Front-End Web Development

Unit 3: Cascading Style Sheets (CSS)

UCD DUBLIN PROFESSIONAL ACADEMY

Course Outline

1. Getting Started
2. HTML - Structuring the Web
3. **CSS - Styling the Web**
4. JavaScript - Dynamic client-side scripting
5. CSS - Making Layouts
6. Introduction to Websites/Web Applications
7. CSS - Advanced
8. JavaScript - Modifying the Document Object Model (DOM)
9. Dynamic HTML
10. Web Forms - Working with user data
11. JavaScript - Advanced
12. Building a Web Application with JavaScript
13. Introduction to CSS Frameworks – Bootstrap
14. Building a Web Application with Svelte
15. SEO, Web security, Performance
16. Walkthrough project

Course Learning Outcomes

- Competently write HTML and CSS code
- Create web page layouts according to requirements using styles
- Add interactivity to a web page with JavaScript
- Access and display third-party data on the web page
- Leverage Bootstrap and Static Site Generator

Assessment

- Final Project - 100% of the grade

  - Design and Build functioning Website using HTML5, CSS (including Bootstrap), JavaScript (browser only)

✔ Code will be managed in GitHub
✔ Website will be deployed to GitHub Pages
✔ All code to follow best practice and be documented

- Details and How-To-Guide are available on the course page under the section called Assessments

3. Cascading Style Sheets

In This Unit

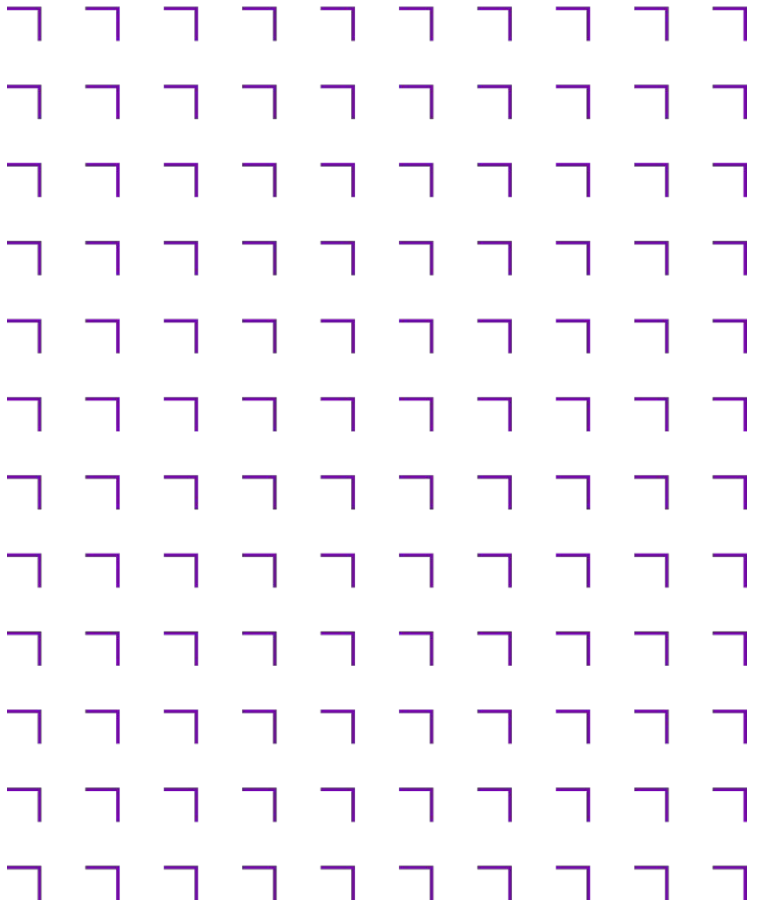
Title

Overview and Principles of CSS

Typography & Media styles

Box Model

Overview and Principles of CSS



What is CSS?

- Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML or XML (including XML dialects such as SVG, MathML or XHTML)
- CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript
- CSS is designed to enable the separation of content and presentation, including layout, colors, and fonts. This separation can improve content accessibility; provide more flexibility and control in the specification of presentation characteristics
- Enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, which reduces complexity and repetition in the structural content
- Enable the .css file to be cached to improve the page load speed between the pages that share the file and its formatting

What is CSS?

- Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices
- CSS also has rules for alternate formatting if the content is accessed on a mobile device
- The name cascading comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element. This cascading priority scheme is predictable
- The CSS specifications are maintained by the World Wide Web Consortium (W3C)
- Internet media type (MIME type) text/css is registered for use with CSS by RFC 2318 (March 1998)
- In addition to HTML, other markup languages support the use of CSS including XHTML, plain XML, SVG, and XUL. CSS is also used in the GTK widget toolkit

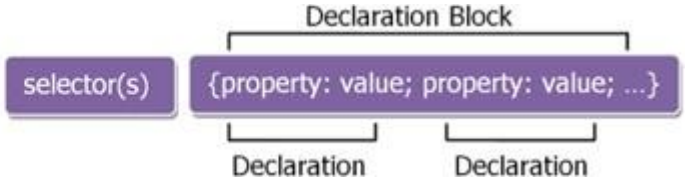
Brief History

- CSS was first proposed by Håkon Wium Lie on 10 October 1994. At the time, Lie was working with Tim Berners-Lee at CERN
- Style sheets have existed in one form or another since the beginnings of Standard Generalized Markup Language (SGML) in the 1980s
- CSS was developed to provide style sheets for the web
- History of CSS until 2016

