1

Cascading Style Sheets

Styling the Web

Have no fear of perfection; you'll never reach it.

- Anonymous

Contents

How To Use This Document

Bits of text in red are links and should be clicked at every opportunity. Bits of text in `monospaced green` represent code. Most the other text is just text: you should probably read it.

Some sections are marked "Read-Only": these are sections that are intended to be read through in your own time and will not have a corresponding lecture.

Copying and pasting code from a PDF can mess up indentation. For this reason large blocks of code will usually have a View Code link above them. If you want to copy and paste the code you should follow the link and copy the file from GitHub.

Taking Notes

In earlier cohorts I experimented with giving out notes in an editable format. But I found that people would often unintentionally change the notes, which meant that the notes were then wrong. I've switched to using PDFs as they allow for the nicest formatting and are also immune from accidental changes.

Make sure you open the PDF in a PDF viewing app. If you open it in an app that converts it into some other format (e.g. Google Docs) you may well miss out on important formatting, which will make the notes harder to follow.

I make an effort to include all the necessary information in the notes, so you shouldn't need to take any additional notes. However, I know that this doesn't work for everyone. There are various tools that you can use to annotate PDFs:

- Preview (Mac)
- Edge (Windows)
- Hypothes.is
- Google Drive (*not* Google Docs)
- Dropbox

Do not use a word processor to take programming notes! (e.g. Google Docs, Word, Pages). Word processors have the nasty habit of converting double-quotes into "smart-quotes". These can be almost impossible to spot in a text-editor, but will completely break your code.

Reference

These notes act as a reference to what you have been taught. This is not a text book, to list all properties & values would be elaborate, thus therefore these are just notes and not definitive. Use the links to read more about all the properties.

Chapter 1

CSS Syntax

1.1 Introduction

1.1.1 Intro & Syntax

CSS stands for Cascading Style Sheets. It's a specific language created to style HTML. You style HTML elements by specifying the element and adding property/value pairs to define how it will look and where it will be placed.

```
body {
    background-color: red;
}

/* called */
selector {
    property: value;
}
```

Syntax

Every character is important. The curly braces, the colons and semicolons. Computers can't read CSS properly without these things in the correct place.

1.1.2 Selectors

There are various ways to *select* the html element you want to write styles for in CSS. Below are the most common, however check out the Complex Selectors section for

more.

Element

You can write the element itself

```
p {
    font-size: 1.2rem;
}
```

Class

There is a `class` attribute in HTML. We can add this attribute to elements and use it as a selector in out CSS. We have to remember to add a fullstop (period) at the beginning.

```
.myclass {
    width: 50%;
}
```

id

There is also an `id` attribute in HTML and we can use that in much the same way as a class. However we add a hash at the beginning instead.

```
#myid {
    padding: 20px;
}
```

Two at the same time

Separate the selectors with a comma

```
    .header-main, section {
    background-color: lightgreen;
}
```

Child

Use a space

```
    article p {
    border-bottom: 2px solid grey;
}
```

1.1.3 Where do we write CSS?

Where we include CSS is important.

inline

We can add it as an attribute to an element

```
<p style="font-size:1.2rem;">Some text here</p>
```

style element

We can put it inside a `style` element within our HTML

```
<style>
    .myclass {
        width: 50%;
    }
</style>
```

CSS file

We can create a CSS file and write CSS in that. We link to it in the `head` of our html document.

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

Where is best?

All three places have their advantages and drawbacks. The most common is to create a separate CSS file. This keeps our CSS together and means we can affect more than one HTML page at a time, by including the same CSS file in them.

1.1.4 Whitespace

Whitespace matters when you write values in CSS, the part after the colon. But it doesn't affect anything else. However you should structure your CSS like the examples as this is industry standard.

