CSS

Table of Contents

CSS and html.......................................1
Three ways to set styles.......................2
  For each element.............................2
  Using a selector in the web page.....2
  Using a separate style sheet............4
CSS versions.......................................4
Inheritance..4
Selectors...5
  Using class.....................................6
  Styling by ID...................................6
  Styling by page section - div and span...7
Colour...8
Styling text.......................................9
  Font issues....................................9
  Text styles....................................10
The box model..................................11
Containers..12
Layout...13
  Flexbox...13
  Grid...14
A pixel is not a pixel.........................17
Media queries...................................21

Try out the examples, and explore your own. You need to know html.

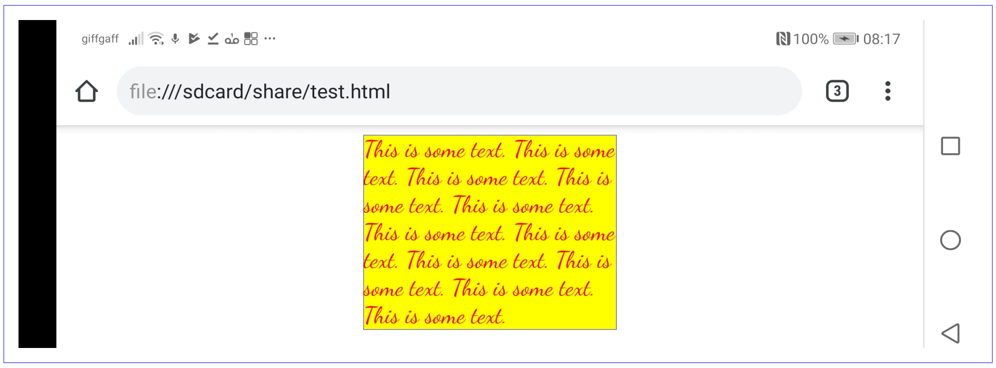
CSS and html

html is used to say what content is in a web page - what text, images and links.

CSS - Cascading Style Sheets - is used to control the *appearance* of the content. That means colours, fonts, layout, borders, margins and so on.

Example of CSS

```
<!DOCTYPE html>
<html lang="en">

<head>
<title>Hello</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
</head>

<body>

<p style =
"color: red;
background: yellow;
border: thin blue solid;
font-family: cursive;
font-size:14pt;
width: 200px;
height: 200 px;
margin:auto;
">
This is some text. This is some text. This is some text.
This is some text. This is some text. This is some text.
This is some text. This is some text. This is some text.
This is some text. This is some text. </p>
</body>

</html>
```



This is some text. This is some text. This is some text. This is some text. This is some text. This is some text. This is some text. This is some text.

so a style setting is a pair, of an attribute and a value. For example

```
background: yellow;
```

sets the background colour to be yellow.

Three ways to set styles

For each element

We used this in the example. As

```
<p style =
"
color: red;
background: yellow;
..
">
.. </p>
```

That sets the style of *that element*, and nothing more. It does not affect any other paragraph.

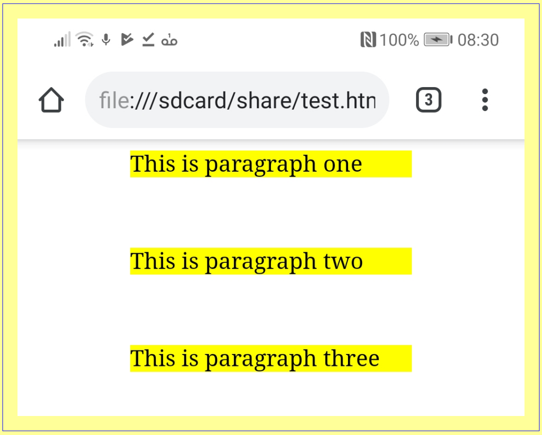
This is *not* a good way to do it. We will have inconsistent appearances. Each part of the web page will look different, unless we copy the style to every element. If we want to change it, we must change every copy.

Using a selector in the web page

For example

```
<!DOCTYPE html>
```

```
<html lang="en">

<head>
<title>Hello</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
p
{
background: yellow;
width: 200px;

margin:auto;
margin-bottom: 50px;
}
</style>
</head>

<body>

<p>
This is paragraph one </p>
<p>
This is paragraph two </p>
<p>
This is paragraph three </p>
</form>
</body>
```



This is paragraph one

This is paragraph two

This is paragraph three

Now instead of setting styles in one element, we set them in a styles element in the head element:

```
<head>
..
<style>
..
</style>
</head>
```

In the style element, use a *selector* to pick out a set of one or more elements that the style should apply to. Here, our selector is simply p

```
p
{
..
}
```

That means the styles apply to every p element through the page.

The rules about the selector are very flexible. We can apply it to one particular element (by id) or some of them (by class) or some in particular parts of a page (by div or span).

Using a separate style sheet

We put the styles in a separate file, and link the web page to it. So our web page says:

```
<!DOCTYPE html>
<html lang="en">

<head>
..
<link rel="stylesheet" type="text/css" href="styles.css">
</head>
.. as before..

</html>
```

and in a file named styles.css we have

```
p
{
background: yellow;
width: 200px;

margin:auto;
margin-bottom: 50px;
}
```

This is the best way to do it for a real website. This is because all the pages in the web site can use the same style sheet, and if we edit that single file, the entire website appearance changes, simply and consistently.

CSS versions

Like html, CSS is agreed by W3C, the Worldwide Web Consortium, and there are different versions. Version 3 is in development. This is split into a set of modules covering different topics, and these are in various stages of development.

This text is based on version 2.1, which is here:

https://www.w3.org/TR/CSS2/

The Mozilla documents on CSS are here

https://developer.mozilla.org/en-US/docs/Web/CSS

Inheritance

We often have elements inside other elements. For example, all headings and paragraphs are inside the body element.

In that case the inside element style is usually *inherited* from the enclosing element, unless it has a new value set.

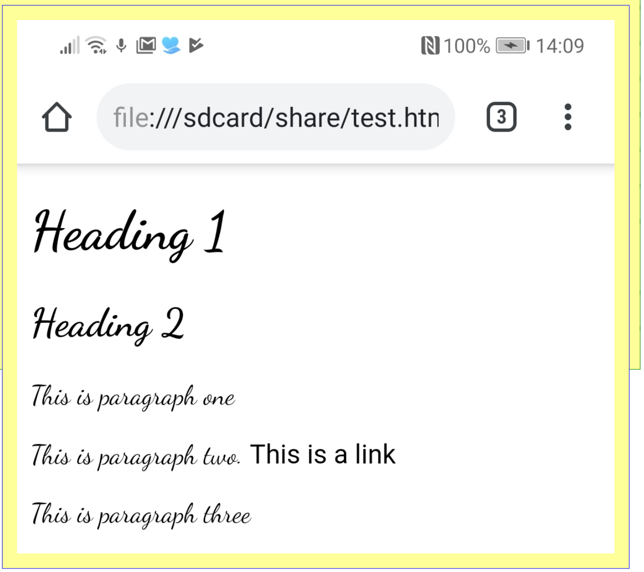
For example this CSS

```
body
{
font-family:cursive;
}

a
{
font-family: mono;
}
```

and this html

```
..
<link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<p>
This is paragraph one </p>
<p>
This is paragraph two. <a>This is a
link</a> </p>
<p>
This is paragraph three </p>
```

produces:



100% 14:09

file:///sdcard/share/test.htn

Heading 1

Heading 2

This is paragraph one

This is paragraph two. This is a link

This is paragraph three

The font for the body element is set to be cursive. The h1 h2 and p elements are inside the body, and so inherit that font. But the <a> element has a different font defined, and this overrides the inherited value.