CSS Syntax

- CSS is a rule-based language
- You define the rules by specifying groups of styles that should be applied to particular elements or groups of elements on your web page

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



- In the above example, the CSS rule opens with a selector. This selects the HTML element/tag that we are going to style, i.e. h1 tag
- We then have a set of curly braces { }
- Inside the braces will be one or more declarations, which take the form of property and value pairs. This example contains two declarations, one for text color and the other for font-size

Add CSS to a HTML Page

3 different ways to apply CSS to an HTML document that you'll commonly come across:

- External stylesheet: Add an external stylesheet to the head section of html page

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

- Internal stylesheet: Embed the styles directly in the head section

```
  <style>
    body {font-family: arial, helvetica; font-size: 90%;}
    h1 {font-size: 250%;}
  </style>
```

- Inline styles: Use the style attribute to apply styles to a single element in the html

```
  <h1 style="font-size: 250%; color: red;>Town Hall</h1>
```

Best Practice: use external stylesheets

Selectors

A selector targets the specific HTML that you want to apply the styles to

Types of Selectors

- Type Selectors: This group includes selectors that target an HTML element such as an `<h1>`

```
<h1 style="font-size: 250%; color: red;>Town Hall</h1>
```

- ID Selectors: Apply the style rules to the element with the specific ID attribute

```
#heading1 {
  color: #000000;
}
```

Types of Selectors

- Class Selectors: You can define style rules based on the class attribute of the elements. All the elements having that class will be formatted according to the defined rule. This rule renders the content in black for every element with class attribute set to black in our document

```
.black {
  color: #000000;
}
```

CSS Property: color

- The color property sets the foreground color value of an element's text and text decorations, and sets the currentcolor value
- Colours are typically represented as six digit hexadecimal numbers preceded by # character.
  - e.g #FFFFFF, #09AB4E
- These are actually three 2 digit hexadecimal numbers representing the amount of Red, Green and Blue in the colour, ie. 09, AB, 4E
- There are also a limited set of well known colour names https://www.w3schools.com/colors/colors_names.asp
  - e.g. red, green, blue, aliceblue, aqua, cyan, gray
- Always ensure a high contrast ratio between the text color and the background colour

CSS Comments

- CSS comments are not displayed in the browser, but they can help document your source code
- A CSS comment is placed inside the <style> element or css file, and starts with /* and ends with */

```
/* This is a single-line comment */
p {
  color: red;
}

/* This is
a multi-line
comment */
p {
  color: red;
}
```

Activity

- Join a breakout room
- Download the CSS example code (unit-03/exercises) from Moodle
- Start work on the CSS-basics exercise
- The README outlines the steps
- Any questions let me know

You have 35 minutes

- Lecturer will visit each room in turn, etc...
- Will start next topic on the hour

Typography & Media Styles

Selectors Continued

- Universal Selector: Rather than selecting elements of a specific type, simple match the name of any element type

```
* { color: #000000; }
```

Attribute Selectors

- Different ways to select elements based on the presence of a certain attribute on an element

Selector	Example	Description
[attr]	a[title]	Matches elements with an attr attribute (whose name is the value in square brackets)
[attr=value]	a[href="https://ucdpa.ie"]	Matches elements with an attr attribute whose value is exactly value - the string inside the quotes
[attr~=value]	p[class~="special"]	Matches elements with an attr attribute whose value is exactly value, or contains value in its (space separated) list of values
[attr|=value]	div[lang|="zh"]	Matches elements with an attr attribute whose value is exactly value or begins with value immediately followed by a hyphen.

Pseudo-classes

A pseudo-class is a keyword which starts with a colon and is used to define a special state of an element

For example, it can be used to:

- Style an element when a user mouses over it
- Style visited and unvisited links differently
- Style an element when it gets focus

```
a:link { color: #FF0000; } /* unvisited link */
a:visited { color: #00FF00; } /* visited link */
a:hover { color: #FF00FF; }
a:active { color: #0000FF; }

article p:first-child { font-size: 120%; font-weight: bold; } /* make the first paragraph in an article larger and bold */

:last-child
:only-child
:invalid /* styles differently any <form>, <fieldset>, <input> or other <form> element whose contents fail to validate */
```

Pseudo-elements

Pseudo-elements behave in a similar way. However, they act as if you had added a whole new HTML element into the markup, rather than applying a class to existing elements.

Pseudo-elements start with a double colon ::. ::before is an example of a pseudo-element.

Other examples:

```
article p:first-child::first-line { font-size: 120%; font-weight: bold;} /* style the first line of the first paragraph */

/* Generating content with ::before and ::after */
.box::before { content: "This should show before the other content. "; }
```

Combinators

A combinator is something that explains the relationship between the selectors. A CSS selector can contain more than one simple selector. Between the simple selectors, we can include a combinator

There are four different combinators in CSS:

- descendant selector(space)
- child selector (>)
- adjacent sibling selector (+)
- general sibling selector (~)

```
.box p { color: red; } /* style all paragraphs which have an ancestor which has a class called box */
.box > p {color: red; } /* style all paragraphs which have a parent which has a class called box */
p + img {} /* select all <img> elements that are immediately preceded by a <p> element */
p ~ img {} /* select all <img> elements that come anywhere after <p> elements */
```

CSS values and units

All CSS properties take a value which has a type. One of the more common types is numeric

