CSS

- CSS - Cascading Style Sheets is used to style HTML sheets, e.g. altering Fonts, colour, size and spacing of the content. You can even add animations

CSS Syntax:

Selector:

- A CSS selector is part of a CSS rule that describes what the element in a document will match.
- Inside the curly braces are what are called **declarations,** which take the form of a **property** and **value** pairs. We specify the property. In the example below the property is colour and the value is red:

```css
h1 {
color : red;
font-size : 5em;
}
```

▼ **Example:**

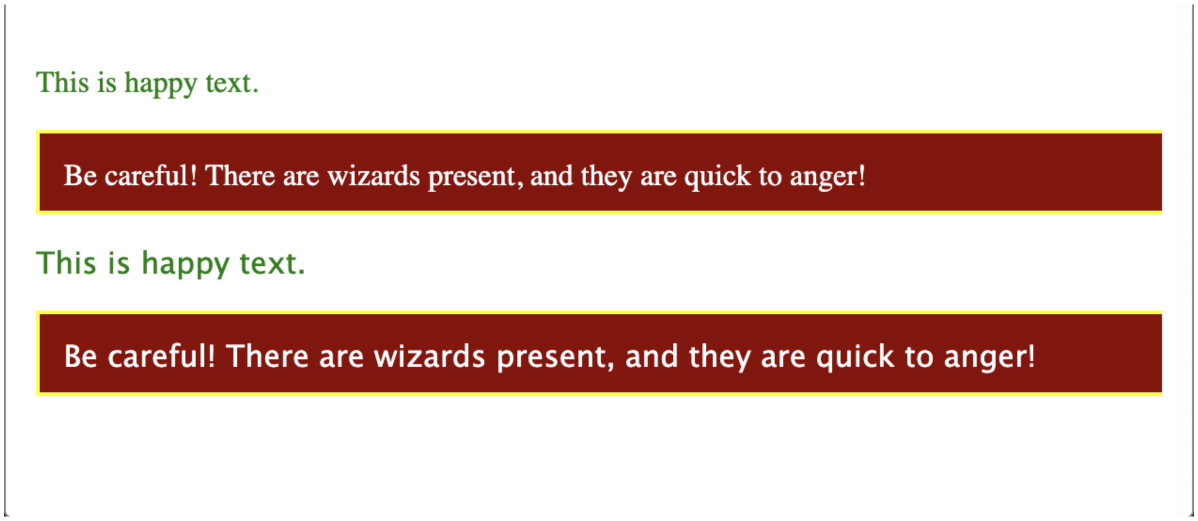
```css
p {
  color: green;
}

div.warning {
  width: 100%;
  border: 2px solid yellow;
  color: white;
  background-color: darkred;
  padding: 0.8em 0.8em 0.6em;
}
```

```
#customized {
  font:
    16px Lucida Grande,
    Arial,
    Helvetica,
    sans-serif;
}
```

- Here the selectors are:
  - `<p>` which will apply the colour green to any paragraph text
  - `div.warning` which makes any <div> element with the class “warning” to look like a warning box
  - `#customised` which sets the base font of the ID “customised” to a 16-pixel tall Lucidia Grande
- The corresponding HTML text looks like:

```
<p>This is happy text.</p>

<div class="warning">
  Be careful! There are wizards present, and they are quick t
</div>

<div id="customized">
  <p>This is happy text.</p>

  <div class="warning">
    Be careful! There are wizards present, and they are quicl
  </div>
</div>
```

Output:


This is happy text.

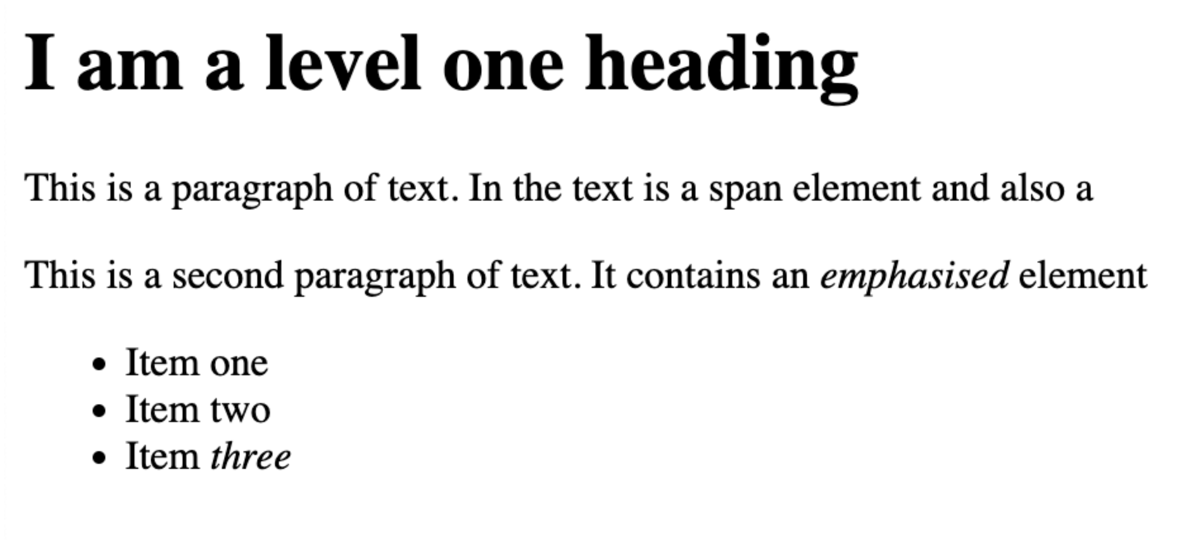
Be careful! There are wizards present, and they are quick to anger!

This is happy text.

Be careful! There are wizards present, and they are quick to anger!

Getting Started with CSS

- HTML doc before styling:


I am a level one heading

This is a paragraph of text. In the text is a span element and also a

This is a second paragraph of text. It contains an *emphasised* element

- Item one
- Item two
- Item *three*

▼ Corresponding HTML code:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Getting Started with CSS</title>
        <link rel="stylesheet" href="styles.css" />
    </head>

    <body>
        <h1>
         I am a level one heading
        </h1>

        <p>
            This is a paragraph of text. In the text is a <s
                span element</span> and also a <a href="https
        </p>

        <p>
            This is a second paragraph of text. It contains
        </p>

        <ul>
            <li>Item <span>one</span></li>
            <li>Item two</li>
            <li>Item <em>three</em></li>
          </ul>

    </body>

</html>
```

Adding CSS to our doc

The very first thing we need to do is to tell the HTML document that we have some CSS rules we want it to use. There are three different ways to apply CSS to an HTML document that you'll commonly come across, however, for now, we will look at the most usual and useful way of doing so — linking CSS from the head of your document.

- This is done by creating a .css style sheet in the same folder as the HTML doc. (actually as long as you provide the path of the CSS doc you can have it anywhere on your computer)
- The `<llink>` goes anywhere inside the `<head/>`
- This `<link>` element tells the browser that we have a stylesheet, using the `rel` attribute, and the location of that stylesheet as the value of the `href` attribute.
- We can target multiple selectors at the same time, by adding a comma, e.g.

