Grouping Elements and Semantic Structure

What we know so far…

So far, we have used a limited number of *elements* (aka: tags) in our pages:

- Headings, paragraphs, lists and images

As we have seen, with CSS and clever use of *selectors*, we can transform the appearance of these elements to suit our design.

However, once our sites become more complicated, these elements alone are not enough!

Page regions

Commonly, web pages are divided into distinct *regions* or *sections*:

- Navigation menus, Branding, Content, Social media feeds, Advertising, Et cetera.

The elements contained in each *region* are thematically related and the regions are styled so they are separate from the rest of the page

- E.g. With borders, backgrounds or simply with empty space around the region

In this exercise, we will be uncovering techniques involved in creating and styling *regions* of the page.

Consider the navigation bar we made…

The HTML for the navigation bar consisted of a *ul* element, which had *li* elements inside it.

- In other words, the *ul* functioned as a *container* for the list items

Containers are extremely useful:

- They can provide extra information about their content…
  - *ul* says its children are a group of related items
- They provide *hooks* for us to attach styles to…
  - By adding border/background to the *ul* we visually defined the navigation bar as a distinct *region* of the page

HTML grouping elements

HTML provides a wide range of *grouping* (or *container*) elements that we can use the same way we used the *ul* with our navigation bar.

Like the *ul* and *ol*, most of the *grouping* elements provide information about the elements within them.

However, unlike the *ul* and *ol* elements, we can put anything we like inside the other *grouping* elements (that is, we are not limited to using *li* elements).

Some *grouping* elements:

- **main** : represents the main content of a page and can only be used once on each page. Should not contain anything that appears on other pages.
  - E.g. On the balloons website, the content heading and the text that follows it could be placed in a *main* tag, but the site heading and navigation bar would not.
- **section** : represents a generic section of a document or a group of thematically related elements (usually with a heading).
  - E.g. Each of the three *summaries* on the balloons home page could be contained in a *section* (the heading and text). Each of the sub-headings and accompanying text on the balloons services page could also be considered *sections*.
- **article** : represents a self-contained composition in a document (that is, it still makes sense if read without the rest of the page content).
  - The example for *main* above could also apply here, but not on the home page. However, each of the home page summaries could be placed in an article tag.
- **nav** : represents a collection of links to other pages or to areas within the page (can be more than one on each page).
  - E.g., the balloons website navigation bar
- **header** : represents a group of introductory or navigational aids for the region it appears in.
  - E.g. The *Balloons Ltd.* h1 heading *and* the website navigation bar, or just the h1 heading on its own.
- **footer** : represents a footer for the region it appears in.
  - E.g., Displaying the page author's details, the date the page was published, Copyright info., et cetera.
- **aside** : represents a section of the page with content connected tangentially to the rest of the page, but which could be considered separate from that content.
  - E.g., Related advertising, Related links, Author's biography
- **div** : Used to group elements for visual purposes only. Provides **no** semantic information about its contents.
  - E.g., we want to place a border around a group of elements, but they don't belong in an article/section/nav/whatever, then we use a div.

Grouping elements usage

The *grouping* elements can be used on their own to group a collection of elements.

Or different grouping elements can be *nested* within each other to present the content in a highly semantic manner.

An *article* on its own:

```
<article>
    <h1>Our products</h1>
    <p>Visit our products page...</p>
    <p>Blah blah blah...</p>
</article>
```

An *article* with a *header* and a *footer*:

```
<article>
    <header>
        <h1>Code Tips</h1>
        <p>Everything you need to know to write code!</p>
    </header>
    <p>Get yourself a good editor and:</p>
    <ul>
        <li>indent your code</li>
        <li>be consistent with the CaSe of you FiLEnAmeS</li>
        <li>use meaningful class names</li>
    </ul>
    <footer>
        <p>Published on 7th August 2016 by Mr Coder</p>
    </footer>
</article>
```

An *article* with *sections*, a *header,* and a *footer*:

```
<article>
    <header>
        <h1>Code fuel</h1>
        <p>For late night coding sessions!</p>
    </header>
    <section>
        <h2>Coffee</h2>
        <p>Without coffee, the late nights are tough!</p>
    </section>
    <section>
        <h2>Chocolate</h2>
        <p>Goes great with coffee!</p>
    </section>
    <footer>
        <p>Published on 12th June 2016</p>
    </footer>
</article>
```

Styling the grouping elements

By default, all the *grouping* elements are *block* elements.