1.1.5 Comments

We write a CSS comment like so:

```
/* This is a CSS comment */
```

1.2 Complex Selectors

Child & Sibling

```
/* direc t child */
parentEl > childEl  {}

/* sibling */
a ~ img  {}

/* ajacent sibling */
a + img  {}
```

Attribute

```
/* given attr is present */
section[class] {}

/* attribute */
```

```
input[type="submit"]  {}

/* any part of attr */
a[href*="http"]  {}

/* begins with */
a[href^="https"]  {}

/* ends with */
a[href$="pdf"]  {}

/* space separated */
a[class~="btn"] {}

/* hyphen separated */
[class|="page"] {}
```

Pseudo classes

```
a:link
a:visited
a:hover
a:active
a:focus

input:enabled
input:disabled
input:checked
input:indeterminate {}
```

Indeterminate

Targeting children

```
el:first-child {}
el:last-child {}
el:only-child {}

el:first-of-type {}
el:last-of-type {}
el:only-of-type {}
```

```
el:nth-child(1) {}
el:nth-last-child(2n) {}

el:nth-of-type(2n) {}
el:nth-last-of-type(3) {}

/* Has id which is linked to via <a> with # */
section:target {}

/* no children */
div:empty {}

/* does not have */
article:not(.about) {}
```

Pseudo elements

```
div::before
section::after

p::first-letter
p::first-line
p::selection
```

1.3 CSS Specificity

Specificity determines, which CSS rule is applied by the browsers.

It's usually the reason why your CSS-rules don't apply to some elements, although you think they should.

Rules

Every selector has its place in the specificity hierarchy.
1. Inline styles 2. IDs 3. Classes, attributes & pseudo classes 4. Elements & pseudo elements
If two selectors apply to the same element, the one with higher specificity wins.

Order is important - latest rule still applies...Unless specificity value is higher

```
<div id="thing">Content Here</div>
```

```
#thing {background-color: red;}
div {background-color: blue;}
```

The background colour of the div will be red.

Chapter 2

CSS Styles

2.1 Backgrounds & Borders

2.1.1 Backgrounds

- `background-color` | Takes a colour value
- `background-image` | `url()` function or gradient
- `background-repeat` | repeat, repeat-x, repeat-y or no-repeat
- `background-position` | x & y units or keywords (top, bottom, left, right)
- `background-size` | x & y units or keywords
- `background` | Shorthand: colour, image, repeat, position

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

2.1.2 Borders

- `border-width` | Unit
- `border-style` | Keyword (like solid or dashed)
- `border-color` | Colour
- `border` | Shorthand: unit, style, colour
- `border-right` | All of the above but for each side, top, right, bottom, left

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

2.1.3 Box Shadow

- `box-shadow` | horizontal, vertical, blur, spread, colour

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

2.1.4 Border radius (rounded corners)

- `box-radius` | unit
- `box-top-left-radius` | for each corner - there are lots of these, check the link below

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

2.2 Colours

Anywhere you use a colour

```
article {
    border: 1px solid #fafafa;
    background-color: hsla(146, 56%, 48%, 0.8);
}
```

Colour names

Red, blue, yellow, aliceblue, firebrick
Transparent
List of colour names

Hex codes

3 bit hexidecimal format
Each single or pair represent R G or B channel
`#FF0000` `#F00` `#EEF4F1`

rgb()

Each value 0-255 represents red, green or blue channel
`rgb(127, 63, 25)` `rgb(0, 255, 0)`

RGBA

Same as rgb but with alpha (opacity/transparency) channel 0-1
`rgba(255, 255, 255, 1)` `rgba(0, 255, 0, 0.5)`

HSL

Like rgb, but each value represents *hue*, *saturation* and *lightness* rather than a colour channel
Hue on the colour wheel 0-360, saturation & lightness percentage
`hsl(100, 50%, 50%)`

HSLA

Same as before but with alpha channel
`hsla(200, 20%, 20%, 1)` `hsla(0, 80%, 90%, 0.5)`

2.3 Web Fonts

When a website loads it needs to show the same fonts as the design.

```
body {
    font-family: 'Helvetica', "Arial", sans-serif;
}
```

We all have different computers with different fonts on, these are called *system fonts* and although there are common fonts, it's highly unlikely all users will have the custom font from the design.
We need to make sure the font used in the design loads with the website. We need to serve the font files.

Finding font files

The website designer should provide you with any font files they used in the design. You can google if you know the font name, or there is this website (also an app) called What The Font, which allows you to upload an image of a font and find it for yourself.