```
p, li {
    color: green;
}
## Will turn all paragraphs and list items green
```

Adding a Class:

So far, we have styled elements based on their HTML element names. This works as long as you want all of the elements of that type in your document to look the same. To select a subset of the elements without changing the others, you can add a class to your HTML element and target that class in your CSS.

e.g. Lets add a class attribute to the second item of our list:

```
 <ul>
           <li>Item <span>one</span></li>
           <li class="special">Item two</li>
           <li>Item <em>three</em></li>
         </ul>
```

- We can then target the specific styling to “special” elements:

```
.special {

    color : orange;
    font-weight: bold;
}
```

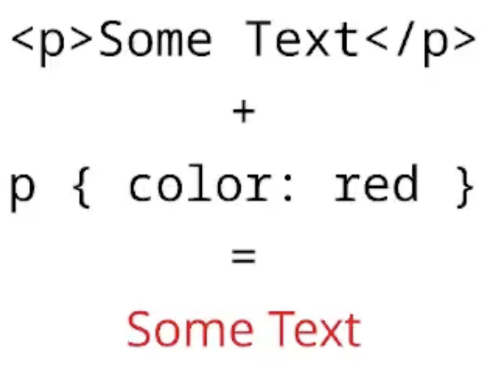
Styling based on location

Something else you might like to try is styling a paragraph when it comes directly after a heading at the same hierarchy level in the HTML. To do so, place a `+` (an **next-sibling combinator**) between the selectors.

```
h1 + p {
  font-size: 200%;
}
```

TOM NOTES

- general idea is that they are additional documents which contain rules which govern how HTML is displayed



```
<p>Some Text</p>
        +
p { color: red }
        =
    Some Text
```

  - e.g. the `p` element with some text in it, then a rule in your style sheet could say that p elements should have their porperty colour set to red —

the browser will then render these elements in red

Selectors

What are they?

A CSS selector is the first part of a CSS Rule. It is a pattern of elements and other terms that tell the browser which HTML elements should be selected to have the CSS property values inside the rule applied to them. The element or elements which are selected by the selector are referred to as the *subject of the selector*.

```
SELECTOR {
    KEY: VALUE;
    ...
}
```

- The `SELECTOR` identifies which elements the rule should apply to
- then the curly braces contain a set (or sets) of `KEY: VALUE`; pairs

Types of `SELECTORS`

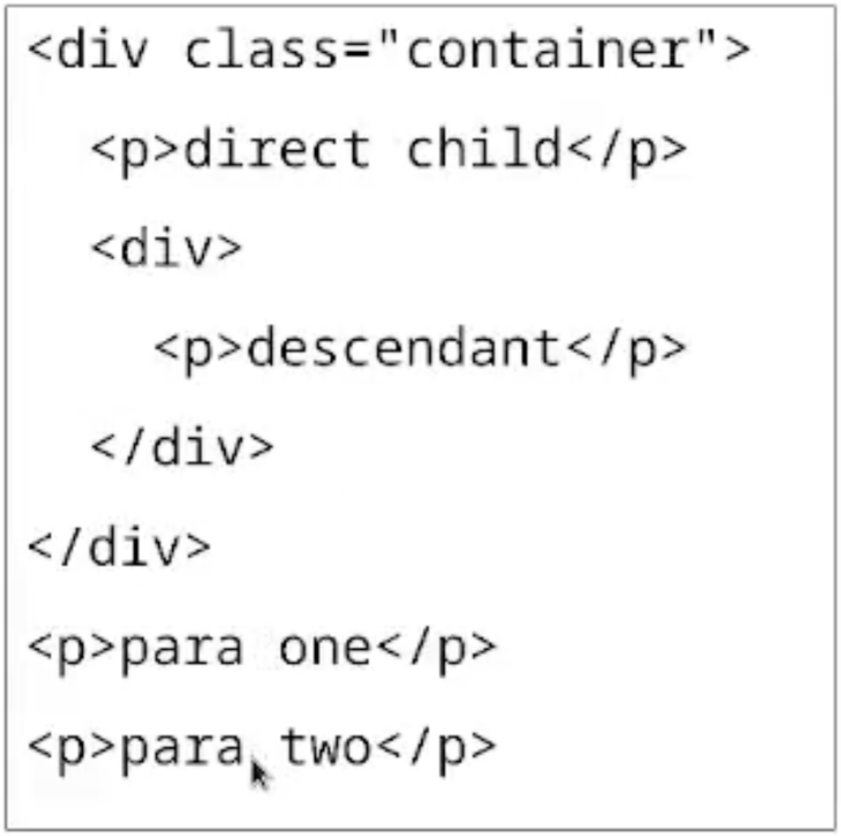
- the simplest form of selector just gives the name of a type of tag → e.g. using `p` to refer to the p tag `<p>`
    - this means that the style you give to this selector will be **applied to all p elements** in the HTML document

```
p    <p>
.important     <div class="important">
#title    <div id="title">
```

```
p.important    <p class="important">
h1#title  <h1 id="title">
```

you often want to alter the style of particular elements with a specific **class**

- e.g. you can begin a selecter with a dot to refer to elements with a class name `.important` in the above refers to any element with a class of important
- can refer to elements of a particular **id** by using a hash `#` ⟶ `#title` now refers to that specific div with the id of title
- you can combine element tags and ID and class selectors to specify kinds of elements e.g. `p.important` would only refer to p elements with the class of important but not to div lements with a class of important
  - `h1#title` would only refer to h1 elements with the class of title



```html
<div class="container">
  <p>direct child</p>
  <div>
    <p>descendant</p>
  </div>
</div>
<p>para one</p>
<p>para two</p>
```

Say if we wanted to only refer to p elements inside a specific class?

`.container p` → this refers only to p elements inside the container class, any elements of the class container

- this would refer to both the direct child p element and the descendant p element

Say if we only wanted to refer tot he direct child?

`.container > p` — this selects **all** direct children

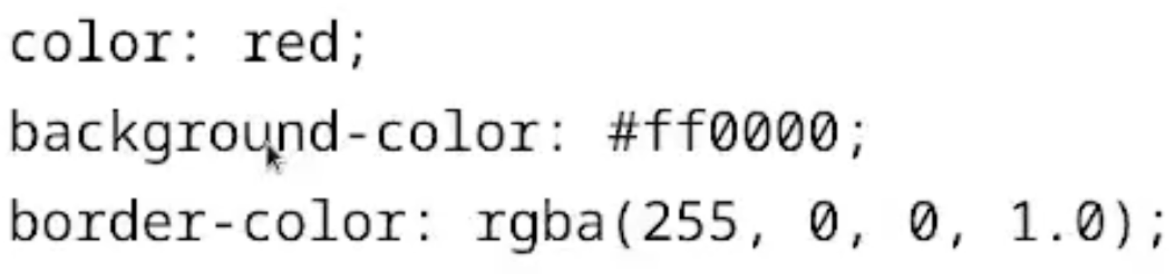
In the above example there are also some p elements which come after the container div but arent container within it, therefore the two code snippets above wouldnt apply to those two elements.

If you want to apply style to an element which follows another element like this there are two options:

1. `.container ~ p` — this refers to ALL p elements which follow a .container class
2. `.container + p` — this refers to the **immediate** p element following the container, in this example: para one but not para two

We can also combine selectors with commas, which is saying we want to apply style to multiple elements at once: `p, div` — this applys the style rules to all p or div elements on the page

Values



