CSS
Cascading Style Sheets
CSS
Cascading Style Sheets

Before we start:
- What do you need to know before starting?
- HTML and Basic knowledge of Document Object Model.

What we will cover
- What CSS is,
- CSS Syntax,
- How we can use CSS to style our webpages.

How this lecture contributes to the course

HTML
The structure of the webpage

CSS
The styling of the webpage

JavaScript
Makes a webpage interactive
Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML.[1]
HTML
Makes websites

CSS
Makes Nice websites
DOM

Document Object Model
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </body>
</html>
How CSS works

Load HTML → Parse HTML → Create DOM Tree → Display

Load CSS → Attach style to DOM nodes
1. `selector` {
2.     `property: value;`
3. }

1. `h1` {
2.     `font-size: 18px;`
3. }

CSS Syntax
Type Selector

The CSS type selector matches elements by node name. In other words, it selects all elements of the given type within a document.

```html
1. <p>This is a paragraph.</p>
```

```css
1. p {
2.   color: red;
3. }
```

This is a paragraph.
Class selector

The CSS class selector matches elements based on the contents of their `class` attribute.

```html
1. <p class="text">This is a paragraph.</p>
```

```css
1. .text {
2.   color: green;
3. }
```

This is a paragraph.
Class selector

Multiple classes

1. `<p class="text grey">This is a paragraph.</p>`

```
1. .text.grey {
2. color: grey;
3. }
```

This is a paragraph.
ID selector

The CSS ID selector matches an element based on the value of its `id` attribute.

Only one `id` attribute is allowed per one HTML element and it’s value has to be unique on that page.

```html
1. <p id="my_paragraph">This is a paragraph.</p>
```

```css
1. #my_paragraph {
2.   font-weight: bold;
3. }
```

This is a paragraph.
Attribute selector

The CSS attribute selector matches elements based on the presence or value of a given attribute.

```
1. <p title="foo">This is a foo paragraph.</p>
2. <p title="bar">This is a bar paragraph.</p>

1. p[title] {
2.     font-weight: bold;
3. }
4. p[title="bar"] {
5.     font-weight: lighter;
6. }
```

This is a foo paragraph.
This is a bar paragraph.
Universal selector

The CSS universal selector (*) matches elements of any type.

1. `<div>`This is a div.</div>
2. `<p>`This is a paragraph.</p>

1. `*` {
2.  `font-family: "Impact";`
3.  `}`

This is a div

This is a paragraph.
Adjacent sibling combinator

The adjacent sibling combinator (+) separates two selectors and matches the second element only if it immediately follows the first element, and both are children of the same parent element.

1. `<div>This is a div.</div>`
2. `<p>This is paragraph 1.</p>`
3. `<p>This is paragraph 2.</p>`

```
1. div + p {
2.     color: blue;
3. }
```

This is a div
This is paragraph 1.
This is paragraph 2.
General sibling combinator

The general sibling combinator (~) separates two selectors and matches the second element only if it follows the first element (though not necessarily immediately), and both are children of the same parent element.
Child combinator

The child combinator (>) is placed between two CSS selectors. It matches only those elements matched by the second selector that are the immediate children of elements matched by the first.

```html
1. <div>
2.   <span>Span inside div.
3.   </span>
4.   </div>
5. </div>
6. <span>Not outside div.</span>

1. span {
2.   background-color: yellow;
3. }
4. div > span {
5.   background-color: green;
6. }
```

Descendant combinator

The descendant combinator — typically represented by a single space ( ) character — combines two selectors such that elements matched by the second selector are selected if they have an ancestor element matching the first selector. Selectors that utilize a descendant combinator are called descendant selectors.

```
1. <div>
2.   <span>Span inside div.
3.   <span>Span inside span.</span>
4.   </span>
5. </div>
6. <span>Not outside div.</span>
```
Pseudo-classes

Pseudo-classes let you apply a style to an element not only in relation to the content of the document tree, but also in relation to external factors like the history of the navigator (:visited, for example), the status of its content (like :checked on certain form elements), or the position of the mouse (like :hover, which lets you know if the mouse is over an element or not).

```
1. button:hover{
2.       background-color: yellow;
3. }
4. a:active{
5.       text-decoration: underline;
6. }
7. ul li:first-child{
8.       margin-left: 15px;
9. }
```
A CSS pseudo-element is a keyword added to a selector that lets you style a specific part of the selected element(s).

```css
1. p::first-line{
2.     font-size: 20px;
3. }
4. a::after{
5.     content: "→";
6. }
7. input[type="text"]::focus{
8.     border: 1px solid red;
9. }
```
Stylesheets **cascade** — at a very simple level this means that the order of CSS rules matter; when two rules apply that have equal specificity the one that comes last in the CSS is the one that will be used.[1]
1. `<h1>My Heading</h1>`

```
1. h1 {
2.   color: blue;
3. }
4. h1 {
5.   color: green;
6. }
```
Specificity rules