2.3.1 Loading Fonts

YOURSELF

```
@font-face {
    font-family: 'Gotham';
    font-weight: normal;
    src: url("fonts/gotham-book.otf") format("opentype");
}
```

Remember if the font has different styles to include them under the same name with the style specified (ie `font-weight: bold;`).

WEBFONT SERVICE

```
<link href="https://fonts.googleapis.com/css?family=Gotham"
↪   rel="stylesheet">
```

2.3.2 Font Styling

If a property starts with `font` by and large it affects children as well.
Common:

```
body {
    font-family: 'Gotham', "Arial", sans-serif;
    font-size: 1em;
    color: #666;
    text-align: right;

    line-height: 1.6;
    letter-spacing: 1px;
    text-decoration: underline;
    text-transform: uppercase;
    text-shadow: 1px 1px 1px black;
}
```

Worth noting:

```
body {
    text-indent: 2em;
    text-overflow: fade(10px);

    word-spacing: 5px;

    word-break: break-all;
    overflow-wrap: break-word;
    white-space: nowrap;
}
```

2.3.3 Further Reading

CSS Tricks articles on variable fonts
Loading fonts performantly
System fonts

2.4 CSS Effects

- `linear-gradient()` function used to create background images
- `radial-gradient()` function used to create background images
- `conic-gradient()` function used to create background images
- `filter` Create media effects, like invert, saturate.
- `mix-blend-mode` Changes blend mode of element
- `background-blend-mode` Changes blend mode of background image
- `shape-outside` Create a path to use as the shape outline
- `mask-image` Takes an image or gradient and masks over the element

This is just a reference list. Check MDN for different values.

Further Reading

https://leaverou.github.io/css3patterns/
https://css-tricks.com/basics-css-blend-modes/
https://www.sarasoueidan.com/blog/css-shapes/

Chapter 3

CSS Layout

3.1 CSS Box Model

Every element on a webpage is basically a rectangle.

This is it's box - when we talk about the box model, we're talking about this box. Pretty much all the layout properties affect this box, the size of it and where it is positioned.
Let's start with the basics.

Every HTML element has a default set of styles

You can check them in devtools

3.1.1 Every element has a display property

One such style is the `display` property
Common default `display` values are `block` & `inline`
Blocks are boxes which take up all the width of the parent. Inline elements sit inline next to each other.
Check out media, lists and tables for some others.
The display property changes how the element behaves

Common display values

- `block` box
- `inline` sits inline
- `inline-block` box that sits inlne (padding and margin work as expected)

3.1.2 Layout properties

- `padding` Unit, can use shorthand or add side specifically (eg padding-top)
- `margin` Unit, can use shorthand or add side specifically (eg margin-bottom)
- `width` Unit or auto keyword
- `height` Unit or auto keyword
- both `min-` & `max-` on width and height. Can't go smaller or bigger than
- `box-sizing` default is content-box which means dimensions affect content. Padding & border is added. border-box include padding and border into dimensions.

No heights needed

When working with content it's always a good idea not to specify height on elements. Content gets changed, browsers get smaller. You want to allow the content to flow down the page.

3.2 CSS Floats

Floating elements is supposed to be used to move something left or right and have other content move around it. It's used this way in CSS Shapes. However it has been used in the past for layout, before we had flexbox or grid.

```
/* you can float left, right or none */
header h1, header nav {
    float: left;
}
```

Floating elements make them jump out of the block scope.
Their containing elements can't see them anymore.
We can get around this by giving the containing (parent) element a **Block Formatting Context**
This happens when
- is floated - is positioned absolute - is displayed inline-block - has an overflow property - has a value other than visible

```
header {
    overflow: auto;
    /* OR */
    display: flow-root;
}

header h1, header nav {
    float: left;
}
```

3.3 CSS Units

Anything requiring a size

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

Used as a value. For example heights, widths, padding, margins, media, borders, background size...