```
color: red;
background-color: #ff0000;
border-color: rgba(255, 0, 0, 1.0);
```

Main values are the colour and font being used

There are a variety of ways to refer to a colour, e.g. standard names such as `red` which CSS understands, however if you want more control then there are the hexadecimal RGB colour notation: `#ff0000`

- there this says youre setting the maximum value in Red, and then nothing in green and blue

Finally to get the most control you can use the `rgba` function where you can set decimal values for rgb as well as an opacity value e.g. `(255, 0, 0, 1.0);` - sets red to 255, green and blue to 0 and the opacity to 1.0

Types of Selectors In depth:

Type Selectors

Type selectors target an HTML element such as an `<h1>`:

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

Class Selectors

Class selectors target an element that has a specific value for its `class` attribute:

```css
.box {
}
```

In CSS, you directly use IDs to style elements but you do not "access" IDs in the sense of retrieving their value. Instead, you reference an element by its ID to apply styles. For example:

```css
#elementId {
    color: red;
}
```

Attribute Selectors

This group of selectors gives you different ways to select elements based on the presence of a certain attribute on an element:

```
a[title] {
    color: red;
    text-decoration: underline;
}
```

- this would target all (`<a>`) anchor tags which have a title, colouring them red and underlining them

Attribute selections can even make a selection based on the presence of an attribute with a particular value:

```
a[href="https://example.com"]
{
    colour: blue;
}
```

Psuedo-classes and psuedo-elements

This group of selectors includes pseudo-classes, which style certain states of an element. The `:hover` pseudo-class for example selects an element only when it is being hovered over by the mouse pointer:

```
a:hover {
}
```

It also includes pseudo-elements, which select a certain part of an element rather than the element itself. For example, `::first-line` always selects the first line of text inside an element (a `<p>` in the below case), acting as if a `<span>` was wrapped around the first formatted line and then selected.

```
p::first-line {
}
```

Selector Lists: Assigning multiple elements at the same time

you can save space by rewriting this:

```css
h1 {
    color: blue;
}

.special {
    color: blue;
}
```

- turn this into a **selector list**

You can also combine them into a selector list by adding a comma:

```css
h1, .special {
  color: blue;
}
```

White space is valid before or after the comma. You may also find the selectors more readable if each is on a new line.

```css
h1,
.special {
  color: blue;
}
```

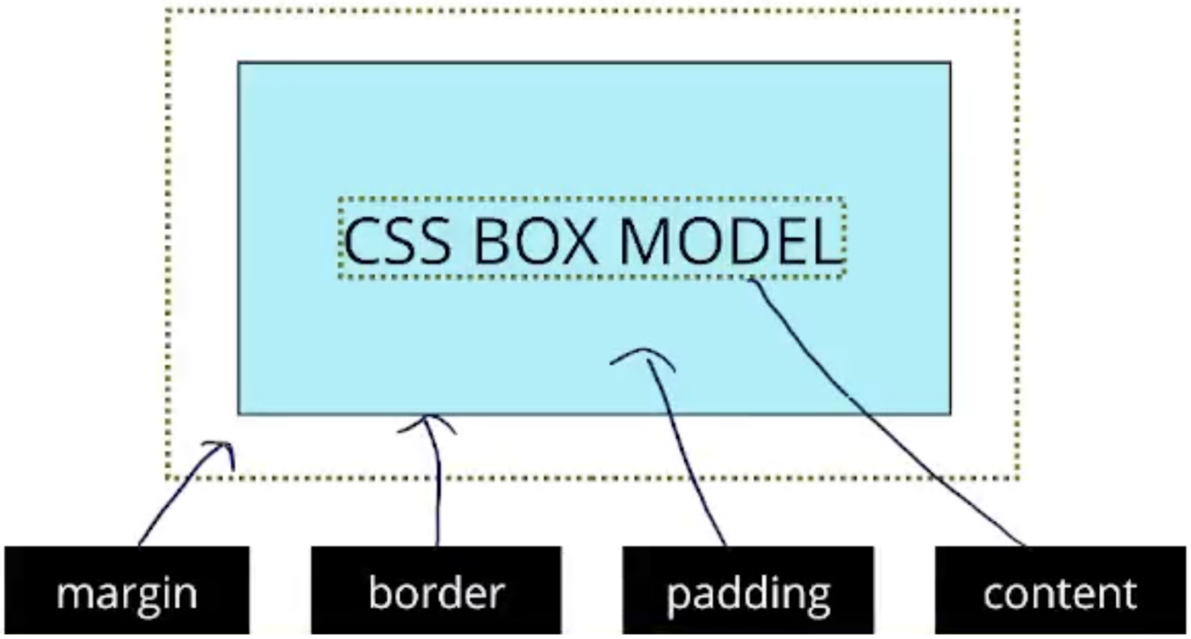
When you group selectors in this way, if any selector is syntactically invalid, the whole rule will be ignored:

```css
h1,
..special {
  color: blue;
}
```

- both h1 and special will be ignored

The Box Model

- every page element can be understood as a box, and a box has 4 main areas:



1. the innermost region — this defines the space for the content itself such as the text

    a. also used to define the size of elements e.g. images

2. **The padding**, is around the content area between the content and the border area and is **inside** the border
3. The border itself
4. The margain - defined **outside** of the border which the element keeps clear to provide visual space between itself and other elements

Margin & Padding

Margin

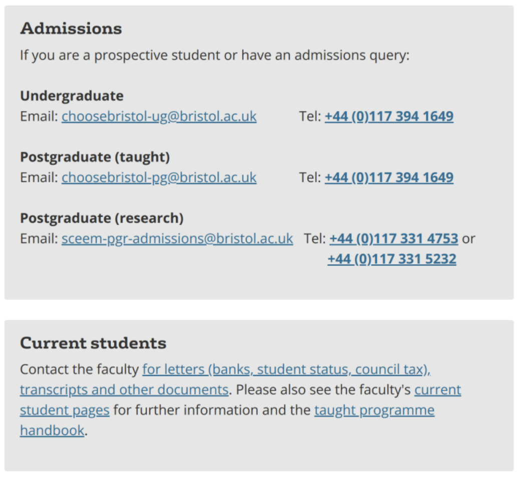
- **Definition:** Margin is the space outside the border of an element. It creates extra space around an element but does not affect the element's size.
- **Usage:** Margin is used to separate an element from surrounding elements without affecting the element's internal dimensions. It is transparent and cannot have a color.
- **Properties:**
    - `margin-top`
    - `margin-right`
    - `margin-bottom`
    - `margin-left`
    - `margin`: shorthand property that can set all four margins at once.
- **Example:** If you set `margin: 20px` on an element, it will ensure there is a 20px space on all sides between its border and any neighboring elements or the document edge.

Padding

- **Definition:** Padding is the space inside the border of an element around its content. It extends the background color or image of the element.

- **Usage:** Padding is used to create space between the content of an element and its border. Unlike margin, padding affects the dimensions of the element it's applied to, increasing its size.
- **Properties:**
  - `padding-top`
  - `padding-right`
  - `padding-bottom`
  - `padding-left`
  - `padding`: shorthand property that sets all padding values at once.
- **Example:** If you set `padding: 20px` on an element, it increases the space between the content of that element and its border by 20px on all sides. This also increases the total dimensions of the element.

An example:



Admissions

If you are a prospective student or have an admissions query:

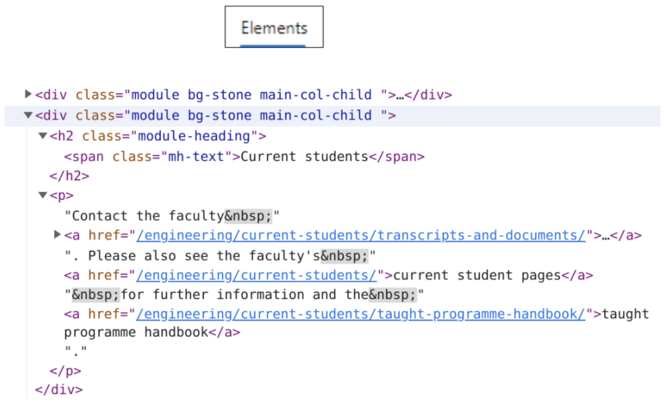
Undergraduate
Email: choosebristol-ug@bristol.ac.uk Tel: **+44 (0)117 394 1649**

Postgraduate (taught)
Email: choosebristol-pg@bristol.ac.uk Tel: **+44 (0)117 394 1649**

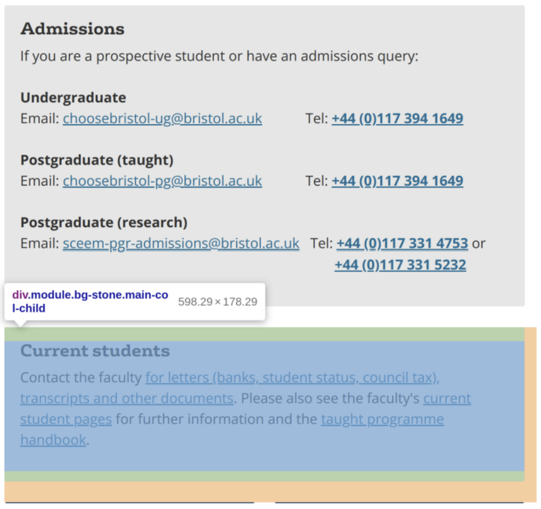
Postgraduate (research)
Email: sceem-pgr-admissions@bristol.ac.uk Tel: **+44 (0)117 331 4753** or **+44 (0)117 331 5232**

Current students

Contact the faculty for letters (banks, student status, council tax), transcripts and other documents. Please also see the faculty's current student pages for further information and the taught programme handbook.