Selectors

In a style rule like

p { color: red }

The p is a selector - it selects what the rule applies to. This rule applies to p elements.

But we can make selectors in different ways

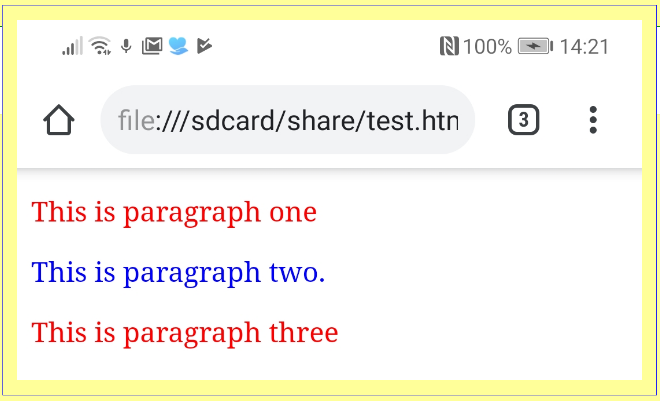
Using class

An html element can have a class attribute value. Then we can style by class. For example html

```
<p class="redP">
This is paragraph one </p>
<p class="blueP">
This is paragraph two. </p>
<p class = redP>
This is paragraph three </p>
```

and style:

```
.redP
{
color: red;
}

.blueP
{
color: blue;
}
```



This is paragraph one

This is paragraph two.

This is paragraph three

So in a style sheet, something like .name is a class.

Styling by ID

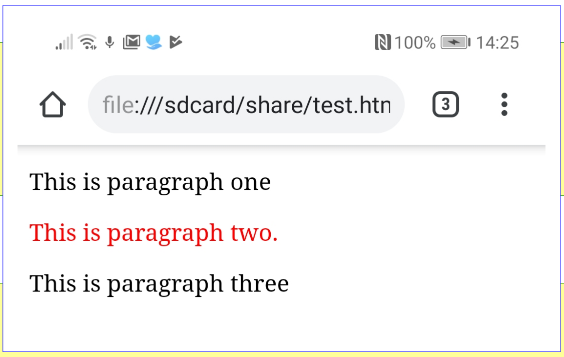
We can give an html element a unique ID. This should apply to only one element. We must not have two elements with the same ID. This is because we use IDs a lot in Javascript, and this will not work if two elements have the same id.

For example

```
#redP
{
color: red;
}
```

and

```
<p>
This is paragraph one </p>
<p id="redP">
This is paragraph two. </p>
<p>
This is paragraph three </p>
```



This is paragraph one

This is paragraph two.

This is paragraph three

So #name in a style sheet is an ID selector

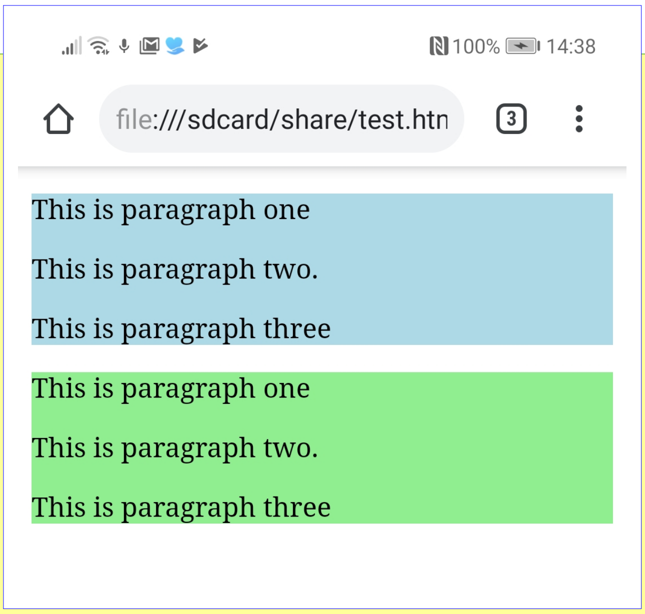
Styling by page section - div and span

A div is used to mark a part - a division - of a web page, so it can be given a class or id. Then that part can be styled. For example

```
.greenBack
{
background: lightgreen;
}
.blueBack
{
background: lightblue;
}
```

and

```
<div class="blueBack">
<p>
This is paragraph one </p>
<p>
This is paragraph two. </p>
<p>
This is paragraph three </p>
</div>
<div class="greenBack">
<p>
This is paragraph one </p>
<p>
This is paragraph two. </p>
<p>
This is paragraph three </p>
</div>
```



This is paragraph one

This is paragraph two.

This is paragraph three

This is paragraph one

This is paragraph two.

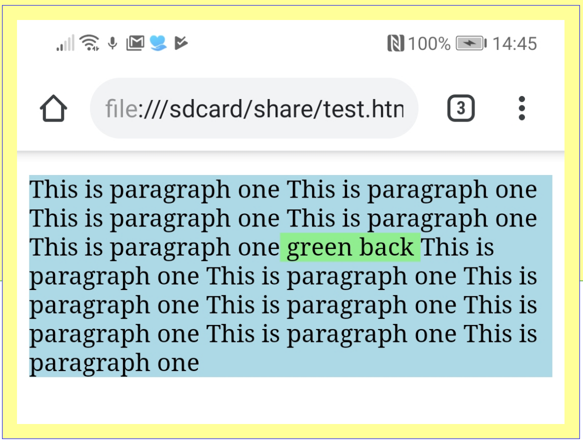
This is paragraph three

We have set up two divs, and given them different classes with different coloured backgrounds.

Or we can use spans. Using the same styles:

```
<p class="blueBack">
This is paragraph one This is paragraph one This is paragraph one This is
paragraph one This is paragraph
one<span class="greenBack"> green back
</span> This is paragraph one This is
paragraph one This is
paragraph one This is paragraph one
This is paragraph one This is
paragraph one This is
paragraph one </p>
```

we get



This is paragraph one This is paragraph one This is paragraph one This is paragraph one This is paragraph one green back This is paragraph one This is paragraph one This is paragraph one This is paragraph one This is paragraph one This is paragraph one This is paragraph one html elements are mostly displayed *block* or *inline*. Block elements are displayed down the page, one below the other. Inline elements are displayed across the page.

Paragraphs and headings and divs are block. Links and spans are inline.

