the container can be set to a fixed width eg:

```
.container{
     width:1200px;
     margin:auto;
}
```

This will give the `#container` a width of 1200px and centre it in the browser window.

We can also use `width` values in percentages. Change the `.container` rule to:

```
.container{
     width:100%;
     margin:auto;
}
```

When using a percentage the container will be fluid and expand or contract within the browser window.

We might want to avoid making it too big or too small. As such we can use properties of `min-width` and `max-width` to set lower and upper limits. Amend the `#container` rule to:

```
.container{
     width:100%;
     margin:auto;
     max-width: 1200px;
     min-width: 600px;
}
```

Normal Flow

As well as the box model, an important concept is that of normal flow. This relates to the way HTML block elements flow down the page, and inline elements flow left to right.

Common block elements are `<p>`, `<h1>`, `<div>`, `<header>, <nav>, <header>`, `<main>`, `<section>`, `<footer>`, `<ul>` and `<li>`.

Common inline elements are `<span>`, `<a>` and `<strong>`.

With CSS however we can change this native behaviour. A `<li>` is a block element. However, we can change it to behave like an inline element. In the *index.html* notice the navigation bar in `<nav>`.

Add the following to the stylesheet in *styles/main.css*.

```
nav ul li{
     display:inline;
}
```

Notice how the links in the list are now aligned left to right – this is because they now behave as inline elements.

Many HTML elements have default values. A `<ul>` has a left padding value. This can be removed with:

```
nav ul{
     padding-left:0;
     list-style:none;
}
```

The `list-style:none` is also used to remove any list marker.

The design calls for white links. Therefore, we'll add the following to create a blue header with white links:

```
header{
     background-color: #263248;
     border-radius: 8px;
}
nav li a{
     text-decoration: none;
     color: #fff;
}
nav li a:hover{
     color: #7E8AA2;
}
```

We can tidy the appearance by amending the *nav ul li* and adding a new rule for only the first list item. This will create a dividing border between the links.

```
nav ul li{
     display:inline;
     padding-left:10px;
     padding-right:10px;
     border-left:1px solid #7E8AA2;
}
nav li:first-child{
     border-left:none;
}
```

Introducing Flexbox

There are a number of ways to create page layouts with HTML and CSS. In recent years a technique known as 'flexbox' has become increasing popular with developers.

Flexbox CSS properties can be used to control how different HTML elements behave when filling / 'flexing' into the space available.

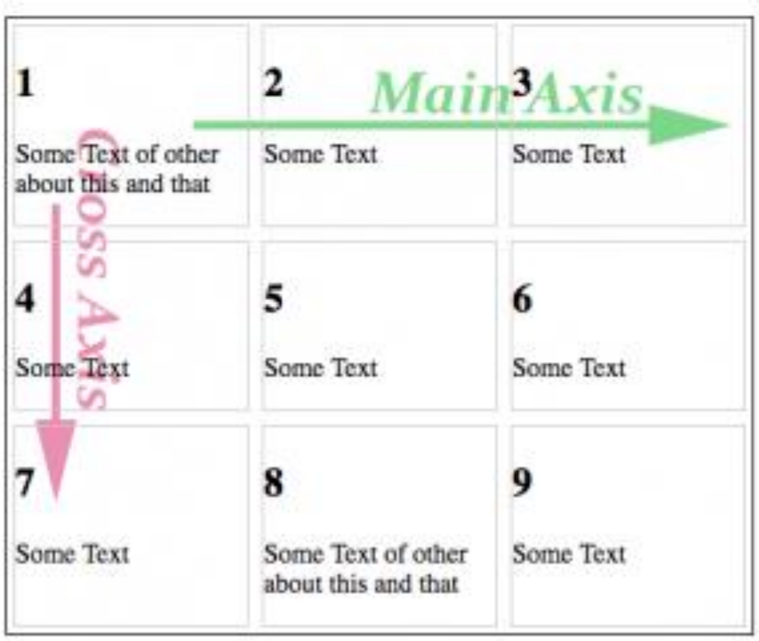
When working with flexbox two types of element are created.

Flex Container: The Flex container is any HTML element that has been allocated a `display: flex` value. Most commonly this is a block element such as a `<div>`.

Flex Item: Flex items are those HTML elements that are the **immediate children** of a Flex container. These can grow or shrink to fill the space available in the flex container.

Flex containers and flex items have a set of properties that can be applied to control how the content of the containers ‘flexes’ to fill the available space.