They have no default styling applied to them by the browser (no padding, margin, border, et cetera.). So, when you add them to your HTML, you will see no difference!

- they simply create an invisible *box* that surrounds the elements contained within them

However, we can use the same CSS properties that we have used with other elements to style them as we please.

- borders, backgrounds, et cetera.

Except for the *main* element, it is common that we use multiple instances of each *grouping* element on our pages. Therefore, we tend to use *class* attributes with them a lot.

Let's group some elements!

Download the *grouping.html* and *grouping.css* files from Moodle and place them in a new folder in your websites folder. Call the folder *grouping* or something equally intuitive.

Open both files you downloaded in your editor, and open *grouping.html* in your browser.

Review the code in *grouping,html* . Most of it should be familiar to you, except for the very bottom of the page where a **footer** element has been used to group some paragraphs.

If you view the page in your browser, you will see that the *footer* has been styled with a background colour and other things.

Review the code in *grouping.css* . Again, most of it should be familiar to you except for the *footer* rule. However, as you can see, we write a rule for it the same way we write other rules.

An additional advantage to using a *grouping* element in this scenario is that some styles we specify for the *footer* will be *inherited* by all the elements contained within it.

- E.g. *font-size*, *color*, *text-align*

Immediately below the *footer* rule some skeleton rules have been defined for you. In this exercise, you will be adding declarations to these rules.

A presentational group

First, we will demonstrate a common web design technique that involves placing the entire page within a single grouping element.

Because this group is just for visual effect, we will use the **div** element. To distinguish it from other *divs*, we will give it a *class* attribute with the value: **page-container.**

Open *grouping.html* in your editor:

- On the line after the opening *body* tag, add the opening tag for a *div*
  - Give it a *class* attribute with the value **page-container**
- It should look something like this:

```
<body>
    <div class="page-container">
        <h1>Element Groups</h1>
```

- On the line before the closing *body* tag, add the closing tag for the *div* • It should look something like this:

```
        </footer>
    </div>
</body>
```

- Preview the page in your browser. You should see no difference.

Open *grouping.css* in your editor:

- Locate the *rule* for **.page-container** • Add these declarations to the rule:
  - Set the **width** to **90%**
  - Set the **padding** to **0.5em**
  - Set the **background-color** to **#ffffff**