So

```
<p class="blueBack">
This is paragraph one This is paragraph one This is paragraph one This is
paragraph one This is paragraph one<span class="greenBack"> green back </span>
This is paragraph one This is paragraph one This is
paragraph one This is paragraph one This is paragraph one This is paragraph one
This is paragraph one </p>
```

Colour

Most elements can be given a foreground colour, using the word color, and a background colour, as background.

The colour can be written in many different ways. This link gives full details.

In rgb, the colour is made up of a red, a green, and a blue part. Each part can be written as a whole number, from 0 to 255. So

rgb(255,0,0) is pure red

rgb(255,255,0) is red+green = yellow

rgb(0,0,0) is black

rgb(255,255,255) is white

rgb(30,30,60) is blueish gray

So for example

```
p
{
color: rgb(0,0,255);
}
```

Or use colour names:

```
p
{
color: hotpink;
}
```

Use that link to check allowed colour names, and the other ways to specify colour, including transparency.

In the old days the colours which browsers could display were limited - these were called websafe colours. Modern browsers will display anything.

Usually on a website you want a palette of three or four related colours. There are websites which let you choose these, such as this one.

Styling text

This is done either by using fonts or text properties.

Font issues

A basic rule would be like

```
p { font-family: Arial; }
```

But a browser can usually only use a font which is installed on that device - and different devices have different fonts installed. The @fontface rule enables dynamic download of a font from a URL.

So usually we provide a list of fonts, like

```
font-family: Helvetica, Arial, sans-serif;
```

Here the browser will use Helvetica, if it is installed. If not, it will use Arial, and if not that, a generic san-serif font. There are five *generic fonts* -

serif - font which has serifs, like Times New Roman

sans-serif - no serifs, like Arial

monospace - each character has the same width, like `this - Andale Mono`

cursive - like handwriting such as this one - Comic Sans

fantasy - decorative such as this one - 1298 Ferrenicus II

You do not get these specific fonts - you get something like them, varying from device to device. The generic fonts are a fall-back choice.

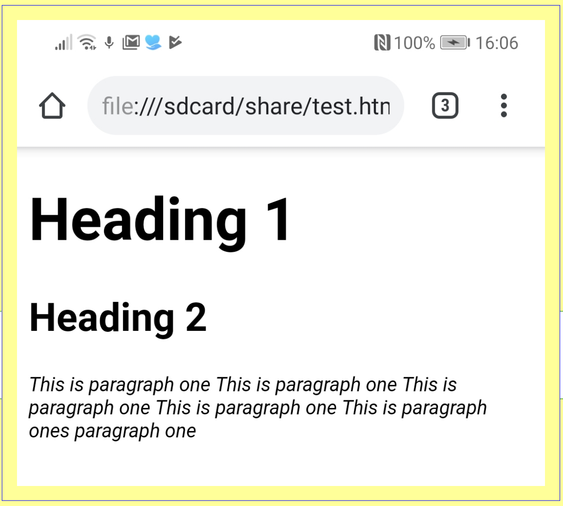
font-size fixes the font size. You can give that in pixels px, or points pt, or mm. An em is very useful - it means relative to the font size of the enclosing element. So for example 2em is twice normal size.

font-weight is bold, or not

font-style is italic or normal

So for example

```
body
{
font-family: Helvetica, Arial, sans-serif;
font-size: 10pt;
}

h1
{
font-size: 3em; // 3 times normal
font-weight: bold;
}

h2
{
font-size: 2em;
}

p
{
font-style: italic;
}
```



file:///sdcard/share/test.htn

Heading 1

Heading 2

This is paragraph one This is paragraph one This is paragraph one This is paragraph one This is paragraph ones paragraph one

and

```
<h1> Heading 1</h1>
<h2>Heading 2</h2>
<p >
This is paragraph one This is paragraph one This is paragraph one This is
paragraph one This is paragraph ones
paragraph one </p>
```

Text styles

These include -

text-decoration : underline, overline, line-through

text-shadow : horiz vert blur color. horiz and vert are how far the shadow is from the text, blur is the blur radius, and colour is the colour

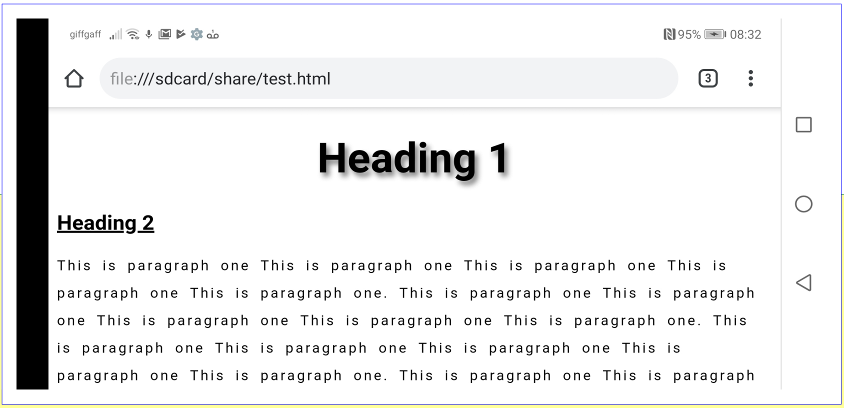
text-align: left or right or center or justify

line-height - vertical distance between lines of text eg 2 gives double spacing

letter-spacing

word-spacing

For example



file:///sdcard/share/test.html

Heading 1

Heading 2

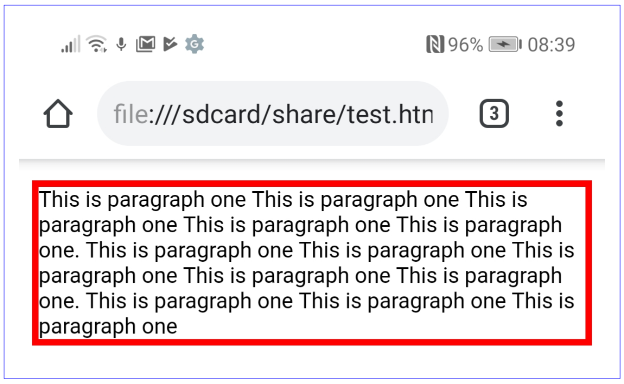
This is paragraph one This is paragraph one This is paragraph one This is paragraph one This is paragraph one. This is paragraph one This is paragraph one This is paragraph one This is paragraph one This is paragraph one. This is paragraph one This is paragraph one This is paragraph one This is paragraph one This is paragraph one. This is paragraph one This is paragraph

```
body
{
font-family: Helvetica,
Arial, sans-serif;
font-size: 10pt;
}
```

```
h1
{
font-size:3em;
text-align: center;
text-shadow: 3px 4px 5px gray;

}

h2
{
text-decoration: underline;
}

p
{
letter-spacing: 2px;
word-spacing: 4px;
line-height: 2;
}
```