```
▶<div class="module bg-stone main-col-child ">…</div>
▼<div class="module bg-stone main-col-child ">
  ▼<h2 class="module-heading">
     <span class="mh-text">Current students</span>
   </h2>
  ▼<p>
     "Contact the faculty&nbsp;"
    ▶<a href="/engineering/current-students/transcripts-and-documents/">…</a>
     ". Please also see the faculty's&nbsp;"
     <a href="/engineering/current-students/">current student pages</a>
     "&nbsp;for further information and the&nbsp;"
     <a href="/engineering/current-students/taught-programme-handbook/">taught
     programme handbook</a>
     "."
   </p>
 </div>
```


Admissions

If you are a prospective student or have an admissions query:

Undergraduate
Email: choosebristol-ug@bristol.ac.uk Tel: +44 (0)117 394 1649

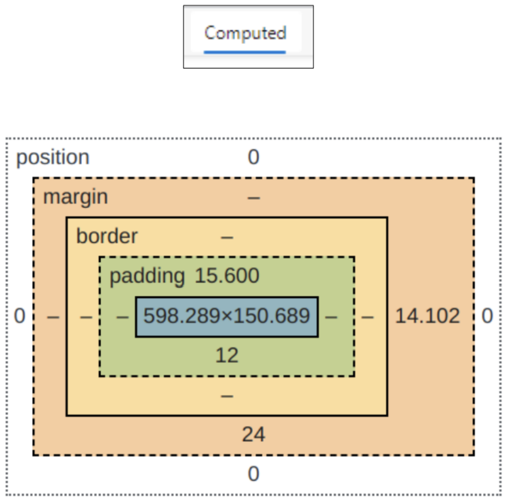
Postgraduate (taught)
Email: choosebristol-pg@bristol.ac.uk Tel: +44 (0)117 394 1649

Postgraduate (research)
Email: sceem-pgr-admissions@bristol.ac.uk Tel: +44 (0)117 331 4753 or +44 (0)117 331 5232

div.module.bg-stone.main-col-child 598.29 × 178.29

Current students

Contact the faculty for letters (banks, student status, council tax), transcripts and other documents. Please also see the faculty's current student pages for further information and the taught programme handbook.



- the blue section is the **content area** itself
- the green is the **area’s padding**
- yellow is border
- the peachy is the margin

How to set these dimensions?

Can set them independandtly eg:

- `margain-top: 10px;`
- `padding-left: 20px;`

Or you can set elements such as the margain all at once:

- `margain: 10px, 5px, 15px, 20px;` — top, right, bottom, left
    - this sets the values **clockwise** starting from the top (12 oclock) then right then bottom then left

For the border you define a thickness as well as a style and colour you want to use:

- `border: 1px solid black`

CSS 2: Units of Measurement

dpi (ppi)

- dots (points) per inch
- some background — the dpi of computer screens used to reliably be **96** you could fit 96 dots in every visible inch of screen
- this is no longer true, most screens can now have upwards of 120 dpi
- for mobile phones that can exceed 200 dpi (iphone 12 pro: 460 dpi)

CSS Units

- `px` - stands for pixel, in practise it is defined as either 1/96th of an inch in printed output, **or** the **smallest** sharply visible space on the device screen
- `pt` - point, defined as 1/72th of an inch
- You can also use typical measures such as `cm` and `in` **(**centimeters and inches**)**
  - On most devices these should result in output of the correct size,
  - But in practise some devices such as low resolution ones may not draw a space that is defined in cm or inches or pt exactly the same way as you would expect.

Due to this it is best to **avoid** absolute measures in styling unless there is good reason to use them.

There are other non-typical measures of size, which are defined in relation to the font that is used in the HTML document:

`em` - a value related to the width of the letter *m* in the current font

`ex` - the height of the letter *x* in the current font

`lh` which is the overall height of a line (+space) in the current font, is slightly less used than the others

These units are helpful if you want to measure things relative to the text that appears on the page.

- however due to how they are defined, the actual size of these values can change even **within the same page**
    - e.g. if you set one element to have a different font to another, anything inside that element which is using `em` or `ex` will have a different sizes

Other units

Viewpoint

`vw` and `vh` - these are simply percentages of the viewport width/height

- usually the width and height of the *browser* window
- useful to use if you want to create elements in a webpage that **scale dynamically** to the size of the browser window

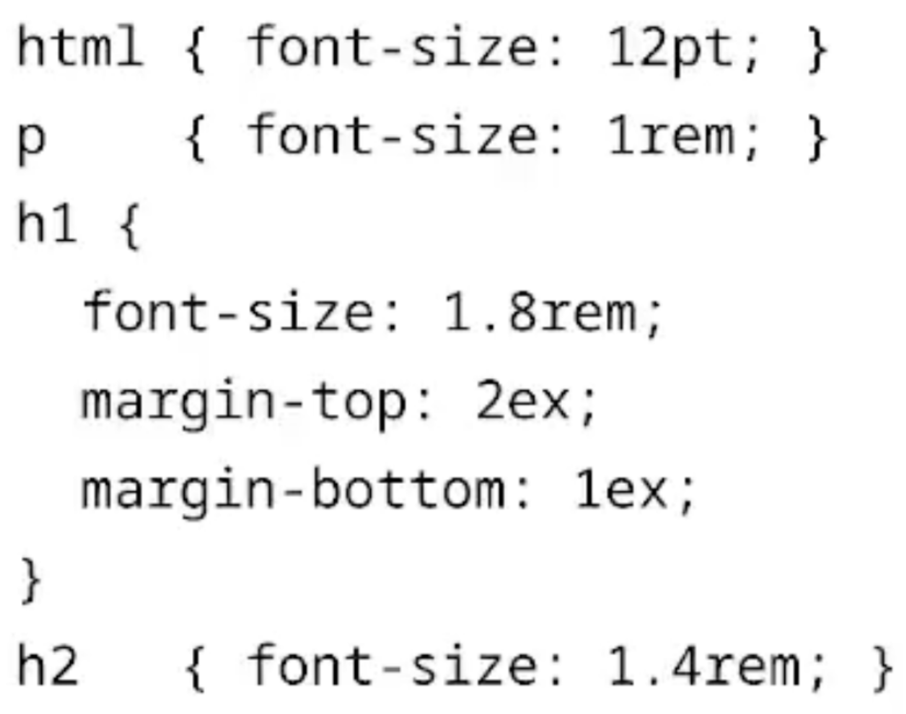
`rem` - this uses a single font size as its value: it uses the font of the **document root (**the html element)

- useful if you want to have elements which scale with the font size with a document, i.e. if someone changes the browser size but you dont want to have different fonts for different areas like with *em* or *ex*

`%` - this measurement is defined in relationship to the parent element width/height

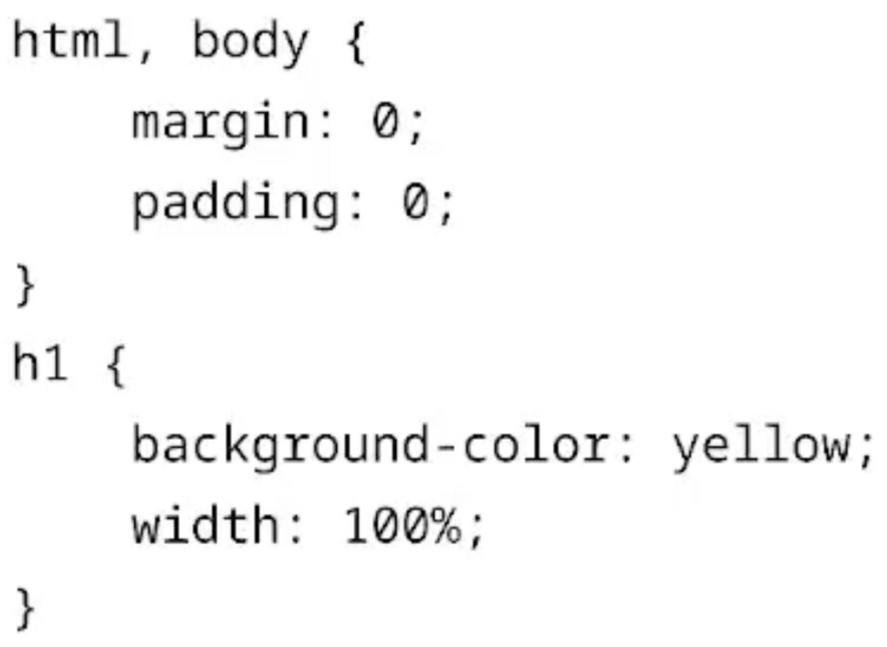
- for example if you have a `div` of height 100px and within that div you have a `<p>` element of a height of `2%` then it will have a height of 2px. If you change the div height then the p height will scale accordingly

Some examples