Data Type	Description
integer	An <integer> is a whole number such as 1024 or -55.
number	A <number> represents a decimal number — it may or may not have a decimal point with a fractional component. For example, 0.255, 128, or -1.2.
dimension	A <dimension> is a <number> with a unit attached to it. For example, 45deg, 5s, or 10px. <dimension> is an umbrella category that includes the <length>, <angle>, <time>, and <resolution> types
percentage	A <percentage> represents a fraction of some other value. For example, 50%. Percentage values are always relative to another quantity. For example, an element's length is relative to its parent element's length

Absolute length units

Unit	Name	Equivalent to
cm	Centimeters	1cm = 37.8px = 25.2/64in
mm	Millimeters	1mm = 1/10th of 1cm
Q	Quarter-millimeters	1Q = 1/40th of 1cm
in	Inches	1in = 2.54cm = 96px
pc	Picas	1pc = 1/6th of 1in
pt	Points	1pt = 1/72nd of 1in
px	Pixels	1px = 1/96th of 1in

Most of these units are more useful when used for print, rather than screen output. For example, we don't typically use cm (centimeters) on screen. The only value that you will commonly use is px (pixels)

Relative length units

Relative length units are relative to something else, perhaps the size of the parent element's font, or the size of the viewport. Very important for responsive design and accessibility. Most common ones:

Unit	Relative to
em	Font size of the parent, in the case of typographical properties like font-size, and font size of the element itself, in the case of other properties like width
rem	Font size of the root element, i.e. html element
vw, vh	1% of the viewport's width, height
vmin, vmax	1% of the viewport's smaller, larger dimension

Percentages

- In a lot of cases, a percentage is treated in the same way as a length
- The thing with percentages is that they are always set relative to some other value
- For example, if you set an element's font-size as a percentage, it will be a percentage of the font-size of the element's parent
- If you use a percentage for a width value, it will be a percentage of the width of the parent

Cascade, specificity, and inheritance

CSS stands for Cascading Style Sheets, and that first word cascading is incredibly important to understand — the way that the cascade behaves is key to understanding CSS

Stylesheets cascade — at a very simple level, this means that the origin, the cascade layer, and the order of CSS rules matter. When two rules from the same cascade layer apply and both have equal specificity, the one that is defined last in the stylesheet is the one that will be used

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

Specificity

Specificity is the algorithm that the browser uses to decide which property value is applied to an element and so the effect of the cascade

If there are two or more CSS rules that point to the same element, the selector with the highest specificity value will "win", and its style declaration will be applied to that HTML element

Think of specificity as a score/rank that determines which style declaration is ultimately applied to an element. For example:

```
<html>
  <head>
    <style>
      .test {color:
      green;}   p {color:
      red;}
    </style>
  </head>
  <body>
    <p class="test">Hello World!</p>
  </body>
</html>
```

Specificity Hierarchy

Every CSS selector has its place in the specificity hierarchy.

There are four categories which define the specificity level of a selector (highest to lowest):

1. Inline styles - Example: <h1 style="color: pink;">
2. IDs - Example: #navbar
3. Classes, pseudo-classes, attribute selectors - Example: .test, :hover, [href]
4. Elements and pseudo-elements - Example: h1, ::before

There is one exception to this rule: if you use the !important rule, it will even override inline styles!

Inheritance

- Inheritance also needs to be understood in this context — some CSS property values set on parent elements are inherited by their child elements, and some aren't
- For example, if you set a color and font-family on an element, every element inside it will also be styled with that color and font, unless you've applied different color and font values directly to them
- Some properties do not inherit — for example, if you set a width of 50% on an element, all of its descendants do not get a width of 50% of their parent's width.
- If this was the case, CSS would be very frustrating to use!

Import

- The @import rule allows you to import a style sheet into another style sheet
- The @import rule must be at the top of the document (but after any @charset declaration)
- The @import rule also supports media queries, so you can allow the import to be media-dependent

```
@import "navigation.css"; /* Using a string */

@import url("navigation.css"); /* Using a url */

@import "printstyle.css" print; /* Import the "printstyle.css" style sheet ONLY if the media is print */

@import "mobstyle.css" screen and (max-width: 768px); /*Import the "mobstyle.css" style sheet
ONLY if the media is screen and the viewport is maximum 768 pixels */
```

url(), calc()

- The url() CSS function is used to include a file. The parameter is an absolute URL, a relative URL, a blob URL, or a data URL

```
/* associated properties */
background-image: url("star.gif");
list-style-image: url('../images/bullet.jpg');
content: url("pdficon.jpg");
```

- The calc() CSS function lets you perform calculations when specifying CSS property values

```
/* property: calc(expression) */
width: calc(100% - 80px);
```

Backgrounds

The CSS background properties are used to add background effects for elements

Property	Description
background-color	sets the background color of an element
background-image	sets an image to use as the background of an element
background-repeat	sets how the background-image should repeat
background-attachment	specifies whether the background image should scroll or be fixed (will not scroll with the rest of the page)
background-position	sets the starting position of a background image
background-origin	specifies where the background image(s) is/are positioned
background-size	specifies the size of the background image(s)

Backgrounds continued

The CSS background properties are used to add background effects for elements

Property	Description
background-clip	defines how far the background (color or image) should extend within an element
background	sets all the background properties in one declaration
opacity	sets the opacity/transparency of an element. It can take a value from 0.0 - 1.0. The lower value, the more transparent. All child elements inherit same value, this can make text hard to read. To avoid this use RGBA values instead

Multiple images are also allowed. The different background images are separated by commas, and the images are stacked on top of each other, where the first image is closest to the viewer

Custom Properties, i.e. var()