The box model

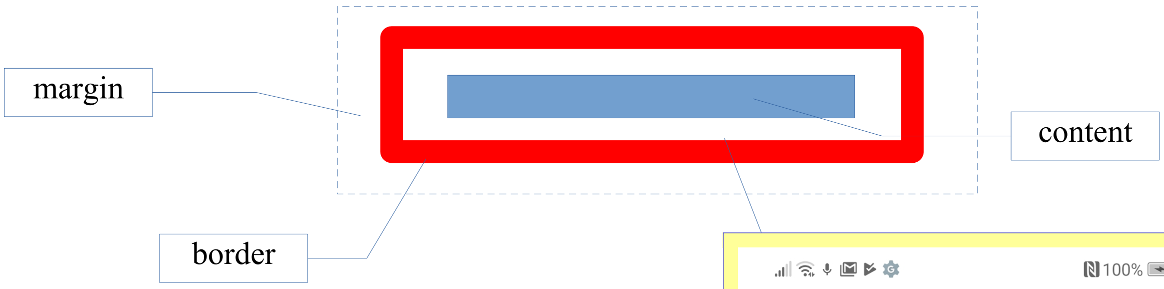
We start with a *border* - elements often have a border:



This is paragraph one This is paragraph one This is paragraph one This is paragraph one This is paragraph one. This is paragraph one This is paragraph one This is paragraph one This is paragraph one This is paragraph one. This is paragraph one This is paragraph one This is paragraph one

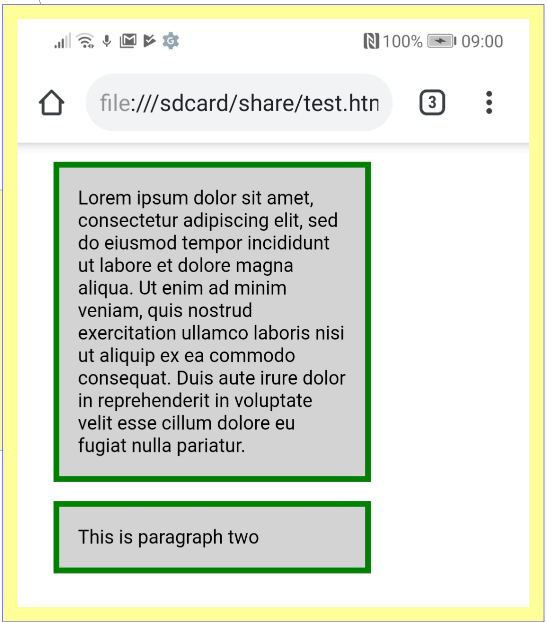
But we might want a space between the border and the text - this is called *padding*. We might want to fix the top bottom left right padding.

And we might want some space outside the border, between this element and those to the left or right, above and below. This space is a *margin*.



For example:

```
p
{
background: lightgray;
padding: 1em;
border: 4px green solid;
margin-left: 5%;
margin-right:30%;
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

This is paragraph two

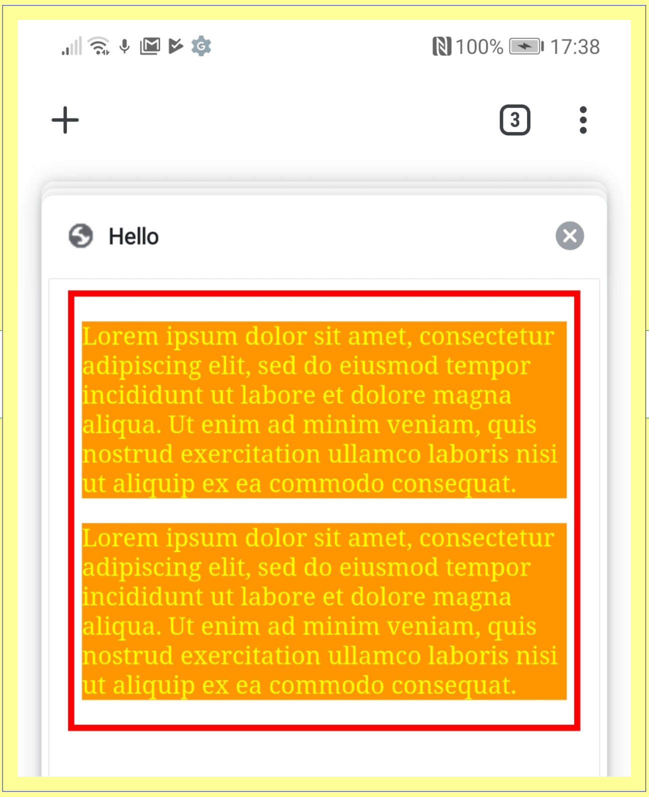
```
}
```

The 'Lorem ipsum dolor..' is Latin. The idea is that this is just some text used as an example, and you should ignore what it says.

Containers

One container, such as a section, might contain other elements, such as articles. For example

```
..
<body>

<section>
  <article>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
    eiusmod tempor incididunt ut labore et dolore magna aliqua.
    Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
    nisi ut aliquip ex ea commodo consequat.
    </p>
  </article>
  <article>
        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
    eiusmod tempor incididunt ut labore et dolore magna aliqua.
    Ut enim ad minim veniam, quis
nostrud exercitation ullamco
laboris
    nisi ut aliquip ex ea commodo
consequat.
    </p>
  </article>

</body>
..
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

and

```
article
{
margin: 5px;
background: gold;

}

section
{
border: 4px red solid;
margin: 5px;
}

p
{
```

```
background: rgba(255,0,0,0.3);
color: yellow;
}
```

Here there is a section, inside the body element. Inside the section we have two articles, each of which contain a paragraph.

An element will inherit the styles of its containing element, unless over-ridden.

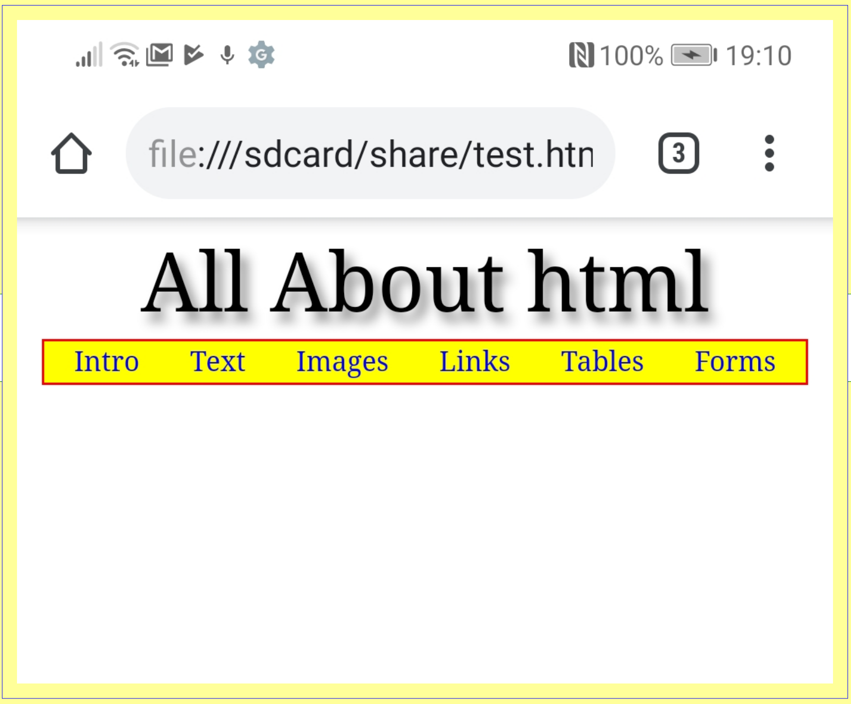
Layout

Styles concern two main things - what things look like (colour, font and so on), and where they are laid out - the layout. Layout used to be tricky, using a lotof work-arounds to get the desired layout, working in all browsers. It is easier with modern CSS.