A flex container has two axis to consider:



By default, flex will try to fit all the flex items in horizontally across the page. This is a solid starting point from which tweaks can be made to achieve the desired effect.

It is best to think of flex as a controlling a row in your design and how it behaves, even if this is to ‘wrap’ to another row.

The core flex properties for the flex container are:

`justify-content`: defines how to distribute space along the main axis. It controls the alignment of flex items.
`flex-direction`: defines the direction of the flex. By default is row. Changes this changes the axis.
`flex-wrap`: defines wrap behaviour. By default, flex items try to fit on a single line. If there's not enough space, they may shrink. To allow items to wrap to the next line, set the `flex-wrap` property to `wrap`.

Others that may prove useful are:

`flex-flow`: shorthand for flex-direction and flex-wrap.
`align-items`: cross axis alignment.
`align-content`: Controls the alignment of lines within the flex container when there is extra space along the cross-axis (due to wrapping). It accepts values similar to align-items. Only applies if flex-wrap applied.

The core flex properties for the flex item is `flex`:

`flex`: This property combines three individual flex properties into one: `flex-grow`, `flex-shrink`, and `flex-basis`. It specifies how a flex item should grow or shrink to fill the available space. The values are as follows:

1. `flex-grow`: Defines the ability of a flex item to grow relative to other items. It accepts a unitless value that represents the proportion of available space it should take up.
2. `flex-shrink`: Determines how a flex item should shrink relative to other items when there is not enough space. It also accepts a unitless value.
3. `flex-basis`: Specifies the initial size of a flex item before it grows or shrinks. You can set it to a specific size or use auto to let the content determine the size.

Commonly we use flex to set a target size for the flex item and indicate whether it can shrink or grow ie:

```
flex:0 0 33%
```

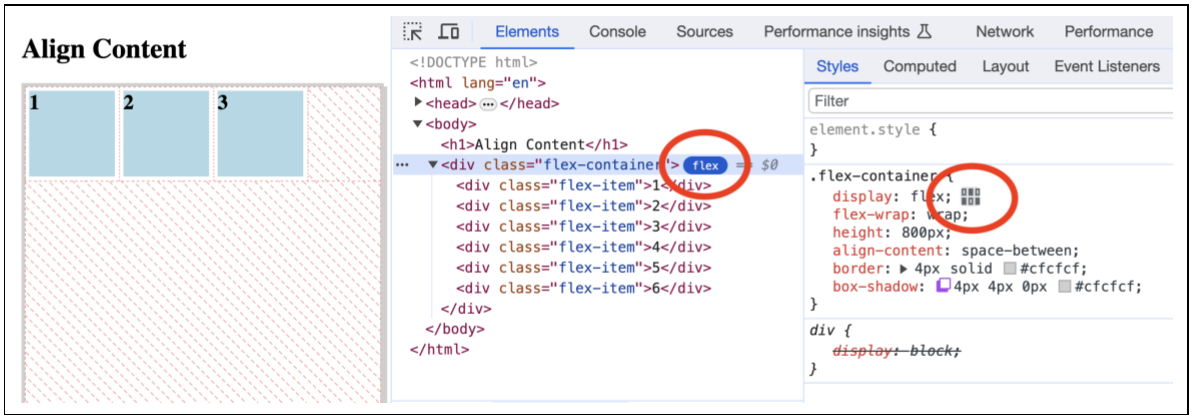
Other lesser used flex item properties are:

`order`: This property controls the order in which flex items are displayed within the flex container. By default, all items have an order of 0. You can use positive or negative integers to reorder items.

`align-self`: Overrides the align-items property for an individual flex item, allowing you to control its alignment along the cross-axis independently of other items.

In the lab files there are a series of example to experiment with in the folder *flex-examples*.

In Google Chrome use the Developer tools to experiment with the settings listed above.



```
<!DOCTYPE html>
<html lang="en">
  <head> … </head>
  <body>
    <h1>Align Content</h1>
    <div class="flex-container"> flex == $0
      <div class="flex-item">1</div>
      <div class="flex-item">2</div>
      <div class="flex-item">3</div>
      <div class="flex-item">4</div>
      <div class="flex-item">5</div>
      <div class="flex-item">6</div>
    </div>
  </body>
</html>
```

```
element.style {
}
.flex-container {
  display: flex;
  flex-wrap: wrap;
  height: 800px;
  align-content: space-between;
  border: ▸ 4px solid #cfcfcf;
  box-shadow: 4px 4px 0px #cfcfcf;
}
div {
  ~~display: block;~~
}
```

Applying Flexbox to our Design

There is much more to flexbox that we've covered here but you'll see how effective it can be to create layouts.