```
html { font-size: 12pt; }
p    { font-size: 1rem; }
h1 {
  font-size: 1.8rem;
  margin-top: 2ex;
  margin-bottom: 1ex;
}
h2   { font-size: 1.4rem; }
```

- the `1rem` is a default
- for the h1 element we are setting the font size to `1.8rem` which makes the text in `h1` elements font size 1.8x larger than the other text in the document
  - we also set the h1 margain relative to the current font size `2ex` — the heading with have a space above it as large as 2 x's in its own font and a space below in half that size `1ex`
- for the h2 element, this is a smaller heading so we are giving it a size of `1.4rem` — larger than ordinary text but still smaller than the `h1` which makes sense



```
html, body {
    margin: 0;
    padding: 0;
}
h1 {
    background-color: yellow;
    width: 100%;
}
```

- we are setting the margain and padding in the html body to be 0
  - usually we would have to define a unit of measurement except when assigning it to 0, as this would be the same in any unit, e.g. `0px` is the same as `0pt` is the same as `0rem` etc etc
  - in any other case you would **have to define the unit of measurement**
- for the h1 element, we are setting the width of it to be 100% of its parent element, this wont affect the text in th3e h1, but will have an impact on the page construction

---

The CSS Grid is one of the **most favoured** solutions to the age old problem in web design - how should you lay out a website?

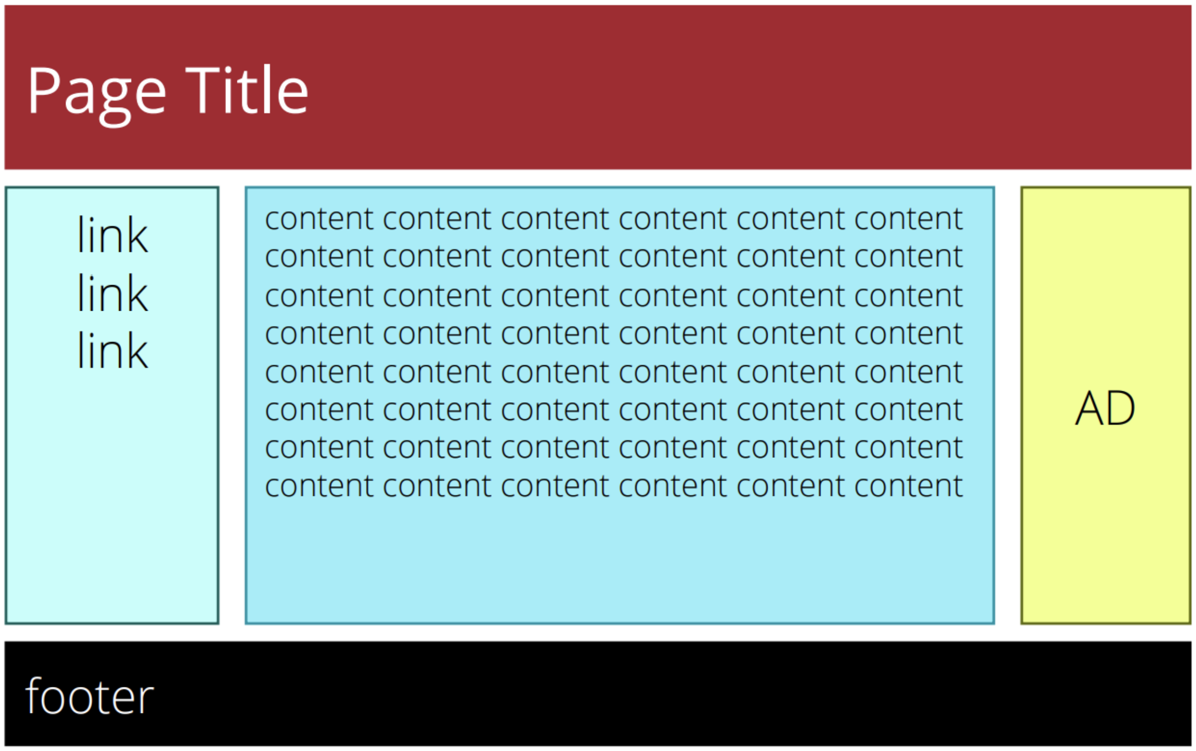
Historically there have been different phases of solutions to this problem:

- starting with tables - mis use of a html table to structure a webpage
- Float -
- flexbox
- Grid - where we are now

Centering challenge