Flexbox

A flexbox is a set of elements. These are usually horizontal, in a row, but it can be a column. For example

```
<header>All About html</header>
<nav>
<a href="#">Intro</a>
<a href="#">Text</a>
<a href="#">Images</a>
<a href="#">Links</a>
<a href="#">Tables</a>
<a href="#">Forms</a>
</nav>
</body>
```



and

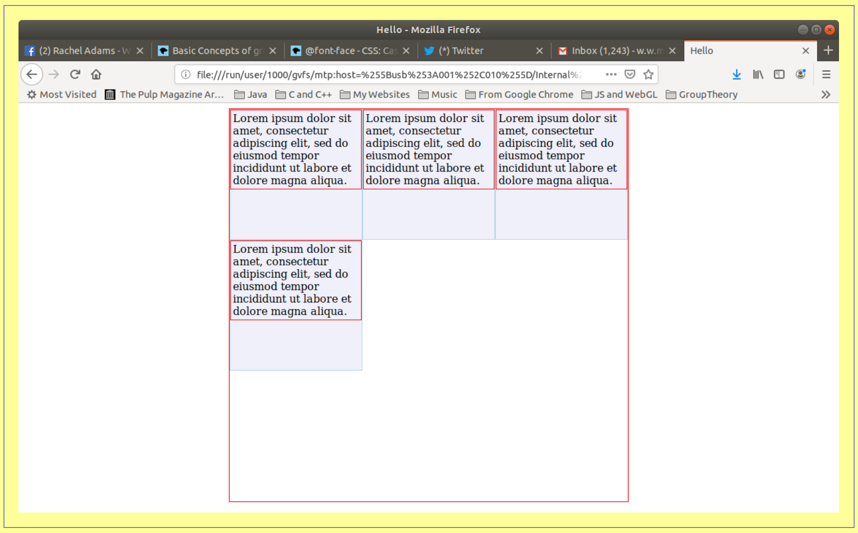
```
header
{
font-size:3em;
text-align: center;
text-shadow: 3px 4px 5px
gray;
}

nav
{
display: flex; /* in a row */
align-items: center; /* center vertically */
justify-content: space-around; /* equal spacing horizontally */
border: 1px red solid;
background: yellow;
margin: 3px;
padding: 2px;
}
```

```
a
{
color: blue;
margin-left:4px;
margin-right:4px;
text-decoration: none;
}
```

Grid

A grid is used for a grid layout:

```
section
{
  display: grid;
  grid-template-columns: 200px 200px 200px;
  grid-template-rows: 200px 200px 200px;
  border: thin red solid;
  width: 600px;
  margin: auto;
}
article
{
border: 1px lightblue
solid;
background:
rgb(240,240,250);
padding:0;
}
p
{
border: thin red solid;
margin:0;
padding: 3px;
}
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

and

```
<body>

<section>
<article> <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
</article>
<article> <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
</article>
<article> <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
</article>
<article> <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
</article>
```

```
</section>
</body>

</html>
```

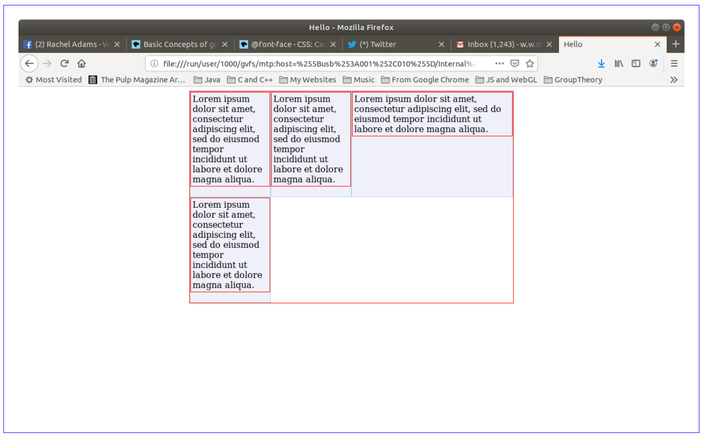
The number and size of the rows and columns is controlled by the

```
  grid-template-columns: 200px 200px 200px;
  grid-template-rows: 200px 200px 200px;
```

These do not need to be equal. We can also use the fr unit:

```
  grid-template-columns: 1fr 1fr 2fr;
  grid-auto-rows: 200px;
```

The fr is a fractional part. Here we have 1+1+2 parts, so they are split in proportion ¼, ¼ and ½. auto-rows means the rows are 200px high, and the number will be as many as are required for the contents:



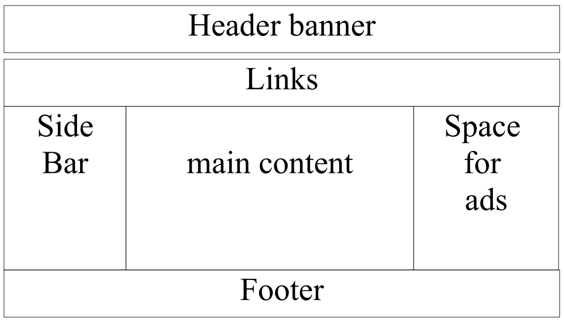
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

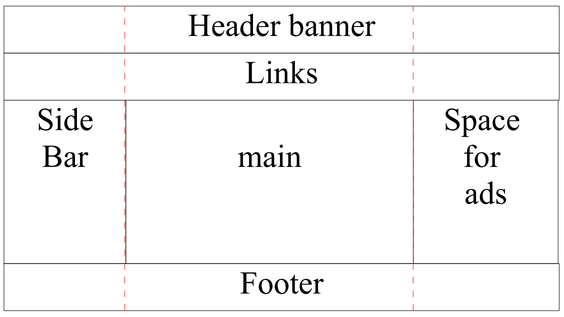
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Elements can cover several cells, and start and end wherever we want. This means we can create almost any layout if it is basically grid-like. For example to make:



This has 6 components.

This is really a 3 by 4 grid, with components covering several cells:



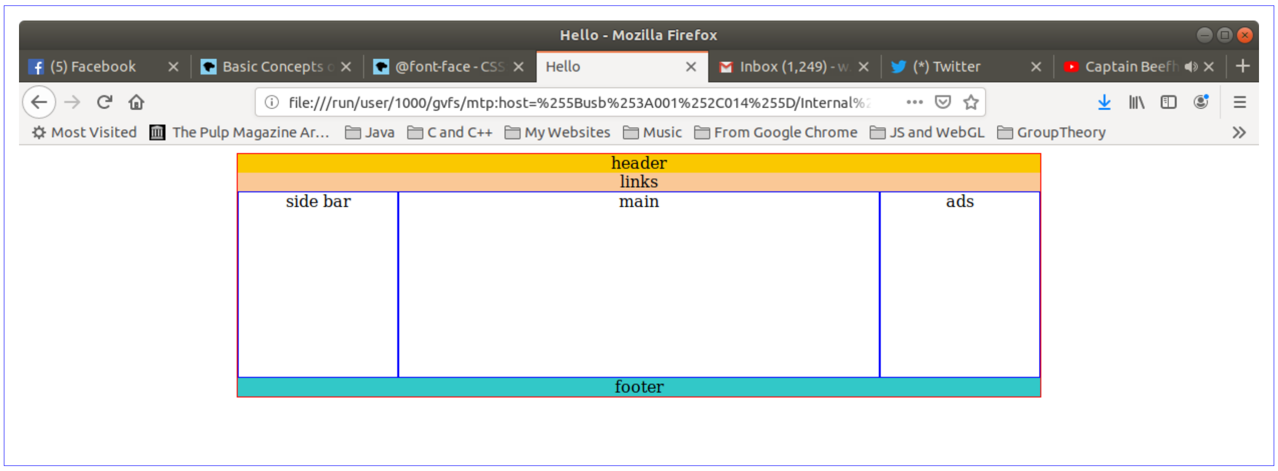
For each component we can say which column it starts and ends - but in fact we say the 'track' = line number. So links for example starts 1 and ends 4:

```
 <div class="wrapper">
  <header>header</header>
  <nav>links</nav>
  <div class="side">side bar</div>
  <div class="main">main</div>
  <div class="ads">ads</div>
  <footer>footer</footer>
</div>
```

and

```
.wrapper {
  display: grid;
  grid-template-columns: 1fr 3fr 1fr;
  grid-template-rows: 1fr 1fr 10fr 1fr;

  text-align: center;
  border: thin red solid;
  width: 800px;
  margin: auto;
}
div
{
border:thin blue solid;
}

header {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 1;
  background: rgb(250,200,0);
}

nav
{
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 2;
  grid-row-end: 3;
   background: rgb(250,200,150);
}
.side
{
  grid-column-start: 1;
  grid-column-end: 1;
  grid-row-start: 3;
  grid-row-end: 3;
}
.main
```

```
{
  grid-column-start: 2;
  grid-column-end: 3;
  grid-row-start: 3;
  grid-row-end: 4;
}
.ads
{
  grid-column-start: 3;
  grid-column-end: 4;
  grid-row-start: 3;
  grid-row-end: 4;
}
footer
{
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 4;
  grid-row-end: 5;
  background: rgb(50,200,200);
}
```



A pixel is not a pixel

Actually - things are not that simple. Pixels are not pixels.

Suppose we have a very simple webpage:

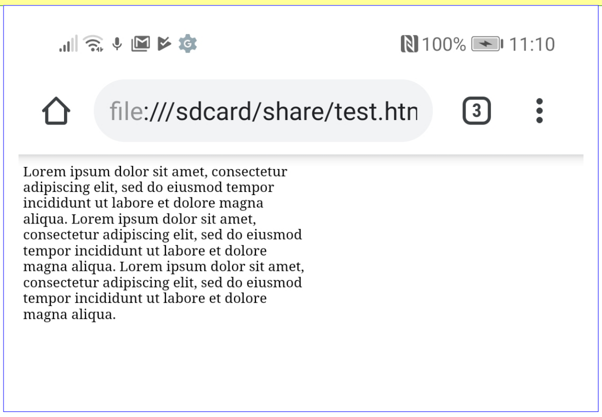
```
<!DOCTYPE html>
<html lang="en">

<head>
<title>Hello</title>
<meta charset="UTF-8">
<!-- <meta name="viewport" content="width=device-width, initial-scale=1"> -->
<link rel="stylesheet" type="text/css" href="styles.css">
</head>