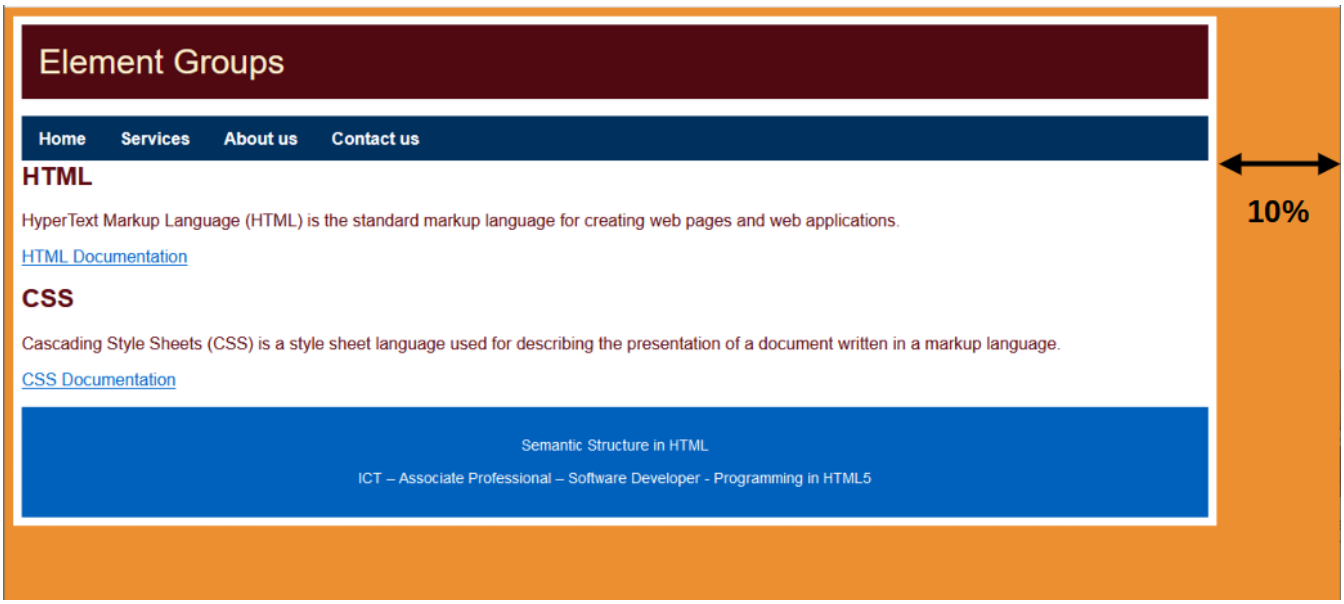
- Save your files and preview *grouping.html* in your browser. You should see that:
  - The page content no longer fills the screen width (because of the *90%* value we specified for the group's width)
  - The *group* now has a white background (but the *body* still has an orange one)
  - The page content does not touch the edge of the *group* box (because of the *padding* we applied to it)

This simple example shows us how we can use a *group* to control elements collectively (the width we set affects everything in the group). It also shows how we can delineate the *group* by adding a background colour (we could also give it a border).

To complete this simple layout technique, we will center the *group* box within the *body* element as opposed to leaving it in its current position (touching the left edge of *body*).

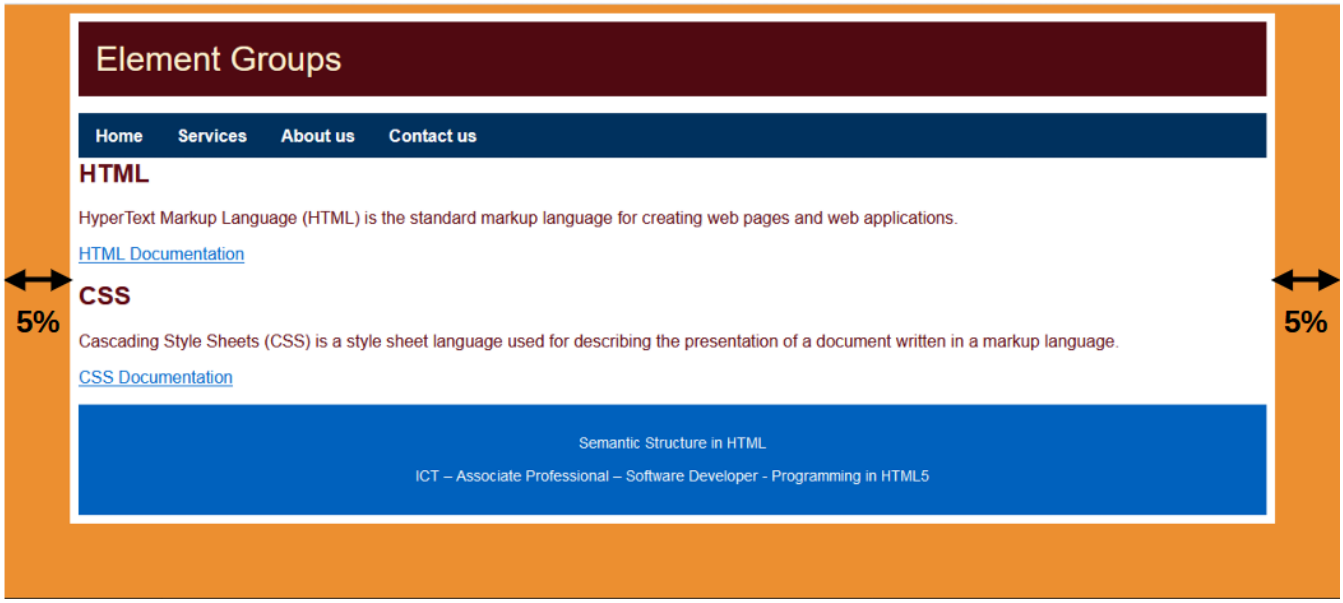
To do this, we will take advantage of a special value that can be used with *margin*. If we set the left and right margin of an element to **auto**, the browser will divide the available space between the two sides, creating a centering effect.