Pixels

Absolute/fixed

```
article {
    margin: 10px;
}
```

Note: screens work in pixels

Percentage

Responsive, taken from parent

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

Viewport

```
article {
    width: 50vw;
}
```

- 1vw = 1% of viewport width
- 1vh = 1% of viewport height
- 1vmin = 1vw or 1vh, whichever is smaller
- 1vmax = 1vw or 1vh, whichever is larger

em

Font measurement - relative to parent

```
body {font-size: 16px;} /* default */
p {font-size: 1.2em;} /* 16 x 1.2 */
p a {font-size: 1.2em;} /* 16 x 1.2 x 1.2 */
```

rem

Like em but always relative to 'root'

```
body {font-size: 1rem;}
p {font-size: 1.2rem;}
p a {font-size: 1.2rem;}
```

Others

- ex - height of 'x'
- ch - width of '0'
- lh - equal to the line-height
- pt - 1/72"
- mm - mm
- cm - cm

3.4 CSS Flexbox

Is used to distribute items along an axis .The browser is in control of the distribution

Container Properties

- `display: flex;`
- `flex-direction` row | row-reverse | column | column-reverse
- `justify-content` flex-start | flex-end | center | space-around | space-between | space-evenly
- `flex-wrap` if you want to wrap
- `align-items` on cross axis;
- `align-content` cross axis space in container

Child Properties

- `order` default 0, change to order
- `flex-grow` default 0, change to give item more room
- `flex-shrink` default 0, change to give item less room

Flexbox on CSS Tricks. This is an incredibly good guide. It is recommended over anything we can put in these notes. It is also the most hit page on the whole of the website, we all use it all the time!

3.5 CSS Grid

Designers have been using grids for a very long time in print - magazine and newspapers.
This transferred over to web design. Specifically when we were working with fix width sites, we didn't have flexbox or grid back then.
So we created our own grid systems: CSS you added to your website which helped put elements in the right place. They were usually 12 columns and you added classes to your HTML (a bit like bootstrap today, but just for layout).

CSS Grid is here

Bonuses include: - Not tied into a system - No addons needed - No added classes
We use it for laying out a page, it's easy to put content in the right place. Think both horizontal and vertical axis, rather than flexbox's one. It's DOM dependent.

3.5.1 How to use grid

```
.page__home {
    display: grid;
}
```

Specify the amount of cols & rows

```
.page__home {
    display: grid;
    grid-template-rows: repeat(3, auto);
    grid-template-columns: 20vw 1fr 1fr 10vw;
}
```

How do we get things in places?

We can name the areas we have created

```
.page__home {
    display: grid;
    grid-template-rows: 20vh 1fr 1fr 16vh;
    grid-template-columns: repeat(3, 1fr);
    grid-template-areas:
        "top top top"
        "middle middle side"
        "middle middle ."
        ". bottom ."
    ;
}
```

Then we can reference those names when we're styling the child sections

```
.header-main {
    grid-area: top;
}
```

Or

Grids have number lines automatically, we can use those

```
.header-main {
    grid-row-start: 1;
    grid-row-end: 2;
    grid-column-start: 1;
    grid-column-end: 4;
}
```

Shorthand

```
.header-main {
    grid-row: 1 / 2;
    grid-column: 1 / 4;
}
```

```
.header-main {
    grid-area: 1 / 1 / 2 / 4;
}
```

Explicit and implicit grids

Explicit is what you know, implicit is what you don't know. When we specify rows and columns and names as above we create an *explicit* grid. If we didn't CSS grid would place items automatically for us and be an *implicit* grid

```
section {
    display: grid;
    /* how big the rows are */
    grid-auto-rows: 140px;
    /* Direction */
    grid-auto-flow: column;
}
```

The 'fr' unit

The 'fr' unit is only available for grid (We are considering it as a global unit ø/)

Grid Gap

Adds gaps in-between cells

```
section {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-column-gap: 1rem;
    grid-row-gap: 1rem;
    /* or */
    grid-gap: 1rem;
}
```

'minmax()'

Like min/max width/height. Specifies how big or small something can be

```
section {
    display: grid;
    grid-template-row: minmax(100px, auto);
}
```

You can name grid lines

```
section {
    display: grid;
    grid-template-columns: [first] 40px [line2] 50px [line3] auto
    ↪  [col4-start] 50px [five] 40px [end];
}
```

3.5.2 Children

Alignment

```
/* on parent */
section {
    justify-items: start;
    align-items: stretch;
}

/* on individual child */
section p {
    justify-self: end;
    align-self: center;
}
```

Span keyword

```
section p {
    grid-row-start: 1;
    grid-row-end: span(2);
}
```

3.5.3 Further help

Guide on CSS Tricks

> **Firefox**
>
> Firefox is the best browser for debugging your grid. It has a great deal of tools in it's devtools which are extremely helpful for viewing the grid you have created and it's information.