We'll now apply some flexbox to our main design. We are going to look at the header first so before we do so add a couple of rules to tidy up the font colours.

```
header h1 a{
     color:#fff;
     text-decoration: none;
}
header h1 a:hover{
     text-decoration: underline;
}
#logo{
     padding:10px;
}
```

Next, amend the `<header>` rule for as follows:

```
header{
     background-color:#263248;
     border-radius:8px;
     display:flex;
}
```

The `div#header` will now behave as a flex container.

The logo text and the navigation bar are now stacked from the left and we have plenty of available space to the right. As such we can use `justify-content`.

```
header{
     background-color: #263248;
     border-radius: 8px;
     display:flex;
     justify-content:space-between;
}
```

Test and preview your page. The navigation bar should now appear to the right-hand side.

Next we'd like a sidebar on the right hand side of the main content. Add the following to distinguish between the *sidebar* and the *content*.

```
section{
     padding:0 10px 10px 0;
}
.sidebar{
     padding:20px 40px;
     border:1px solid #ccc;
     border-radius:8px;
     margin: 70px 0 0 10px;
 }
```

Both `<section>` and `.sidebar` are children of `<main>` so by adding the following they will become flex items.

```
main{
     display:flex;
}
```

We might expect when resizing the browser window that the `<section>` content would adjust. However, it currently won't because of the image in that part of the HTML. The image cannot by default be resized as it has a given width and height. As the image has a class of `.resize-img`, by adding the following we can change the behaviour of the image such that it will expand or contract to fit inside its parent element.

```
.resize-img{
     width:100%;
     height:100%;
}
```

This now the image in the `<section>` resizes as required. However, we only want the `.sidebar` to have a width of 200px (the width of the profile PNG image) and we don't want it resize. Therefore we should add the following to .sidebar:

```
flex:0 0 200px;
```

The `.sidebar` now cannot flex (grow or shrink) as `flex-grow` is 0 and `flex-shrink` is 0 and has a `flex-basis` of 200px.

Test your page in the browser.

For completeness we could also add the following to the `.section`:

```
flex: 1 1 auto;
```

This indicates that the .section can grow or shrink and its width/flex-basis should be automatically set based on the space available.

Finally add the following to style the footer.

```
footer{
     border-top:1px solid #ccc;
     margin:20px 10px 10px 0;
     font-size:0.8em;
     text-align:center;
     font-style:italic;
}
```

Styling the table

To finish this example, other styling can be added for the `<table>` on the *qualifications.html* page.

```
table{
     width:100%;
     max-width: 600px;
     min-width: 300px;
}
th, td{
     padding: 10px 4px;
     text-align: left;
}
th{
     text-transform: uppercase;
     border-bottom: 4px solid #000;
     background-color: #fff;
}
tr:nth-child(odd){
     background-color: #ccc;
}
```

The `tr:nth-child(odd)` selector is used to give the table a stripped effect by colouring odd table rows.

Creating and Styling a Definition List

On the *skill-set.html* page, there is an HTML construct we haven't seen which is called a definition list. This consists of:

`<dl>`	to define a definition list.
`<dt>`	to define a definition term.
`<dd>`	to contain a definition description.

This can be styled like any other HTML. Add these rules to your stylesheet.

```
dt{
     font-weight: bold;
     border-top: 1px dashed #ccc;
     padding: 20px 0;
}
dt:first-child{
     border-top: none;
}
dd{
     padding: 0 0 20px 0;
}
```

Float

Until recently CSS layouts almost exclusively relied on a CSS property known as `float`.

With `float` values of `left` or `right` could be applied to move HTML elements (float them) to the left or right respectively. This technique was used in combination with the box model to stack HTML elements to create grids. The most well-known exponent of this technique was the CSS Bootstrap framework.

However, modern browser support for flexbox is such that you no longer need to use the `float` technique for layouts. As such `float` can be relegated to want it was originally designed for which is allowing an element to be placed along the left or right side of its container. This then allows text and inline elements to wrap around the floated element.

In the *index.html* add the following `<img>` tag to the second paragraph of text.

```
<img src="images/300x200.jpg" alt="" class="left-float">
```

In the *main.css* create a rule `.left-float` as follows:

```
.left-float{
     float:left;
     padding:0 15px 15px 0;
}
```

The image will now 'float' to the left allowing text to flow around it. A padding value of 15px is set to the right and bottom only, in order to pad the image away from the text.