The group's *width* is set at *90%*, which means *90%* of its parent's width (its parent is *body*). The remaining 10% can be seen on the right edge of the screen (with the orange background).



By setting the group's left and right margin to *auto*, that space on the right of the screen will be divided evenly between the left and right side of the group (see *Appendix 1* at the end of this document for more info).

- In *grouping.css* , add a declaration to the *.page-container* rule which sets the top and bottom margin to *zero*, and the left and right margin to *auto*:
  - `margin: 0 auto;`
- Save your files and preview *grouping.html* in your browser. The *group* should now be centered within the browser window.



Semantic groups

Now we will add some grouping tags for semantic purposes only. We will not be styling these elements, although we could if we wanted to.

First, we will address the navigation bar.

The navigation bars we have built all used tags to define the menu structure. This is good as it tells devices that the links are part of a group. However, the tells nothing about the nav bar's relationship to the rest of the page. We can resolve this with the <nav>tag.

In *grouping.html* , locate the that represents the navigation bar. Add a <nav>element to the page which will *wrap* around the <ul>.

E.g.:

```
<nav>
    <ul class="navigation">
        <li><a href="#">Home</a></li> <li><a
        href="#">Services</a></li> <li><a
        href="#">About us</a></li> <li><a
        href="#">Contact us</a></li>
    </ul>
</nav>
```

As we are not applying any CSS rules to this element, we do not need to add a *class* attribute to it. But we could if we needed to!

Preview your page in the browser. There should be no visible changes.

Now have a look at the description of the <header>element on the MDN reference site. This sounds like something we could place our <h1>and <nav>elements inside.

Locate this comment in *grouping.html* : <!-- Start of the site header -->. Add the opening tag for a <header>element immediately below the comment and add the closing tag above the comment that signifies the end of the header. Make sure both the <h1>and <nav>elements are contained within the <header>.

Preview your page in the browser. Again, there should be no visible changes.

Semantic groups with style

Next, we will add some semantic groups to the page and apply some CSS to them so that they are visibly distinct from each other.

The main element

In *grouping.html* , there are 2 <h2>elements, each with accompanying content. These elements represent the content of the page. The definition for `<main>` states:

represents the main content of a page. Should not contain anything that appears on other pages., so it makes sense for us to use it to group the headings and paragraphs into a single unit.

In *grouping.html* , add the opening tag for a `<main>`element above the first <h2>. Add the closing tag after the last paragraph, but before the `<footer>`starts. As the `<main>`tag is unique within the page, we do not need to add a *class* attribute to it and can simply use a *tag selector* in our CSS.

In *grouping.css* , locate the rule targeting `main` and add declarations to achieve the following:

- Set the top and bottom padding to 2em.

Save your files and preview your page in the browser. There should be extra space above and below the main content.

Now we will use **article** elements to divide the page content into two distinct regions. i.e., a HTML article and a CSS article.