<body>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
```

```
do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua.
</p>
</body>

</html>
```

We have commented out the viewport setting. And

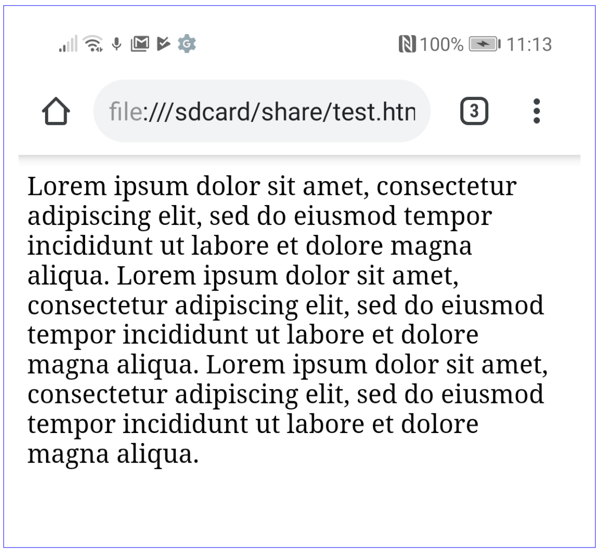
```
p
{
width: 500px;
}
```

On a smartphone it looks like this - pretty hard to read:



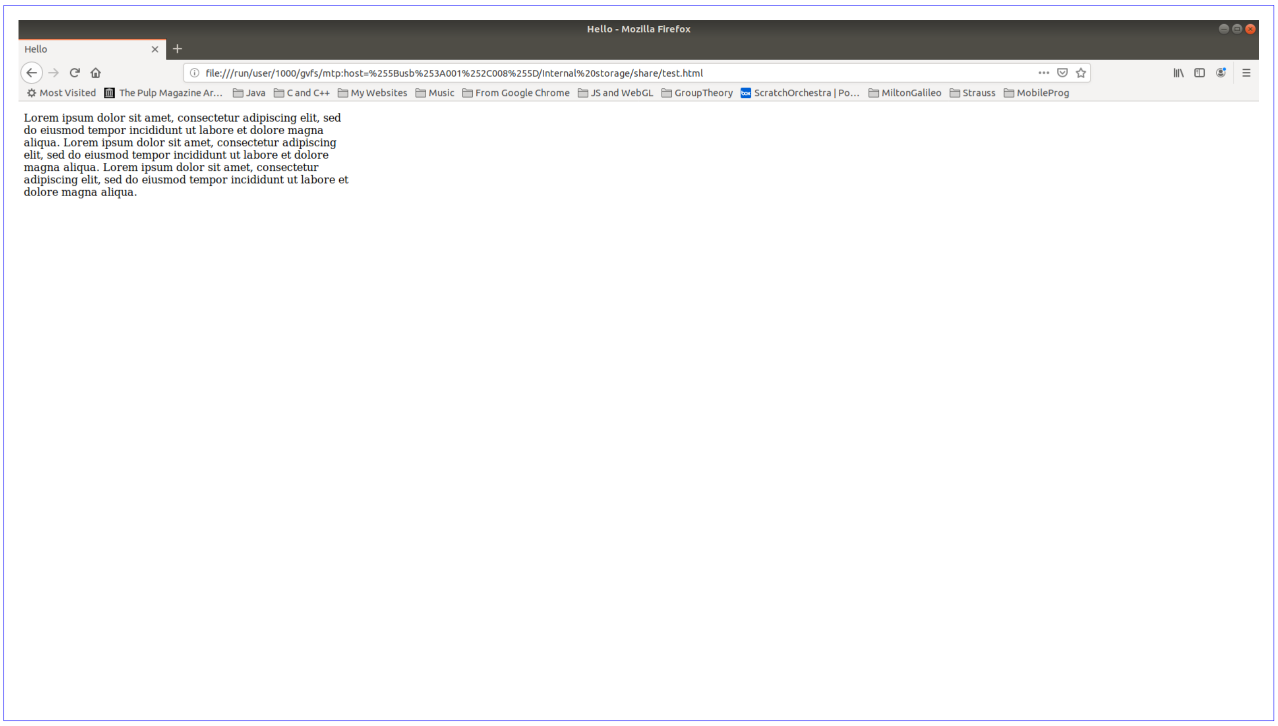
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

But we can zoom it, to look like this. Much more readable - but still 500 px wide, according to the page.



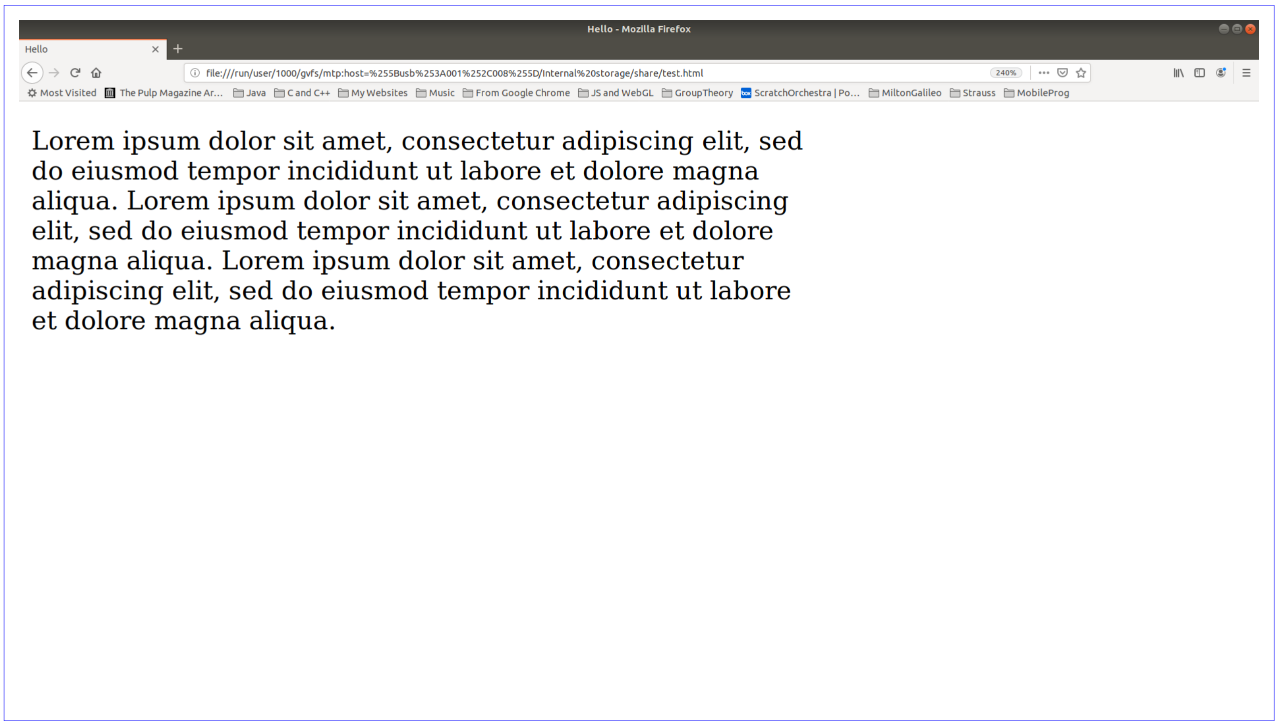
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

On a desktop browser it looks like



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

but if we do ctrl + a few times, we get



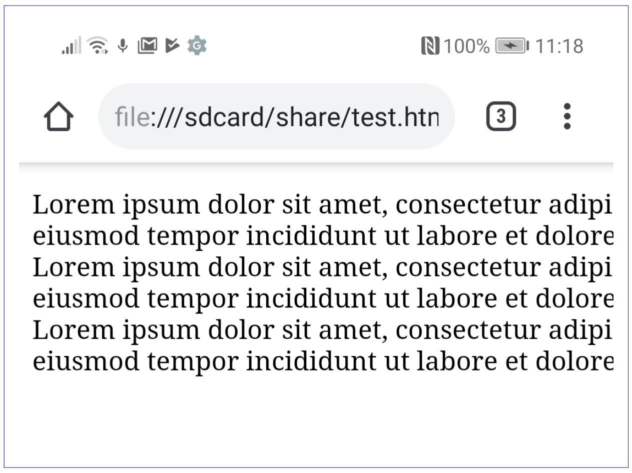
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Still 500px wide, by the web page.

Now put the viewport back in:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

and on the phone, see:



Lorem ipsum dolor sit amet, consectetur adipi
eiusmod tempor incididunt ut labore et dolore
Lorem ipsum dolor sit amet, consectetur adipi
eiusmod tempor incididunt ut labore et dolore
Lorem ipsum dolor sit amet, consectetur adipi
eiusmod tempor incididunt ut labore et dolore

Therefore when we say

```
width: 500px;
```

the px are not the actual *physical pixels* on the device - because they are a fixed size.

In fact they are some size related to physical pixel size, scaled by some factor. That scale factor is altered by

1. The html viewport meta tag

2. On a phone, the user zooming the display

3. On a laptop or desktop, the user doing ctrl+ or ctrl- to zoom the browser

Two more factors on a laptop or desktop

4. The user may change the screen resolution. They might choose higher resolution to get sharper images, but also get smaller images, so they need better eyesight

5. The browser window may not be maximised. The user may have re-sized it to see several windows at once. This does not change the resolution, but changes the window dimensions in pixel count terms.

So the purpose of

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

is to get a good scaling of physical pixels to CSS px units.

What is the target?

To design web pages which are readable on all devices, without adjustment by the user.

Some options are using %, like

```
p
{
width: 50%;
}
```

then our paragraph will be half the screen width. Only problem is that a big paragraph will be very tall, and require scrolling.

Or in ems

```
 p
{
width: 15em;
}
```

then our paragraph is 15 average character width. Again this might produce very tall paragraphs.

A bad plan is