- The var() function is used to insert the value of a CSS variable
- CSS variables have access to the DOM, i.e. you can create variables with local or global scope, change the variables with JavaScript, and change the variables based on media queries
- Typical usage is setting theme colours

```
:root { /* :root selector matches the documents root element */
  --blue: #1e90ff;  /* <variable-name>:<variable-value> */
  --white: #ffffff;
}

body { background-color: var(--blue); }

h2 { border-bottom: 2px solid var(--blue); }
```

The var() function cannot be used as property names, selectors, or anything else besides property values, i.e. can't be used in a media query or container query

Text Colour & Alignment

CSS has a lot of properties for formatting text

Property	Description
color	The color property is used to set the color of the text. The color is specified by a color name (red), a HEX value (#ff0000), an RGB value (rgb(255,0,0))
background-color	sets the background color of the element containing the text
text-alignment	set the horizontal alignment of a text (left, right, center, justify)
text-alignment-last	specifies how to align the last line of a text
direction and unicode-bidi	can be used to change the text direction of an element (rtl and bidi-override)
vertical-align	sets the vertical alignment of an element (baseline, text-top, etc.)

Text Decoration & Transformation

Property	Description
text-decoration	sets all the text-decoration properties in one declaration
text-decoration-color	specifies the color of the text-decoration
text-decoration-line	specifies the kind of text decoration to be used (underline, overline, etc.)
text-decoration-style	specifies the style of the text decoration (solid, dotted, etc.)
text-decoration-thickness	specifies the thickness of the text decoration line
text-transform	controls the capitalization of text (uppercase, lowercase, capitalize)

Text Decoration & Transformation

Property	Description
letter-spacing	specifies the space between characters in a text
line-height	specifies the line height
text-indent	specifies the indentation of the first line in a text-block
white-space	specifies how to handle white-space inside an element
word-spacing	specifies the space between words in a text

Text Shadow

The text-shadow property adds shadow to text. In its simplest use, you only specify the horizontal shadow (2px) and the vertical shadow (2px):

```
h1 {
  text-shadow: 2px 2px;
}
```

Add a color (red) to the shadow and a blur effect (5px) to the shadow:

```
h1 {
  text-shadow: 2px 2px 5px red;
}
```

Fonts

- Choosing the right font is important and has a huge impact on how the readers experience a website
- The right font can create a strong identity for your brand
- It is also important to choose the correct color, is easy to read and text size for the font

airbnb	LL Brown	Walt Disney	Walt Disney
Medium	Noe Display Bold	vodafone	Vodafone
Spotify	Gotham	TOSHIBA	Eurostile Black
American Apparel	Helvetica Black	Rolls-Royce	Gill Sans Roman
WhatsApp	Neue Helvetica 75	Walmart	Bogle Font

Font Families

In CSS there are five generic font families:

- *Serif* fonts have a small stroke at the edges of each letter. They create a sense of formality and elegance
- *Sans-serif* fonts have clean lines (no small strokes attached). They create a modern and minimalistic look
- *Monospace* fonts - here all the letters have the same fixed width. They create a mechanical look
- *Cursive* fonts imitate human handwriting
- *Fantasy* fonts are decorative/playful fonts

All the different font names belong to one of the generic font families.

Note: On computer screens, sans-serif fonts are considered easier to read than serif fonts.

Font Properties

In CSS, we use the font-family property to specify the font of a text

Note: If the font name is more than one word, it must be in quotation marks, like: "Times New Roman"

Tip: The font-family property should hold several font names as a "fallback" system, to ensure maximum compatibility between browsers/operating systems. Start with the font you want, and end with a generic family (to let the browser pick a similar font in the generic family, if no other fonts are available). The font names should be separated with a comma

```
.p1 {
  font-family: "Times New Roman", Times, serif;
}

.p2 {
  font-family: Arial, Helvetica, sans-serif;
}

.p3 {
  font-family: "Lucida Console", "Courier New", monospace;
}
```

Web Safe Fonts

Web safe fonts are fonts that are universally installed across all browsers and devices

However, there are no 100% completely web safe fonts. There is always a chance that a font is not found or is not installed properly. Therefore, it is very important to always use fallback fonts.

This means that you should add a list of similar "backup fonts" in the font-family property. If the first font does not work, the browser will try the next one, and the next one, and so on. Always end the list with a generic font family name . The following list are the best web safe fonts for HTML and CSS:

- Sans-serif: e.g. Arial, Verdana, Tahoma, Trebuchet MS
- Serif: e.g. Times New Roman, Georgia, Garamond
- Monospace: e.g. Courier New
- Cursive: e.g. Brush Script MT

Font Libraries to browse: https://fonts.adobe.com/, https://fonts.google.com/

Font Properties

Property	Description
font-style	is mostly used to specify italic text. This property has three values: normal - The text is shown normally, italic - The text is shown in italics, oblique - The text is "leaning" (oblique is very similar to italic, but less supported)
font-weight	specifies the weight of a font. Numeric values (100-900) can be used if the font supports it
font-variant	specifies whether or not a text should be displayed in a small-caps font

```
p.normal {
  font-style: normal; font-weight: normal;
}
p.italic {
  font-style: italic; font-weight: bold;
}
p.oblique {
  font-style: oblique; font-weight: 400
}
```

Font Properties: font-size

The font-size property sets the size of the text and is very important in web design. However, you should not use font size adjustments to make paragraphs look like headings, or headings look like paragraphs
Always use the proper HTML tags, like `<h1>` - `<h6>` for headings and <p> for

paragraphs The font-size value can be an absolute, or relative size.