Open *grouping.html* in your editor:

- Locate the first *h2* element: <h2>HTML</h2>. We want to place this element and the two paragraphs that follow it in an *article* element.
- Above the *h2*, add the opening tag for an *article*.
  - Give it a *class* attribute with the value: **html-feature**
- After the second paragraph that follows the heading, add the closing tag for the *article*. • When

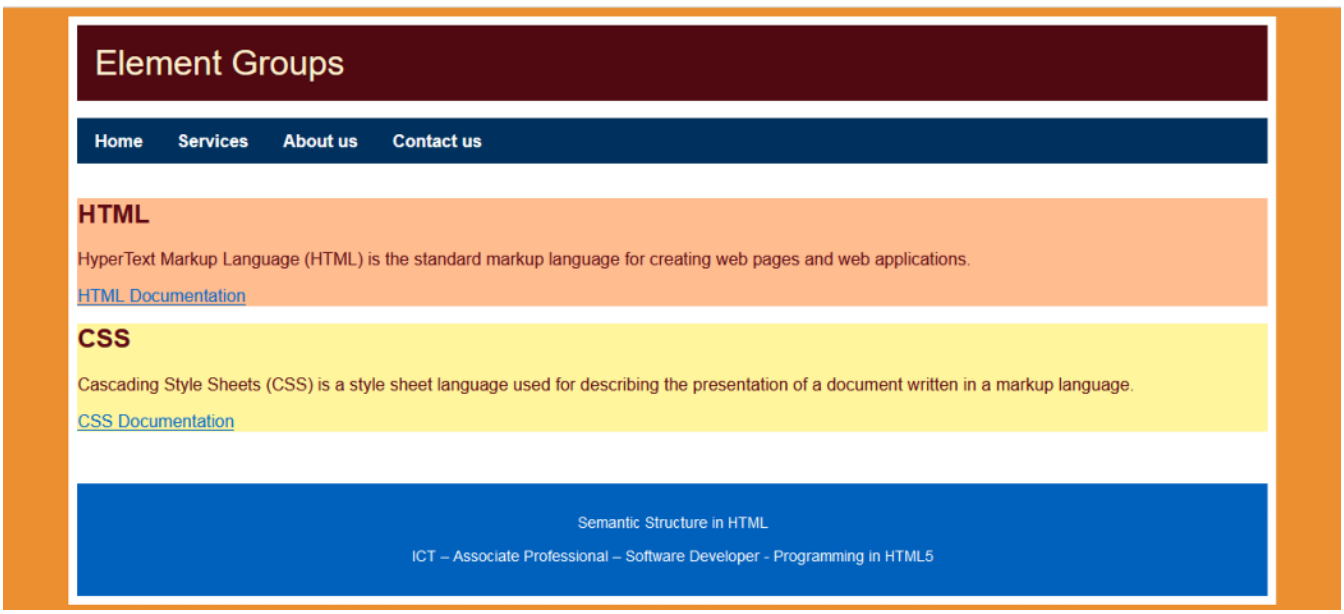
done, it should look something like this:

```
<article class="html-feature">
    <h2>HTML</h2>
    <p>HyperText Markup Language (HTML) is the standard markup language for
       creating web pages and web applications.</p>
    <p><a href="https://developer...">HTML Documentation</a></p>
</article>
```

Save your files and preview *grouping.html* in your browser and you should see the *article* is high-lighted with a background colour (the CSS rule for this was already in place in the style sheet).

Now, repeat the same process for the other *h2*: <h2>CSS</h2>and the 2 paragraphs that follow it, but this time, the value to use as the *class* attribute is **css-feature**.

Save your files and preview *grouping.html* in your browser and you should see the new *article* is also highlighted with a background colour (again, the CSS rule for this was already in place).



Now we will style the two articles so that they are displayed side-by-side.

In *grouping.css* , locate the rule with the *grouped* selector: **.html-feature, .css-feature**. Styles we declare in this rule will apply to both articles. Add a declaration to the rule, setting the *padding* on all sides to **1em** (this makes space *inside* each of the article boxes).

To place the *articles* side-by-side, we will use *CSS Grid*. This is like *flexbox* but allows for grid-like layouts as opposed to the linear layouts that flexbox creates. Like *flexbox*, we enable *grid* by applying certain CSS rules to the *parent* of the elements we want to arrange (in this case, the <main> element).