One of the biggest challenges has been centering a block of text on a page which spans more than one line

The Holy Grail of Webpage Layouts

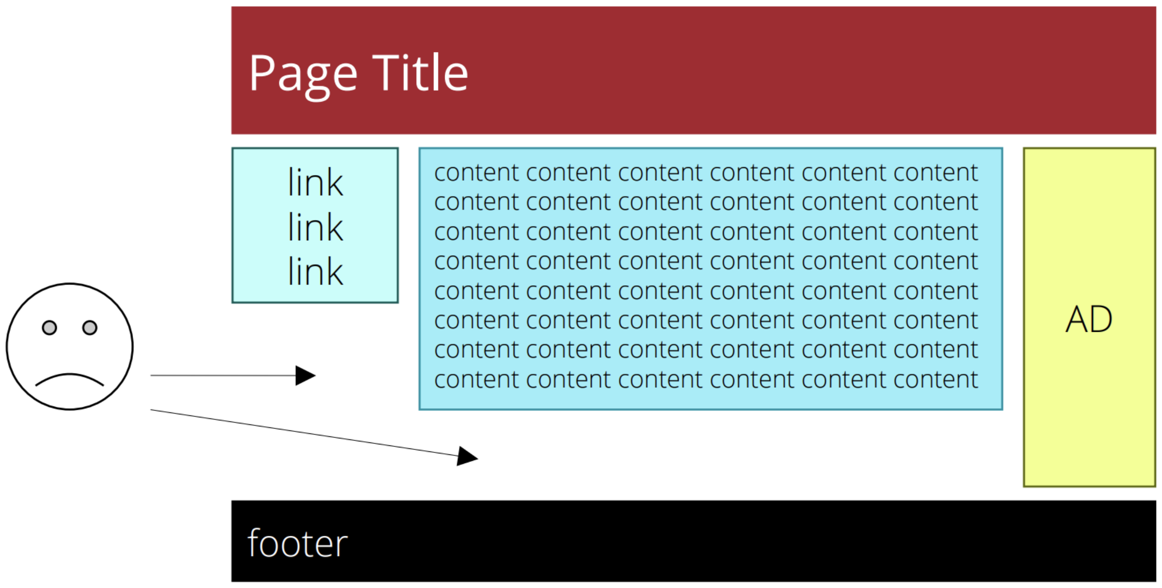

Page Title
link
link
link
content content content content content content
content content content content content content
content content content content content content
content content content content content content
content content content content content content
content content content content content content
content content content content content content
content content content content content content
AD
footer

- links = navigational links
- AD = advertisements

This is a basic and reliable webpage — nothing too crazy

- makes good use of space on the page

However lots of efforts in desigining a webpage such as this fall short, and end up looking something like this:

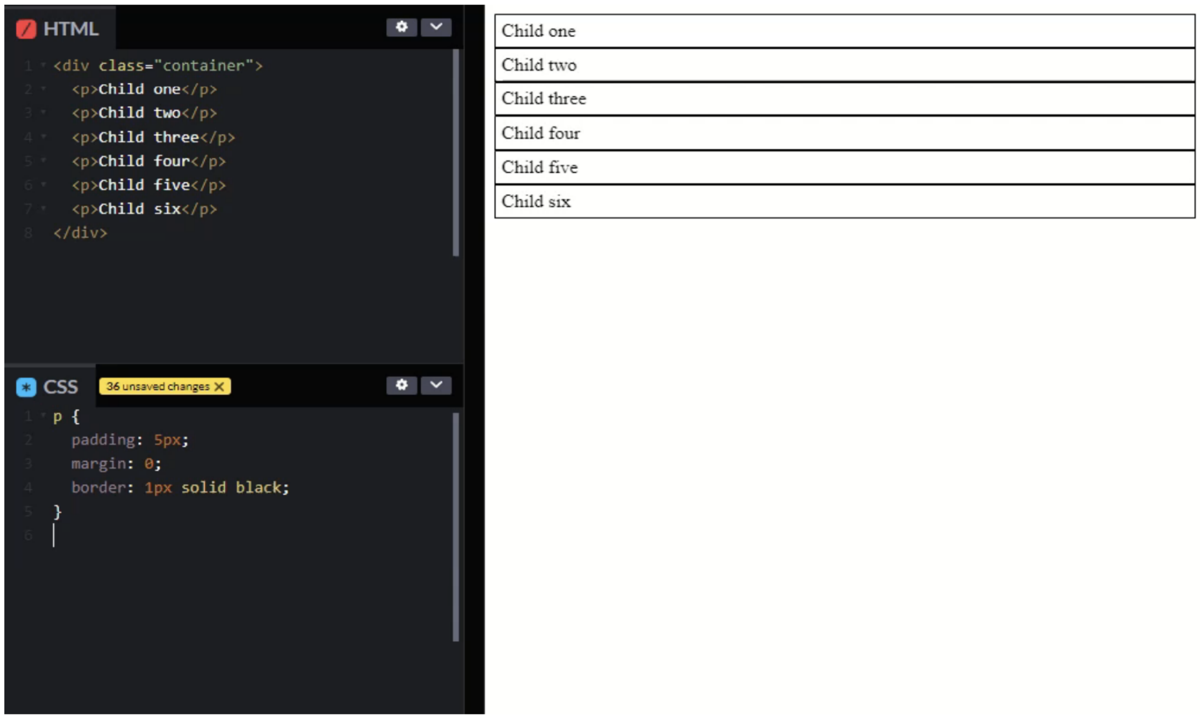


- extra space which appears wasted

How can we use css grids to avoid this, and get tot he holy grail design?

Imagine you have a webpage source like this:

- you have HTML elemt with particular class, and it has a bunch of content you want to layout in a grid
- a rendering is on the right image if you simply drew a border around each element



```html
<div class="container">
  <p>Child one</p>
  <p>Child two</p>
  <p>Child three</p>
  <p>Child four</p>
  <p>Child five</p>
  <p>Child six</p>
</div>
```

```css
p {
  padding: 5px;
  margin: 0;
  border: 1px solid black;
}
```

```
Child one
Child two
Child three
Child four
Child five
Child six
```

Here is the trick:

Achieving a grid layout

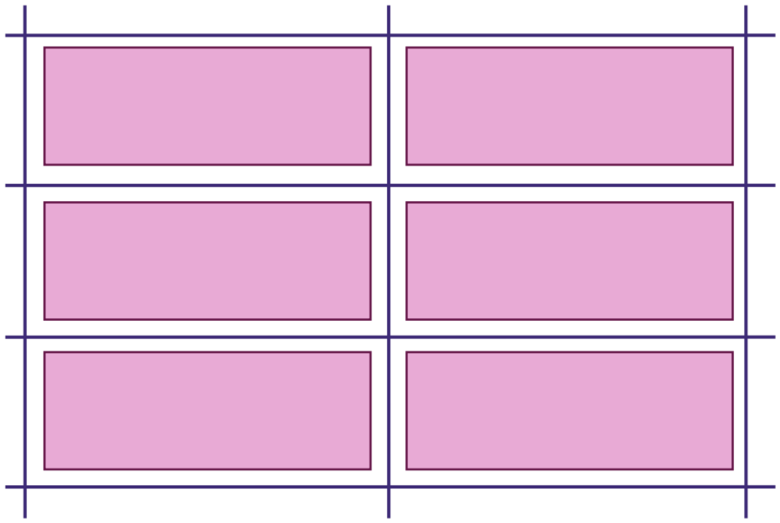
- if you insert a rule for the element which contains other elements (here it is called `container`):