Absolute size:

- Sets the text to a specified size
- Does not allow a user to change the text size in all browsers (bad for accessibility reasons)
- Absolute size is useful when the physical size of the output is known

Relative size:

- Sets the size relative to surrounding elements
- Allows a user to change the text size in browsers (improved accessibility)
- Note: If you do not specify a font size, the default size for normal text, like paragraphs, is 16px (16px=1em)

Font Properties: font-size

It's possible to use only px or em's but a solution that works in all browsers, is to set a default font-size in percent for the `<body>` element:

```
body {
  font-size: 100%;
}

h1 {
  font-size: 2.5em;
}

h2 {
  font-size: 1.875em;
}

p {
  font-size: 0.875em;
}
```

Font Properties: font

To shorten the code, it is also possible to specify all the individual font properties in one `font` property:

- font-style
- font-variant
- font-weight
- font-size/line-height
- font-family

Note: The font-size and font-family values are required. If one of the other values is missing, their default value are used

```
a.   {
   font: 20px Arial, sans-serif;
 }

b.   {
   font: italic small-caps bold 12px/30px Georgia, serif;
 }
```

Third-party Fonts

Google Fonts

If you do not want to use any of the standard fonts in HTML, you can use Google Fonts or other third-party fonts

Google Fonts are free to use, and have more than 1000 fonts to choose from

How To Use Google Fonts

Just add a special style sheet link in the `<head>` section and then refer to the font in the CSS. Read more...

```
<head>
  <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Sofia">
  <style>
  body {
    font-family: "Sofia", sans-serif;
  }
  </style>
</head>
```

CSS Icons

Icons can easily be added to your HTML page, by using an icon library

The simplest way to add an icon, is with an icon library, such as Font Awesome or Google Icons

For Font Awesome simple add the name of the specified icon class to any inline HTML element (like `<i>` or `<span>`)

All the icons are typically scalable vectors that can be customized with CSS (size, color, shadow, etc.)

```
<head>
  <script src="https://kit.fontawesome.com/a076d05399.js" crossorigin="anonymous"></script>
</head>
<body>
  <i class="fas fa-cloud"></i><i class="fas fa-heart"></i><i class="fas fa-car"></i><i class="fas fa-file"></i>
</body>
</html>
```

Tables

The look of an HTML table can be greatly improved with CSS

As a starting point apply the following css:

```
table {
  border-collapse: collapse;
  width: 100%;
}
td, th {
  border: 1px solid #999;
  padding: 0.5rem;
  text-align: left;
}
```

Make sure that the table html uses the available semantic tags including `<thead>` , `<tbody>` , `<caption>`

A lot of the general CSS properties also apply to tables. Read more...

Lists

The CSS list properties allow you to:

- Set different list item markers for ordered lists
- Set different list item markers for unordered lists
- Set an image as the list item marker
- Add background colors to lists and list items

Property	Description
list-style	Sets all the properties for a list in one declaration
list-style-image	Specifies an image as the list-item marker
list-style-position	Specifies the position of the list-item markers (bullet points)
list-style-type	Specifies the type of list-item marker

Lists

```
ul.a {
  background: #ff9999; /* affects entire list */
  list-style-type: circle;
  list-style-image: url('sqpurple.gif');
  list-style-position: outside;
}

ol.a li {
  background: #ffe5e5; /* affects list items */
  color: darkred;
}

ul.b {   /* shorthand property */
  list-style: square inside url("sqpurple.gif");
}

ol.d {   /* remove default styles
  */  list-style-type: none;
  margin: 0;
  padding: 0;
}
```

CSS for print

There may be times in which your website or application would like to improve the user's experience when printing content. There are a number of possible scenarios:

- You wish to adjust layout to take advantage of the size and shape of the paper
- You wish to use different styles to enhance the appearance of your content on paper
- You wish to use higher resolution images for a better result
- You want to adjust the user experience of printing, such as presenting a specially-formatted version of your content before printing begins

Some common things you can do include:

- Use a print style sheet

```
<link href="/path/to/print.css" media="print" rel="stylesheet" />
```

CSS for print

- Use media queries to improve layout
  - You can use the CSS `@media` at-rule to set a different appearance for your webpage when it is printed on paper and when it is displayed on the screen
  - The print option sets the styles that will be used when the content is printed. Add additional print-only styles as you see fit

```
@media print {
  /* All your print styles go here */
  #header,
  #footer,
  #nav {
    display: none !important; /* this will hide areas you don't want printed */
  }
}
```

Exercise

- Join a breakout room
- Download the previous example code from Moodle
- Modify the CSS and text as you like
- You have 35 minutes
- Lecturer will visit each room in turn, etc...
- Will start next topic on the hour