3.6 CSS Positions & Transforms

Used for moving elements and altering elements, not for layout. Only alters element and contents, not surroundings.

3.6.1 Position

```
/* default for all elements */
position: static;
/* gives an element position, can be moved */
position: relative;
/* position an element out of flow */
position: absolute;
/* stays put when scrolling */
position: fixed;
```

Use with position properties. You don't need all four. Movement depends on type of position and parent elements.

```
top: unit;
bottom: unit;
left: unit;
right: unit;
```

3.6.2 Transforms

These modify element

```
transform: translate(10px,10px);
transform: translateX(10px);
transform: scale(x,y);
transform: rotate(30deg) skewX(25deg);
```

Further reading:

David DeSandro's transforms examples
CSS Transforms on CSS Tricks

Chapter 4

Further

4.1 CSS Media Queries

There are lots and lots of different ways you can consume websites and webapps these days. Monitors, phones, TVs, game consoles and watches to name a few.
We have only been looking at our web pages on a screen so far. We need to make sure they look good at all sizes.
To do this we will use responsive design techniques. This is different from adaptive design which means serving a whole different site or app for different devices.

All environments

Responsive design isn't just about making sure things look good at different screen size. It should also take into account the environment the user is in when using your site or app, like how noisy things are or what data connection they have.

4.1.1 How?

- Use relative not absolute CSS units - CSS Media Queries | - Is also JavaScript feature and device detection |

Media queries

Different types of media, like print

```
@media print {
```

```
    * {
        background-color: transparent;
    }

}
```

Breakpoints (viewport width)

```
/* phone */
@media screen and (max-width: 500px) {

    #left-column {
        display: none;
    }

}
```

```
/* Extra small devices (phones, less than 768px) */
@media screen and (max-width: 767px) { ... }

/* Small devices (tablets, 768px and up) */
@media screen and (min-width: 768px) and (max-width: 991px) { ... }

/* Medium devices (desktops, 992px and up) */
@media screen and (min-width: 992px) and (max-width: 1199px) { ... }

/* Large devices (large desktops, 1200px and up) */
@media screen and (min-width: 1200px) { ... }
```

Testing

You can move the screen width when devtools is open. There is also a mobile/tablet icon at the top, if you select this you can choose a number of screen sizes.

4.1.2 Loading different pictures

You can use the `picture` element to load different image types and also different files based on screen size

```
<picture>
    <source srcset="surfer.png" media="(min-width: 800px)">
    <img src="surfer.jpg" />
</picture>
```

Object fit

You can use the 'object-fit' property to display how the image appears in it's container.

```
img {
    height: 100%; width: 100%;
    display: block;
    object-fit: contain;
}
```

Further reading

4.2 CSS Imports

Wouldn't it be great if you could separate out your CSS into smaller files?
But annoying to include them all in your HTML?
This is what imports are for.
We import all smaller CSS files into one file and use **that** one file in our HTML.
CSS knows to go and get all the CSS in the other files.

@import example

```
@import 'reset.css';
@import 'settings.css';
@import 'typography.css';
@import 'buttons.css';
```

Create all these files and add into your main css file

CSS files

How you break up and organise your CSS files is up to you & very much a learning process. Think about your components, having a CSS file for each is a good place to start.