```html
<div class="container">
    <p>Item 1</p>
    <p>Item 2</p>
</div>
```

then if we set a rule fo this class that sets the display mode to grid:

```css
.container {
        display: grid;
    }
```

Will produce a result similar to this:



Grids are broken intop rows and columns, and you can define positions on the grid by referring to the rows and columns

Rows and Columns

- these positions let you set rules for other elements within the grid so they can take up specific amounts of space in the layout
    - for example specify which row and column you want an element to start and end on
    - and there are 3 different ways to specify these values as shown below:

1. **Individually:**

```
.container {
    display: grid;
    grid-row-start: 1;
    grid-row-end: 3;
    grid-column-start: 2;
    grid-column-end: 3;
}
```

2. **Specifying start/end for rows and columns seperately:**

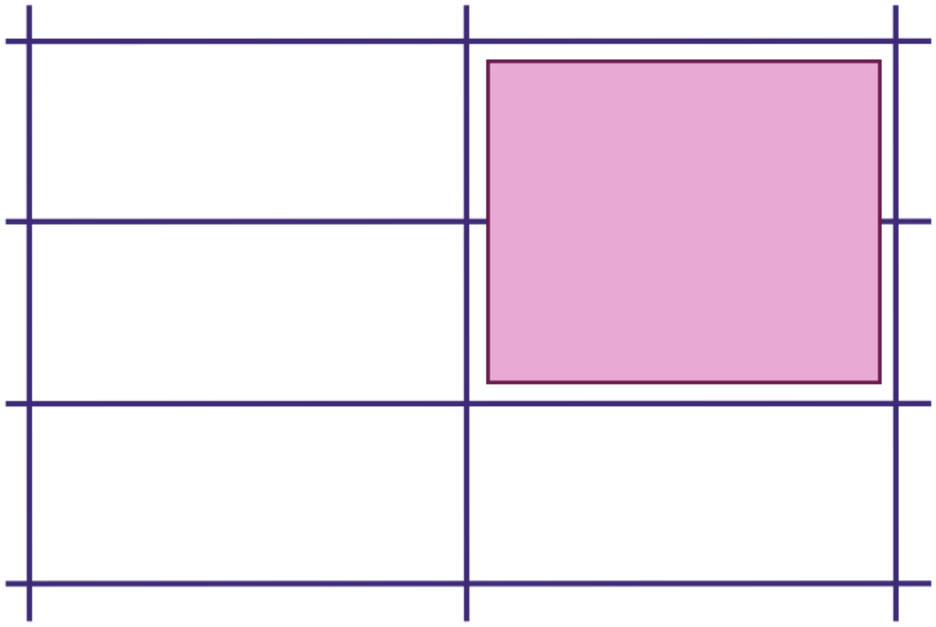
```
.container {
    display: grid;
    grid-row: 1 / 3;
    grid-column: 2 / 3;
}
```

3. **Specifying: start row/start column/end row/end column:**

```
.container {
    display: grid;
    grid-area: 1 / 2 / 3 / 3;
}
```

All three of these options will produce the same rules and will show like this on the grid from earlier:

- starting on the second column, ending on the third colum
- starting on the second row, ending on the third row



`span`

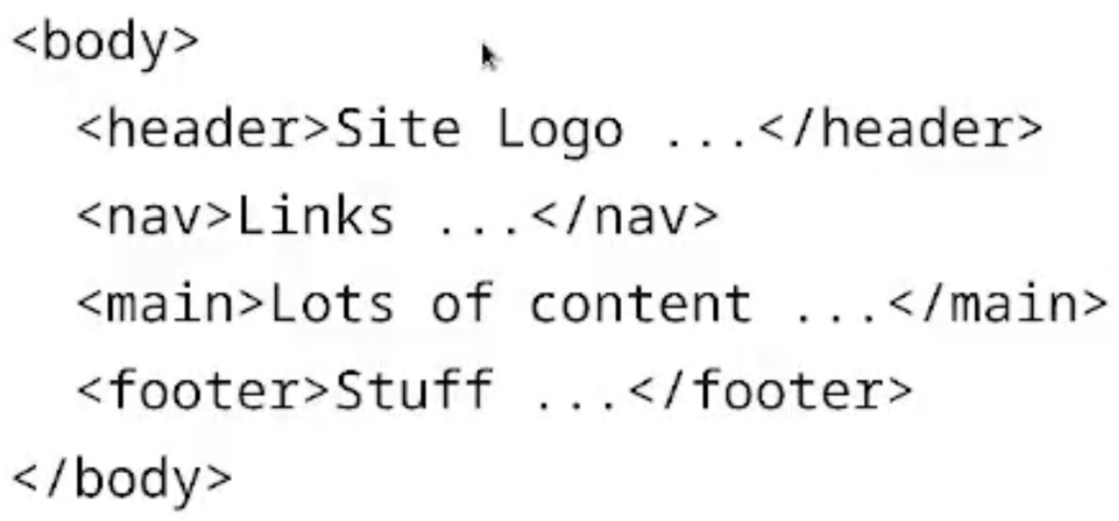
You can also avoid specifying where you want an element to start, and instead just specify how much space you want it to take up

Done by using the `span` keyword

- useful for dynamically adding things to the page but want to control their size in grid terms, you want things to be grid aligned and span a certain amount of space in the grid layout

So how do we achieve the holy grail design making use of the grid?

The HTML webpage will look something like this:

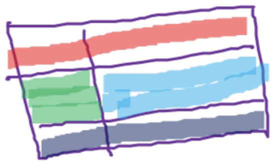


```
<body>
  <header>Site Logo ...</header>
  <nav>Links ...</nav>
  <main>Lots of content ...</main>
  <footer>Stuff ...</footer>
</body>
```

- Set the body display style to grid
- then define the column sizes we will use in the body, in the below example we say we want one column to be 200 pixels (the narrower column in the drawing)
- then use a **special measurement unit for grid use only:** `fr` - this refers to the availabele space within the element that is not already taken up by other columns or fractions of the space
  - `fr` units are distrbuted

```html
<body>
  <header>Sit
  <nav>Links
  <main>Lots
  <footer>Stu
</body>
```

```css
body {
  display: grid;
  grid-template-columns: 200px 1fr;
}
header { grid-row: span 2; }
footer { grid-row: span 2; }
```



the `fr` unit

he `fr` unit in CSS Grid Layout stands for "fraction" and is used to describe a portion of the available space in a grid container. This unit allows you to distribute available space in the grid dynamically based on the fraction values you assign to your grid tracks (columns or rows).

Key Aspects of Using `fr` in Grid Design

1. **Flexible Sizes**: The `fr` unit allocates the remaining free space in the grid container after all fixed-size or content-based sizing is accounted for. This means it is highly responsive and flexible, adapting to the size of the grid container.
2. **Distributing Space**: When you assign `fr` units to grid tracks, you're essentially dividing the space proportionally among those tracks. For example, if you set one column to `1fr` and another to `2fr`, the second column will be twice as wide as the first.

3. **Interaction with Other Units**: The `fr` unit interacts interestingly with other sizing units like pixels (`px`), percentages (`%`), and content-based sizes (`auto`). Fixed-size tracks take up their specified space first, and the remaining space is then divided according to the `fr` units.

Basic Grid with Fractional Tracks

```
.container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
}
```

This sets up a grid with three columns where the middle column is twice the width of the columns on either side, assuming all available space after accounting for any grid gap, padding, or borders.

Grid with Mixed Units

In this layout, the first column is fixed at 100 pixels wide. The remaining space in the grid container is then divided between the second and third columns in a 1:2 ratio.