Box Model

Box Model

Background

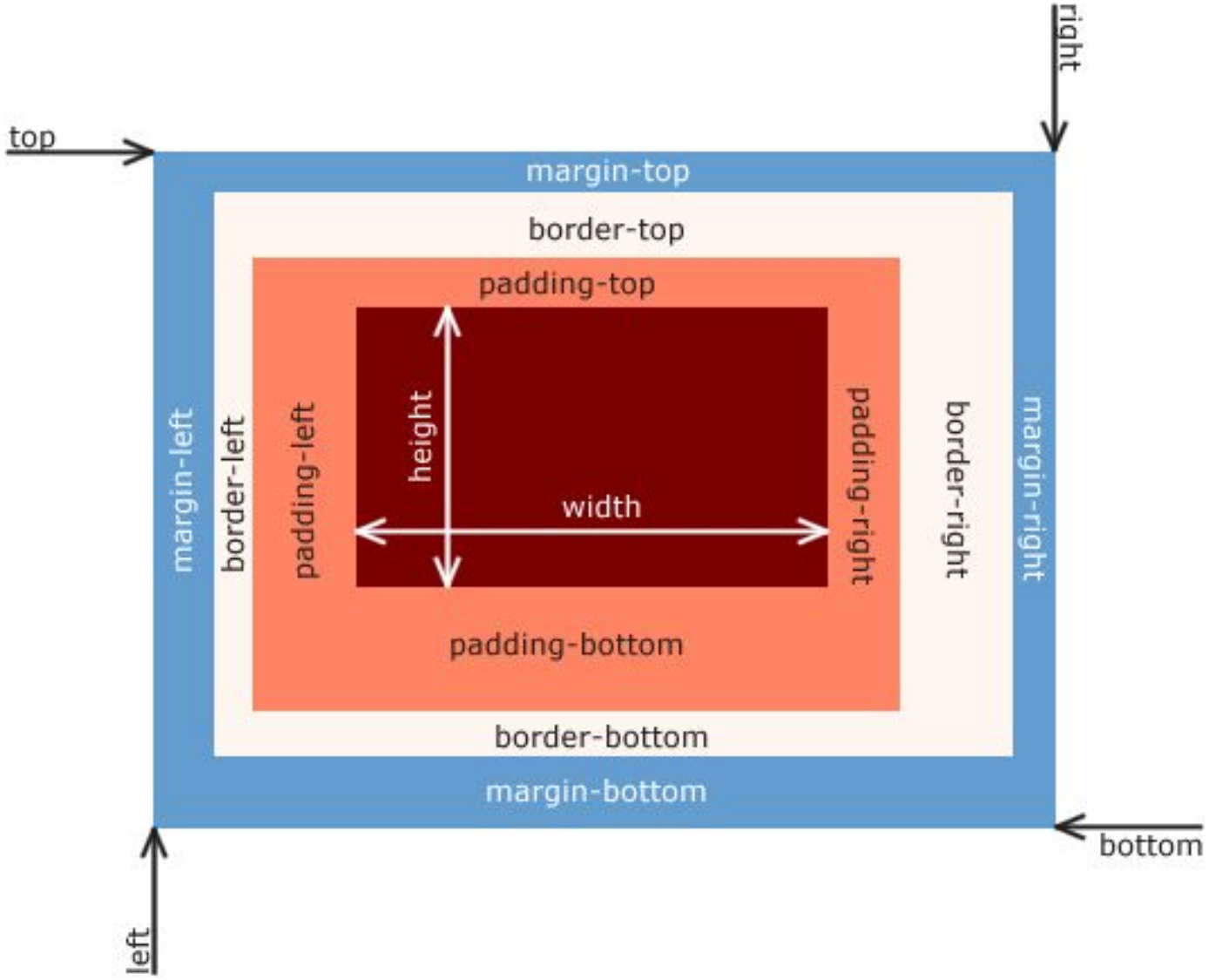
- The CSS box model refers to how HTML elements are modeled in browser engines and how the dimensions of those HTML elements are derived from CSS properties
- It is a fundamental concept for the composition of HTML webpages
- The guidelines of the box model are described by web standards World Wide Web Consortium (W3C) specifically the CSS Working Group
- For much of the late-1990s and early 2000s there had been non-standard compliant implementations of the box model in mainstream browsers
- With the advent of CSS2 in 1998, which introduced the box-sizing property, the problem had mostly been resolved

Box Model

All HTML elements can be considered "boxes", this includes div tag, p tag, or a tag. Each of those boxes has five modifiable dimensions:

- the **height** and **width describe dimensions of the actual content of the box (text, images, ...)
- the **padding** describes the space between this content and the border of the box
- the **border** is any kind of line (solid, dotted, dashed...) surrounding the box, if present
- the **margin** is the space around the border

Box Model


top
right
margin-top
border-top
padding-top
height
width
margin-left
border-left
padding-left
padding-right
border-right
margin-right
padding-bottom
border-bottom
margin-bottom
bottom
left

Box Model: box-sizing

Two different ways to calculate the size of the box depending on the value of the `box-sizing` property

- `content-box` gives you the default CSS box-sizing behavior. If you set an element's width to 100 pixels, then the element's content box will be 100 pixels wide, and the width of any border or padding will be added to the final rendered width, making the element wider than 100px
- `border-box` tells the browser to account for any border and padding in the values you specify for an element's width and height. If you set an element's width to 100 pixels, that 100 pixels will include any border or padding you added, and the content box will shrink to absorb that extra width. This typically makes it much easier to size elements. box-sizing: border-box is the default styling that browsers use for the `<table>`, `<select>`, and `<button>` elements, and for `<input>` elements whose type is radio, checkbox, reset, button, submit, color, or search

Box Model: box-sizing

- `box-sizing: content-box;`
  - The total width of a box is therefore `left-margin` + `left-border` + `left-padding` + `width` + `right-padding` + `right-border` + `right-margin`
  - The total height of a box equals `top-margin` + `top-border` + `top-padding` + `height` + `bottom-padding` + `bottom-border` + `bottom-margin` For example:

```
.myClass {
  box-sizing: content-box;
  width: 600px;
  height: 200px;
  padding: 10px;
  border: solid 10px black;
  margin: 10px;
}
```

would specify the box dimensions of each block belonging to 'myClass'. Moreover, each such box will have total height 260px and width 660px