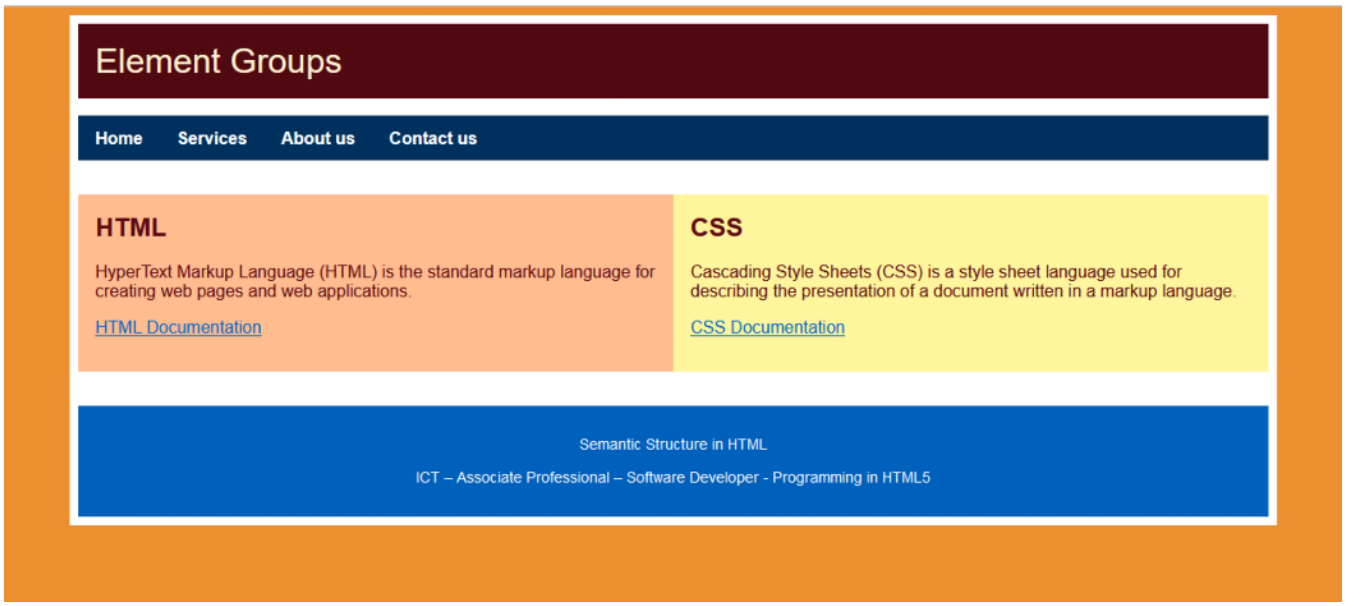
In *grouping.css* , locate the rule that targets the `main` element. Add two new declarations to the rule:

- Set the *display* property to *grid* (this enables the grid layout model)
- Set the *grid-template-columns* property to *1fr 1fr* (this says: make two columns of equal width, using all available space)

Your CSS should now look like this:

```
main {
    padding: 2em 0;
    display: grid;
    grid-template-columns: 1fr 1fr; }
```

Save your files and preview *grouping.html* in your browser. The two articles should be displayed side by side.



The final step is to make space between the two articles so there is clearer separation between them. We can do this by setting the *column-gap* property on the grid *container.*

```
main {
    padding: 2em 0;
    display: grid;
    grid-template-columns: 1fr 1fr;
    column-gap: 2em; /* Make space between the columns */ }
```

Save your files and preview the page in your browser. There should now be space between the 2 boxes.

Summing up

In this exercise we have seen how grouping elements make it easier for us to manipulate the layout of the page. However, you should be aware that most of these grouping elements also effect the *meaning* of the page content, so take care when choosing which to use! If in doubt, use a div!

We have also seen how adding semantic grouping elements does not change the way the page looks, so there is no excuse for not using them!

We have also been introduced to *CSS Grid*, which is an incredibly useful tool that allows for the creation of complex layouts easily. More info about grid:

- https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Grids
- https://www.w3schools.com/css/css_grid.asp
- https://css-tricks.com/snippets/css/complete-guide-grid/

Appendices

Appendix 1: centering things

CSS provides two different ways of centering things. Which one we use depends on what we want to center!