4.3 CSS Variables

AKA Custom Properties

Another day, another CSS declaration

```
p {
  color: #00e6b8;
}
h1, h2, h3 {
  color: #00e6b8;
}
button {
  background-color: #00e6b8;
}
input {
  border-color: #00e6b8;
}
```

Wouldn't it be nice if we didn't have to look up the colour every time we use it? What if the designer changes the colour?

```
:root {
  --devme-teal: #00e6b8;
}

p {
  color: var(--devme-teal);
}
h1, h2, h3 {
  color: var(--devme-teal);
}
button {
  background-color: var(--devme-teal);
```

```
}
input {
  border-color: var(--devme-teal);
}
```

Can be any CSS value!

```
:root {
  --brand-font: Helvetica, sans-serif;
  --brand-colour: #00e6b8;
  --content-padding:    1rem;
}
```

Naming things

Naming things can be tricky. You can call a variable whatever you want, but what makes sense? Something that you and other developers will understand is a good place to start. Try to be generic, actually using the colour name as a variable isn't a good idea as the colours might change. Something like brand-colour works well.

4.4 CSS Frameworks

A CSS Framework exists to answer all the CSS you might want to write for a website, meaning you don't have to.

There are lots:

- Bootstrap
- Foundation
- Skeleton
- Bulma
- UIKit
- Semantic
- Pure
- Milligram

How

Include the CSS (and any other) code. Use the given HTML

Pros

- Quicker (once you've learnt)
- Work done for you
- Kept up to date

Cons

- Tied into a system, 'breaking out' is more work
- Bespoke styling is hard
- No control over support
- So. Much. Code!

Chapter 5

Theory

5.1 CSS Design Systems

5.2 A Section

A design system is a collection of reusable components, guided by clear standards, that can be assembled together to build any number of applications.
cite: https://www.invisionapp.com/inside-design/guide-to-design-systems/
Usually consists of Styleguides & Pattern Libraries

Style Guide

- Branding
- Tone of voice
- Photography
- Style

Pattern Library

Collection of digital elements you'd use in your product, with example code
http://ux.mailchimp.com/patterns/dialogs

Why is this important

Because that's how we approach writing HTML & CSS for a product

Tools

Fractal is the most common
https://fractal.build/
You build your components and it acts a good place to keep them and a reference for the team

Resources

https://adactio.com/links/tags/patterns
http://styleguides.io/

5.3 CSS Atomic Design

Way of thinking about structure:

- **Atoms:** Elements (input, p, button etc...)
- **Molecules:** Set of elements (search form)
- **Organism:** Set of Molecules (site header)
- **Templates:** Set of Organisms (repeated)
- **Pages:** Final site

—

Further reading

- http://atomicdesign.bradfrost.com/

5.4 OOCSS

Separation of structure and skins, style from layout. Encourages reuseable, scaleable CSS.
Example a `.button` class used on a website, where the look of the button is held within the class and how big it is and where it is is held elsewhere.

```
.button {
background-color: green;
border: 1px solid blue;
}
```

```
form .button {
    width: 100px;
    margin: 0px auto;
}

.card .button {
    display: block;
    width: 200px;
}
```

This helps you decide where styles go.
Further reading on the OOCSS site

5.5 BEM: Block Element Modifier

Block Element Modifier

BEM is a class naming convention. It may seem long winded, but we have already seen the benefits of the button class, you can use it on any element and it will adopt the styles.
It also helps other developers understand what CSS is doing what.
A component is a block and inside that you have elements. These are separated by two underscores. If there are modifications to the default layout or style, you add another term to any of these with a double hyphen.

Example

```
<article class="card">

    <header class="card__head">
        <h2 class="card__head__text">Card heading</h2>
    </header>

    <figure class="card__img">
        <img src="" alt="" />
    </figure>

    <p class="card__blurb">Some text here</p>

    <a href="" class="card__link">A link</a>

</article>
```

A card with no image could be described as ‘.card-noimage‘

Glossary

- **Selector**: The part of the CSS block which represents which HTML element(s) to affect
- **Property**: The style you want to change. Written before the colon.
- **Value**: What you want to set the *Property* to, written after the colon.
- **Webfont**: Term to describe the use of custom fonts within websites
- **CDN**: Content Delivery Network. Geographically distributed group of servers which work together to provide fast delivery of assets

Colophon

Created using TEX

Fonts

- Feijoa by Klim Type Foundry
- Avenir Next by Adrian Frutiger, Akira Kobayashi & Akaki Razmadze
- Fira Mono by Carrois Apostrophe

Written by Ruth John

∞

December 1, 2020