Box Model: box-sizing

- `box-sizing: border-box;`
  - The total width of a box is therefore `left-margin` + `width` + `right-margin`
  - The total height of a box equals `top-margin` + `height` + `bottom-margin`. For example:

```
.myClass {
  box-sizing: border-box;
  width: 600px;
  height: 200px;
  padding: 10px;
  border: solid 10px black;
  margin: 10px;
}
```

would specify the box dimensions of each block belonging to 'myClass'. Moreover, each such box will have total height 220px and width 620px

Height/Width

The `height` and `width` properties are used to set the height and width of an element

The `height` and `width` properties do not include padding, borders, or margins. It sets the height/width of the area inside the padding, border, and margin of the element

CSS height and width Values

The `height` and `width` properties may have the following values:

- auto - This is default. The browser calculates the height and width
- length - Defines the height/width in px, cm, etc...
- % - Defines the height/width in percent of the containing block
- initial - Sets the height/width to its default value
- inherit - The height/width will be inherited from its parent value

Height/Width

There are also minimum and maximum versions of both properties which are used to set limits on both properties

These can be used to solve issues when displaying content with larger and smaller viewports

Property	Description
height	Sets the height of an element
max-height	Sets the maximum height of an element
max-width	Sets the maximum width of an element
min-height	Sets the minimum height of an element
min-width	Sets the minimum width of an element
width	Sets the width of an element

Borders

Borders can be specified in a number of different ways

The `border` property is a shorthand property for the following individual border properties:

- border-width
- border-style (required)
- border-color

```
p {
  border: 5px solid red; /* sets border for all four sides */
  /* or */
  border-width: 5px;
  border-style: solid;
  border-color: red;
  /* or */
  border-top: 5px solid red; /* can also be set individually */
  border-right: 5px solid red;
  border-bottom: 5px solid red;
  border-left: 5px solid red;
}
```

Rounded Borders

The `border-radius` property is used to add rounded borders to an element:

```
p {
  border: 2px solid red;
  border-radius: 5px; /* set border radius for all four sides */
  /* or */
  border-top-left: 5px; /* can also be set individually */
  border-top-right: 5px;
  border-bottom-left: 5px;
  border-bottom-right: 5px;
}
```

Padding

Padding is used to create space around an element's content, inside of any defined borders

There are properties for setting the padding for each side of an element: `padding-top` , `padding-right` `padding-bottom` , `padding-left` . All the padding properties can have the following values:

- **length** - specifies a padding in px, pt, cm, etc..
- **%** - specifies a padding in % of the width of the containing element
- **inherit** - specifies that the padding should be inherited from the parent element
- Note: Negative values are not allowed

```
div {
  padding-top: 50px;
  padding-right: 30px;
  padding-bottom: 50px;
  padding-left: 80px;
  padding: 25px 50px 75px 100px; /* top, right, bottom, left */
  padding: 25px 50px 75px; /* top, right & left, bottom */
  padding: 25px 50px; /* top & bottom, left & right */
  padding: 25px;   /* all sides */
}
```

Margins

Margins are used to create space around an element's content, outside of any defined borders

There are properties for setting the margin for each side of an element: `margin-top`, `margin-right`, `margin-bottom`, `margin-left` . All the margin properties can have the following values:

- **length** - specifies a margin in px, pt, cm, etc.
- **%** - specifies a margin in % of the width of the containing element
- **inherit** - specifies that the margin should be inherited from the parent element
- Note: Negative values are not allowed

```
div {
  margin-top: 50px;
  margin-right: 30px;
  margin-bottom: 50px;
  margin-left: 80px;
  margin: 25px 50px 75px 100px; /* top, right, bottom, left */
  margin: 25px 50px 75px; /* top, right & left, bottom */
  margin: 25px 50px; /* top & bottom, left & right */
  margin: 25px;    /* all sides */
}
```

Margins

Auto

- You can set the margin property to auto to horizontally center the element within its container
- The element will then take up the specified width, and the remaining space will be split equally between the left and right margins

```
div {
  width: 300px;
  margin: 0 auto;
  border: 1px solid red;
}
```

Margin Collapse

- Top and bottom margins of elements are sometimes collapsed into a single margin that is equal to the largest of the two margins
- This does not happen on left and right margins! Only top and bottom margins!

For example:

```
.myClass {
  margin: 30px auto;
  background-color: aqua;
  width: 600px;
  height: 200px;
  padding: 10px;
  border: solid 10px black;
}
.middleClass{
  width:100%;
  height: 25px;
  margin: 20px auto; /* resulting margin is 30px and not 50px (20+30px) */
}
```

Display

The display property is the most important CSS property for controlling layout and specifies if/how an element is displayed

Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is block or inline

- **Block** level elements
  - A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can)
  - Examples include: `<di v>, <h1> - <h6>, <p>, <form>, <header>, <footer>, <section>`
- **Inline** elements
  - An inline element does not start on a new line and only takes up as much width as necessary
  - Examples include: `<span>`, `<a>`, `<img>`

Display values

- `display: none;` is commonly used with JavaScript to hide and show elements without deleting and recreating them. Do not confuse `display: none;` with `visibility: hidden;`. The latter also hides the element, but leaves the space it would normally take open or empty
- `display: inline-block;` means the element will not start on a new line and it will take up the space to display the contained content. However you **can set** the width and height of the element, for example a `<a>` which contains another block element such as a `<div>`
- `display: table;` is rarely use as a display value of table these days, but it's still important to know. It was more useful in the past because you would use it for layouts before the advent of floats, Flex, and Grid
- `display: inherit;` makes the element inherit the display property of its parent. So, if you have a `<span>` tag inside a div and you give the span tag a display of inherit, it turns it from inline to block element
- `display: initial;` sets the display property of an element to its default value. So, if you set the display property of a span to initial, it remains inline, and if you set the same value for a div, it remains block
- there are other values such as `display: flex;` , `display: grid;` which we will cover in a future lecture