Centering text

We can use the **text-align** property to align text *within its box* . Some of the values we can specify are *left*, *right*, *center*, *justify*.

To center the text within the *h1* element:

```
h1 {
    text-align: center; }
```

Note, this property does not alter element's box dimensions in any way.

Centering boxes

Sometimes, we want to center an element's *box* within its parent element. To do this we must do two things:

- Set the element's *width* to a value that is less than its parent's width. • Set the element's left and right *margin* to **auto**

To center the *h1* box within its parent:

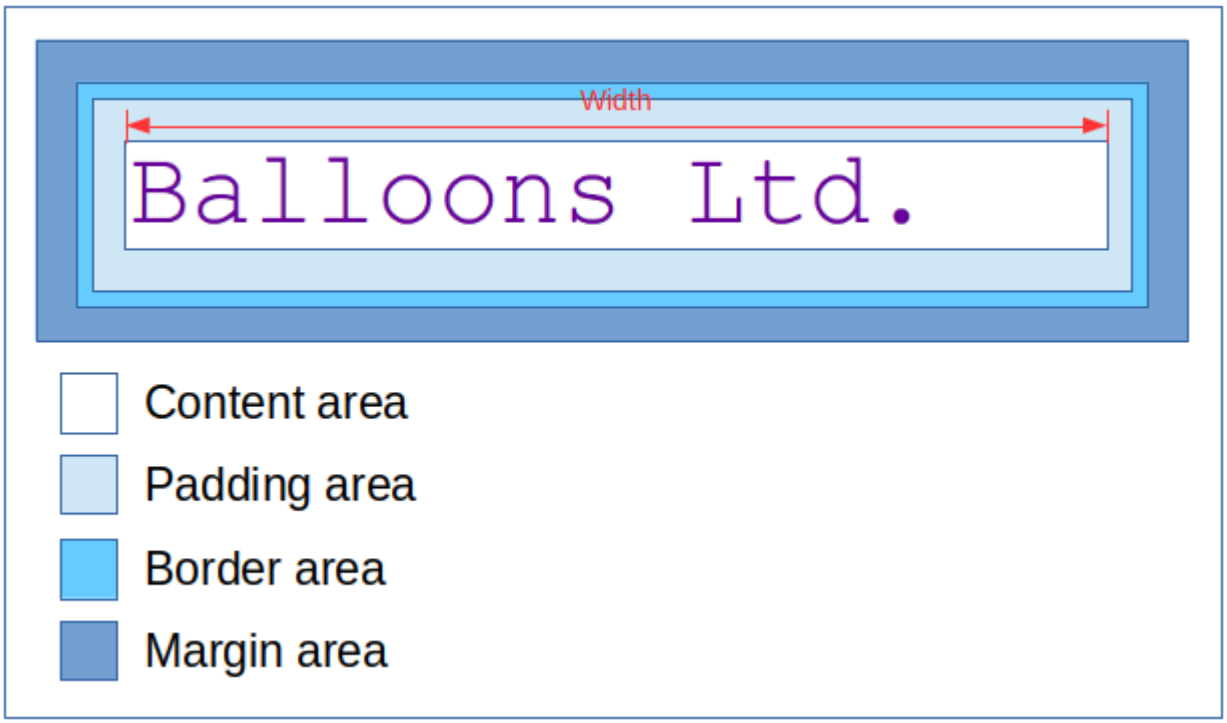
```
h1 {
    width: 75%;
    margin: 0 auto;
}
```

Result, the *h1* will not extend to fill all available space. The remaining space will be divided evenly between the left and right margin.

Note, the text within the *h1* will still be left aligned.

Appendix 2: the *width* property

By default, when we specify the *width* of an element in CSS, the value is applied to the element's *content area*. Remember, the *content area* is *inside* the padding and border areas:



What this means…

If we specify a *width* for an element, but also specify left or right *border*, *margin* or *padding* , the actual space occupied by the element on the screen will be larger than we might expect.