```
.container {
  display: grid;
  grid-template-columns: 100px 1fr 2fr;
}
```

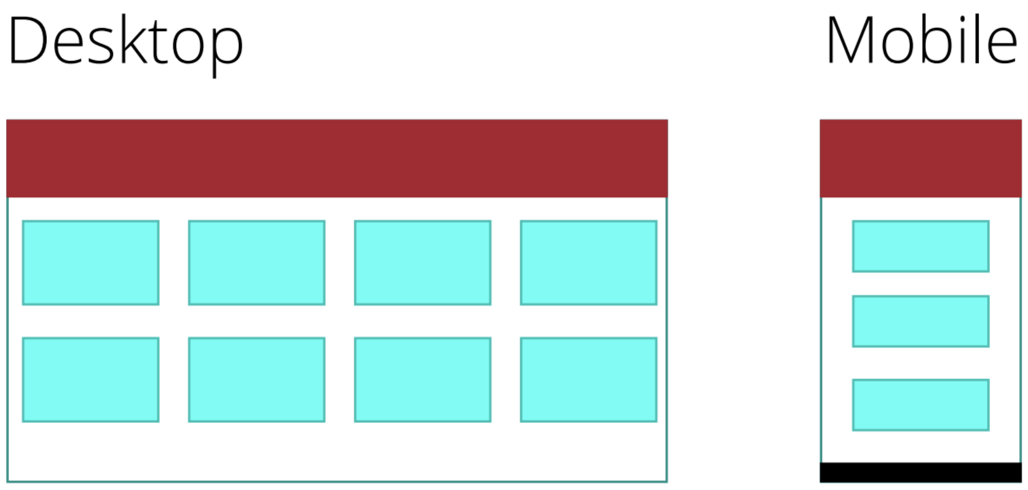
Combining `fr` with Auto and Fixed Sizes

Here, the first column is sized based on its content (`auto`), the second column takes up whatever space is left over, and the third column is fixed at 50 pixels.

```
.container {
  display: grid;
  grid-template-columns: auto 1fr 50px;
}
```

Responsive CSS

You want your websites to be vieweed comfortably on both desktops and mobiles



- two examples of the grid design being used for different devices

General Rules to tailor existing stylesheets for different devices:

Rather than saying we want a page element to be an **absolute size** we instead say we want an element to be **at most** a specifc size

- and if the page is larger then we should increase the margain around the element automatically

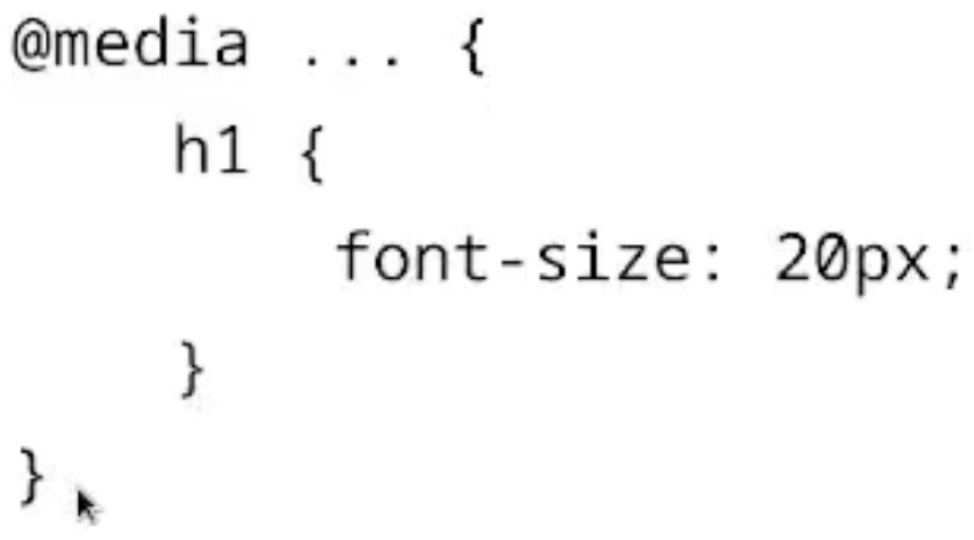
```
main { width: 800px }      // bad implementation this is absolute



main { max-width: 800px;
             margain: 0 auto; } // good implementation
```

You also need to consider how websites will appear to devices such as printers (how will the website appear when printed?)

The solution to this is: **Media Queries**

CSS Media Queries

- They are essentially a block you place around CSS rules prefaced with `@media` and then the type of display you want the content of this block to apply to



```
@media ... {
    h1 {
        font-size: 20px;
    }
}
```

For example you can apply media queries to apply stylesheets when the display **falls within certain bounds:**

```
@media (min-width: 60px)
     and (max-width: 800px) {
  ...
    ...
}
```

You can also set the style sheet to aply to certain media *types* such as screens or printers

```
@media screen

@media print

@media all
```

- `@media all` is a way to say that the styles enclosed in the media query block should be applied universally across all available media types—this includes screens (desktops, tablets, smartphones), printed documents, screen readers, and other output devices.

Media Query syntax

Simple syntax:

```
@media media-type and (media-feature-rule) {
  /* CSS rules go here */
}
```

It consists of:

- A media type, which tells the browser what kind of media this code is for (e.g. print, or screen).
- A media expression, which is a rule, or test that must be passed for the contained CSS to be applied.
- A set of CSS rules that will be applied if the test passes and the media type is correct.

The following media query will only set the body to 12pt if the page is **printed**. It will **not apply** when the page is loaded in a browser:

```
@media print {
    body {
        font-size: 12pt;
    }
}
```

Media Feature Rules

- After specifying the type of media, we can target it with a rule

Width and height

The feature we tend to detect most often in order to create responsive designs (and that has widespread browser support) is viewport width, and we can apply CSS if the viewport is above or below a certain width — or an exact width — using the `min-width`, `max-width`, and `width` media features.

These features are used to create layouts that respond to different screen sizes. For example, to change the body text color to red if the viewport is **exactly 600 pixels**, you would use the following media query:

```
@media screen and (width: 600px) {
    body {
        color: red;
    }
}
```

The `width` (and `height`) media features can be used as ranges, and therefore be prefixed with `min-` or `max-` to indicate that the given value is a minimum, or a maximum. For example, to make the color blue if the viewport is 600 pixels or narrower, use `max-width`:

```
@media screen and (max-width: 600px) {
    body {
        color: blue;
    }
}
```

In practice, using minimum or maximum values is much more useful for responsive design so you will rarely see `width` or `height` used alone.

Orientation

One well-supported media feature is `orientation`, which allows us to test for portrait or landscape mode. To change the body text color if the device is in landscape orientation, use the following media query:

```
@media (orientation: landscape) {
  body {
    color: rebeccapurple;
  }
}
```

Use of pointing devices

As part of the Level 4 specification, the `hover` media feature was introduced. This feature means you can test if the user has the ability to hover over an element, which essentially means they are using some kind of pointing device; touchscreen and keyboard navigation does not hover:

```css
@media (hover: hover) {
  body {
    color: rebeccapurple;
  }
}
```