Layout

The `position` property specifies the type of positioning method used for an element

There are five different `position` values:

- static
- relative
- fixed
- absolute
- sticky

Elements are then positioned using the top, bottom, left, and right properties

However, these properties will not work unless the position property is set first

They also work differently depending on the position value

Layout

- `position: static;`
  HTML elements are positioned static by default

  Static positioned elements are not affected by the top, bottom, left, and right properties

  An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page:

- `position: relative;`
  An element with position: relative; is positioned relative to its normal position

  Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position

  Other content will not be adjusted to fit into any gap left by the element

Layout

- `position: fixed;`
  An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled

  The top, right, bottom, and left properties are used to position the element

  A fixed element does not leave a gap in the page where it would normally have been located

- `position: absolute;`
  An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed)

  However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling

  Note: Absolute positioned elements are removed from the normal flow, and can overlap elements

Layout

- `position: sticky;`

  An element with position: sticky; is positioned based on the user's scroll position

  A sticky element toggles between relative and fixed, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed)

Align

- There are numerous ways to align elements in relation to it's parent.
- Center Align Elements
  To horizontally center a block element (like), use `margin: 0 auto;`

  Setting the width of the element will prevent it from stretching out to the edges of its container.

  The element will then take up the specified width, and the remaining space will be split equally between the two margins

```
.center {
  margin: 0 auto;
  width: 50%;
  border: 3px solid green;
  padding: 10px;
}
```

Align

- Center Align
  Text
  To just center the text inside an element, use `text-align: center;`

```
.center {
  text-align: center;
  border: 3px solid green;
}
```

- Center an Image

  To center an image, set left and right margin to auto and make it into a block element:

```
img {
  display: block;
  margin: auto;
  width: 40%;
}
```

Align

- Left and Right Align - Using position
  One method for aligning elements is to use `position: absolute;` :

```
.right {
  position: absolute;
  right: 0px; /* changing to 'left: 0px;' will left align the element */
  width: 300px;
  border: 3px solid #73AD21;
  padding: 10px;
}
```

- Center Vertically - Using padding
  To center both vertically and horizontally, use padding and text-align: center:

```
.center {
  padding: 70px 0; /* can also use percentages */
  border: 3px solid green;
  text-align: center;
}
```

Align

- Center Vertically - Using line-height

  Another trick is to use the line-height property with a value that is equal to the height property:

```
.center {
  line-height: 200px;
  height: 200px;
  border: 3px solid green;
  text-align: center;
}

/* If the text has multiple lines, add the following: */
.center p {
  line-height: 1.5;
  display: inline-block;
  vertical-align: middle;
}
```

Align

- Center Vertically - Using position & transform

  If padding and line-height are not options, another solution is to use positioning and the transform property:

```
.center {
  height: 200px;
  position: relative;
  border: 3px solid green;
}

.center p {
  margin: 0;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

The `transform` property will be covered in a future lecture

Floating, clearing

An older method for aligning elements is to use the float property:

```
  .right {
    float: right; /* left*/
    width: 300px;
    border: 3px solid #73AD21;  padding: 10px;
  }
```

If an element is taller than the element containing it, and it is floated, it will overflow outside of its container. You can use the "clearfix hack" to fix this (see demo) by adding the `.clearfix` class to the containing element

```
.clearfix::after {
  content: "";
  clear: both;
  display: table;
}
```

Probably best to use it only when you need to wrap text around an image(s). For layouts use Grid or Flexbox.

Read more ...

Overflow

The `overflow` property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area

The overflow `property` has the following values:

- visible - Default. The overflow is not clipped. The content renders outside the element's box
- hidden - The overflow is clipped, and the rest of the content will be invisible
- scroll - The overflow is clipped, and a scrollbar is added to see the rest of the content
- auto - Similar to scroll, but it adds scrollbars only when necessary

Note: The overflow property only works for block elements with a specified height

Note: In OS X Lion (on Mac), scrollbars are hidden by default and only shown when being used (even though "overflow:scroll" is set)

Overflow

All CSS `overflow` Properties

Property	Description
overflow	Specifies what happens if content overflows an element's box
overflow-wrap	Specifies whether or not the browser can break lines with long words, if they overflow its container
overflow-x	Specifies what to do with the left/right edges of the content if it overflows the element's content area
overflow-y	Specifies what to do with the top/bottom edges of the content if it overflows the element's content area

Z-index

When elements are positioned, they can overlap other elements

The `z-index` property specifies the stack order of an element (which element should be placed in front of, or behind, the others)

- An element can have a positive or negative stack order:

```
img {
  position: absolute;
  left: 0px;
  top: 0px;
  z-index: -1;
}
```

Note: `z-index` only works on positioned elements (`position: absolute;`, `position: relative;` , `position: fixed;` , or `position: sticky;`) and flex items (elements that are direct children of `display: flex` elements)

If two positioned elements overlap each other without a z-index specified, the element defined **last in the HTML code** will be shown on top

CSS Resources

- W3 Schools - very good introductory resource
- Mozilla Developer Network (MDN) - the definitive web resource
- CSS Tricks - Great articles and guides on CSS

Summary

- Overview and Principles of CSS Typography & Media styles
- Box Model
- Complete the remaining exercises before next class