Assume this CSS (values in pixels for simplicity):

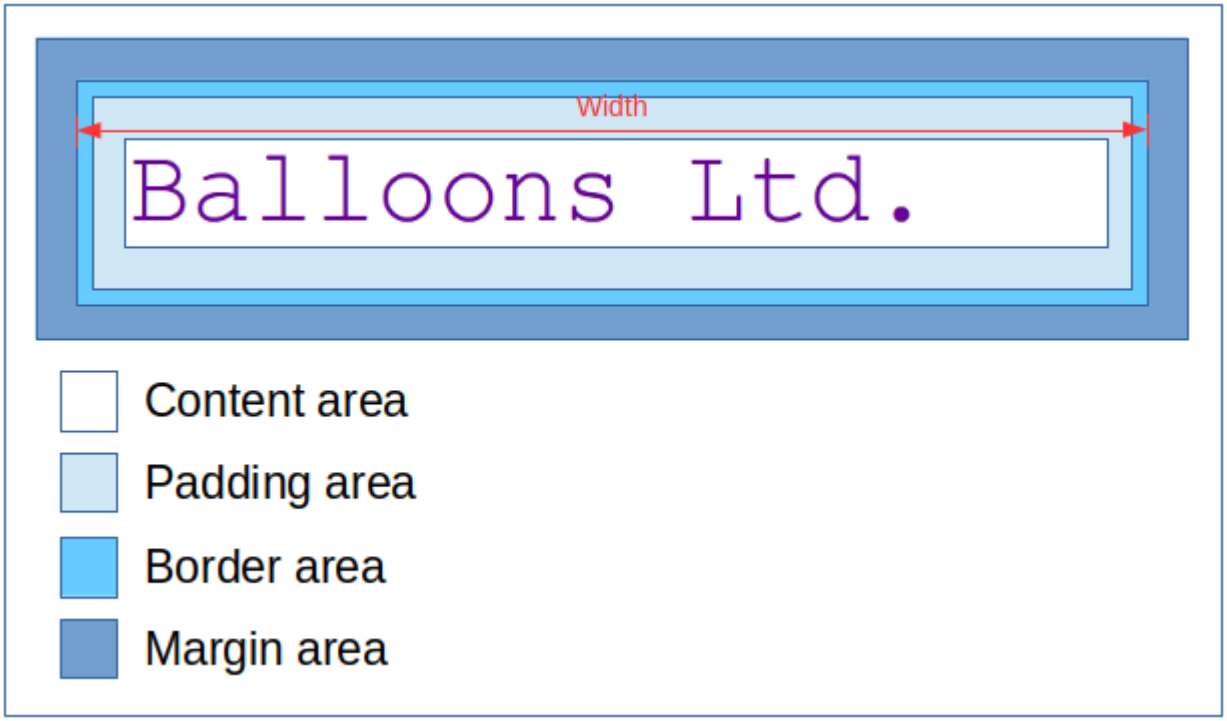
```
p {
    width: 300px;
    border: 10px solid #000000;
    padding: 20px;
}
```

The actual horizontal space occupied by the *p* element will be **360px**

- `width` + L `border width` + R `border width` + L `padding` + R `padding` • `300` + `10` + `10` + `20` + `20`

While this might sound like a crazy way of doing things, it is not so bad once you get accustomed to it. However, when dealing with *relative* values for dimensions (percentages, ems, et cetera.), the math can get quite complicated!

As an alternative, we can use the CSS property **box-sizing** to change the way the browser calculates an element's width. By setting an element's *box-sizing* property to **border-box**, the *width* that we specify will include the *padding* and *border* (but not the *margin*):



The CSS example from earlier, with the *box-sizing* property set to *border-box* :

```
p {
    box-sizing: border-box;
    width: 300px;
    border: 10px solid #000000;
    padding: 20px;
}
```

The actual horizontal space occupied by the *p* element will be **300px** . Due to the padding and border, the *content area* of the element will be **240px** wide.