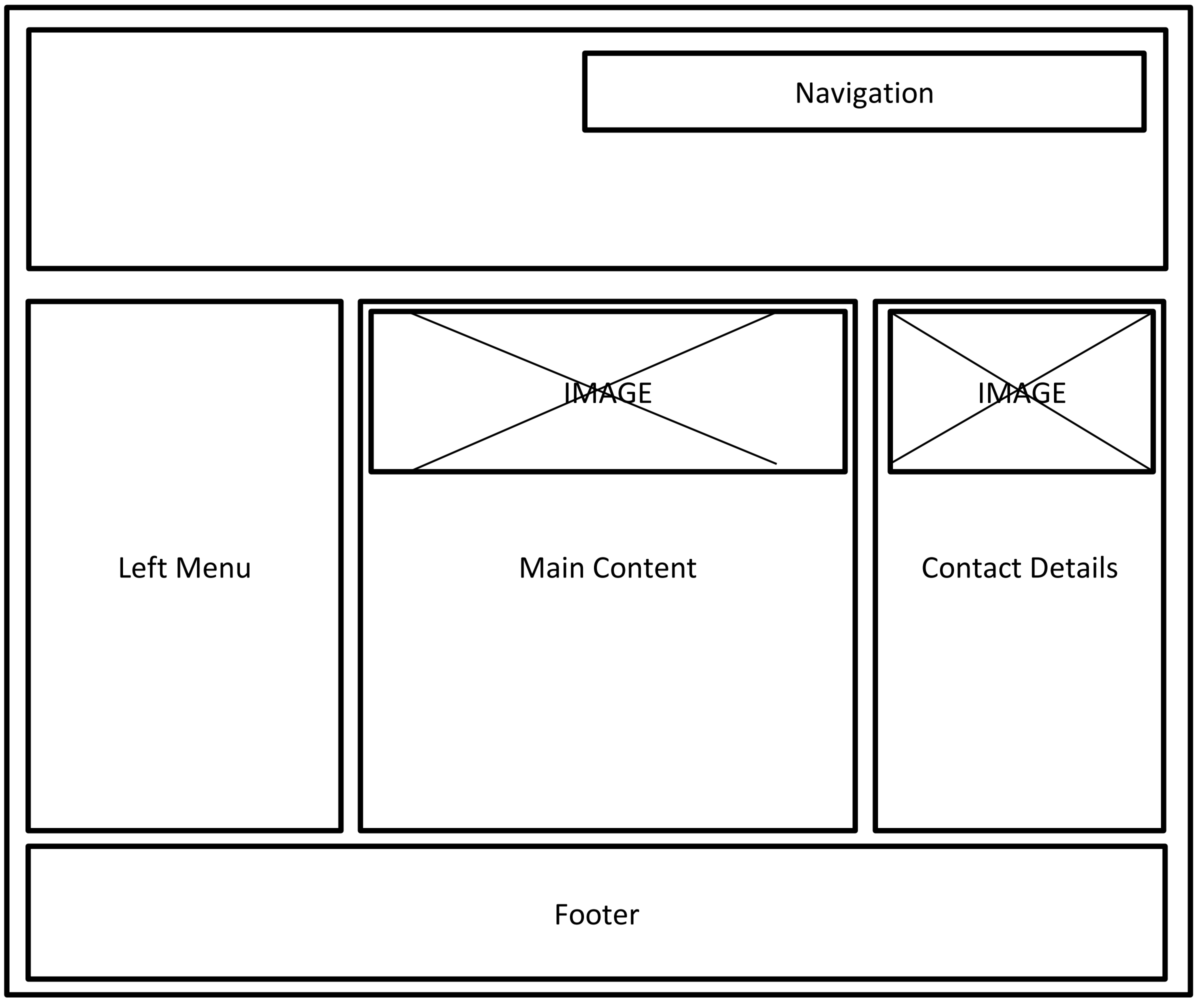
Try creating a `.right-float` rule and floating an image to the right hand side.

Advanced Task

How might you achieve the three column design indicated in the wireframe below. You will need to add some new HTML for the left menu with dummy links such as:

```
<div class="sidebar">
     <ul>
          <li><a href="#">Link 1</a></li>
          <li><a href="#">Link 3</a></li>
          <li><a href="#">Link 3</a></li>
      </ul>
</div>
```

Amend the CSS rule such that the left menu and contact details take up around 25% of the available flex space compared to the main contents 50%.


Navigation
IMAGE
IMAGE
Left Menu
Main Content
Contact Details
Footer