```
p
{
font-size: 8pt;
}
```

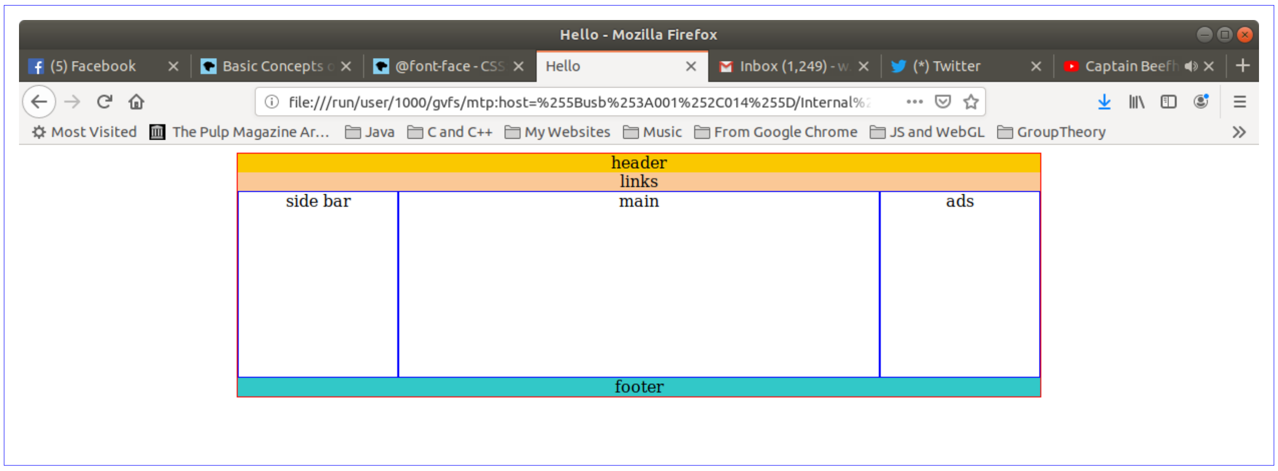
because very small text will probably force the user to zoom to read it.

In practice it is usually impossible to get styling which will work both on very small phone screens and big desktops. We need our web page to be able to use different styles on different types of device.

Media queries

In 2019 around half of website traffic is from mobile phones. It is essential to write web pages that work both on mobiles and desktops. Which is what @media is for.

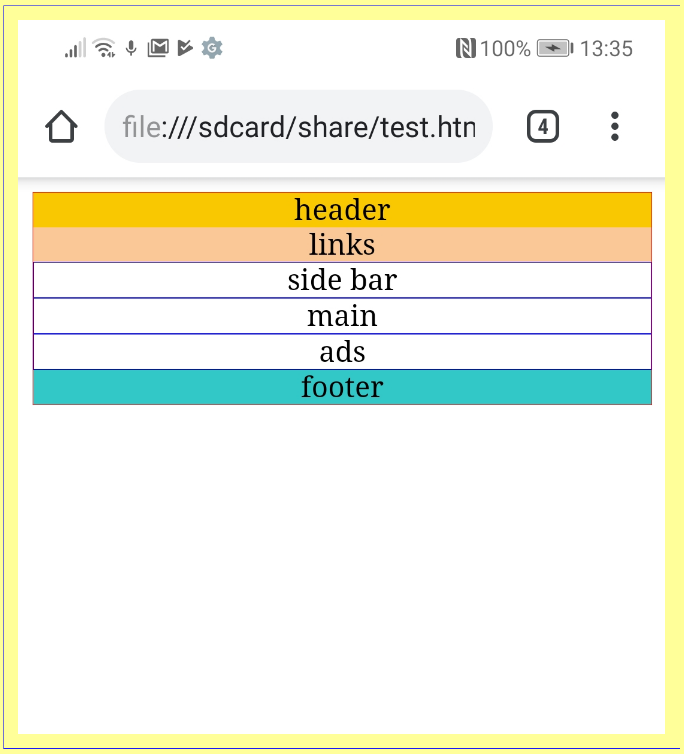
As an example, look at the page producing this



This is too wide to fit a phone, unless the text is made extremely small. We leave the html unchanged and use an altered style sheet:

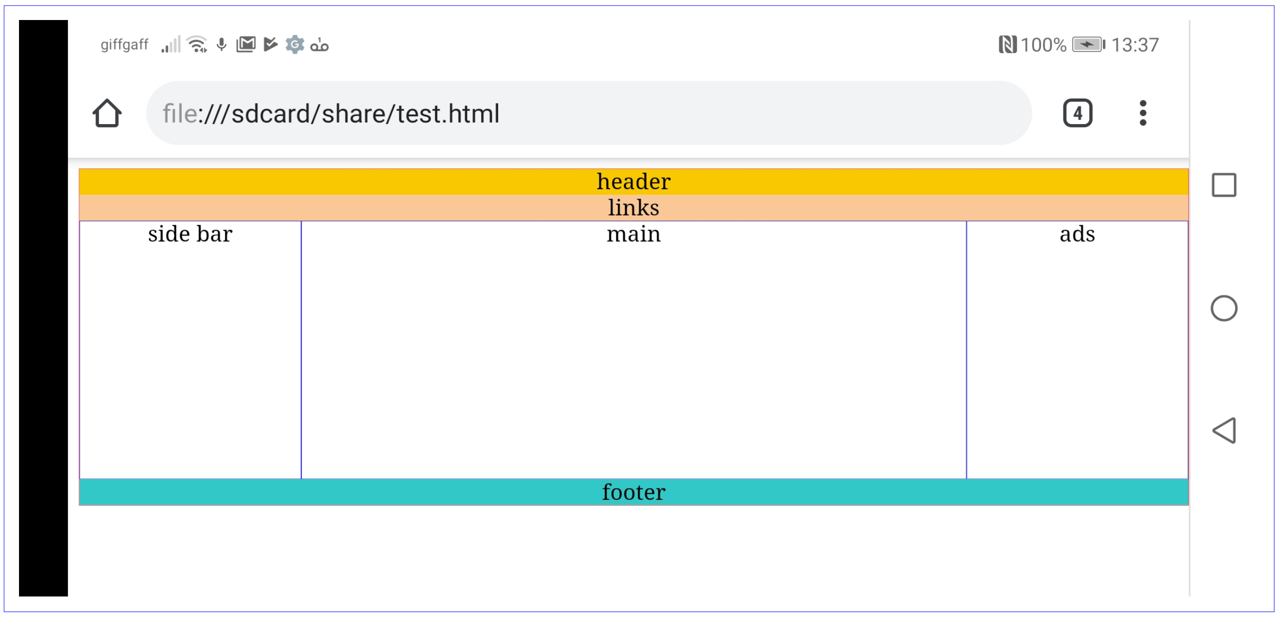
```
@media (min-width: 30em) { /* if screen at least 30 characters wide */
```

```
.wrapper {
  ..as before
}
@media (max-width: 30em) { /* if less than 30 characters wide */

.wrapper {
  display: block; /* normal, one below the other */
  text-align: center;
  border: thin red solid;
  width: 100%; /* full width */
  margin: auto;
}
div
{
border:thin blue solid;
}

header {
  background: rgb(250,200,0);
}

nav
{

   background:
rgb(250,200,150);
}
.side
{

}
.main
{

}
.ads
{

}
footer
{
  background:
rgb(50,200,200);
}
} /* end phone styles */
```



header
links
side bar
main
ads
footer

but landscape on the phone looks like the desktop:



There are a whole set of @media queries - see references for details.