Most Important

!important

Inline CSS

ID

Classes, Pseudo-classes, Attributes

Least Important

Element, Pseudo-element
Specificity

1. `<h1 class="heading">My Heading</h1>`

```html
1. .heading {
2.   color: blue;
3. }
4. h1 {
5.   color: green;
6. }
```

My Heading
1. `<h1 class="heading">My Heading</h1>`

```css
1. .heading {
2.     color: blue;
3. }
4. h1 {
5.     color: red !important;
6. }
```

My Heading
1. /* One Line */
2. */
3. .heading {
4. color: blue;
5. }
6. */
Embedding CSS: External CSS

1. <!--DOCTYPE html--> 
2. <html> 
3. <head> 
4. <link rel="stylesheet" type="text/css" href="mystyle.css"> 
5. </head> 
6. <body> 
7. ... 
8. </body> 
9. </html>
1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `body {`  
6. `  background-color: linen;`  
7. `}
8. `</style>`
9. `</head>`
10. `<body>`
11. `...`
12. `</body>`
13. `</html>`
1. `<p style="color: blue;">This is a paragraph.</p>`

This is a paragraph.
The Box Model

Everything in CSS has a box around it, and understanding these boxes is key to being able to create layouts with CSS, or to align items with other items.

https://www.w3schools.com/css/css_boxmodel.asp
The Box Model

Everything in CSS has a box around it, and understanding these boxes is key to being able to create layouts with CSS, or to align items with other items.

The CSS box model is essentially a box that wraps around every HTML element. It consists of:

- **Content**, where text and images appear
- **Padding**, clears an area around the content.
- **Border**, goes around the padding and content.
- **Margin** - Clears an area outside the border.

Note: the padding and margin are transparent.

https://www.w3schools.com/css/css_boxmodel.asp

```css
1. div{
2.   width: 300px;
3.   padding: 50px;
4.   border: 15px;
5.   margin: 20px;
}
```
FlexBox
FlexBox

- The Flexbox Layout module aims at providing a more efficient way to lay out, align and distribute space among items in a container.
- Flexbox is a whole module that includes a whole set of properties. Some related to the container others to the items.

Simple and well described tutorial: [https://css-tricks.com/snippets/css/a-guide-to-flexbox/](https://css-tricks.com/snippets/css/a-guide-to-flexbox/)
FlexBox

Container’s properties
- Display
- flex-direction
- flex-wrap
- justify-content
- align-items

Items’ properties
- flex-grow
- flex-shrink
- align-self

Simple and well described tutorial: https://css-tricks.com/snippets/css/a-guide-to-flexbox/
FlexBox

Container’s properties
- Display
- flex-direction
- flex-wrap
- justify-content
- align-items

Items’ properties
- flex-grow
- flex-shrink
- align-self

Simple and well described tutorial: [https://css-tricks.com/snippets/css/a-guide-to-flexbox/](https://css-tricks.com/snippets/css/a-guide-to-flexbox/)
FlexBox

Container’s properties
- Display
- flex-direction
- flex-wrap
- justify-content
- align-items

Items’ properties
- flex-grow
- flex-shrink
- align-self

Simple and well described tutorial: https://css-tricks.com/snippets/css/a-guide-to-flexbox/
FlexBox

Container’s properties

- Display
- flex-direction
- flex-wrap
- justify-content
- align-items

Items’ properties

- flex-grow
- flex-shrink
- align-self

Simple and well described tutorial: https://css-tricks.com/snippets/css/a-guide-to-flexbox/
FlexBox

Container’s properties

- Display
- flex-direction
- flex-wrap
- justify-content
- align-items

Items’ properties

- flex-grow
- flex-shrink
- align-self

Simple and well described tutorial: [https://css-tricks.com/snippets/css/a-guide-to-flexbox/](https://css-tricks.com/snippets/css/a-guide-to-flexbox/)
### FlexBox

**Container’s properties**
- Display
- flex-direction
- flex-wrap
- justify-content
- align-items

**Items’ properties**
- flex-grow
- flex-shrink
- align-self

Simple and well described tutorial: [https://css-tricks.com/snippets/css/a-guide-to-flexbox/](https://css-tricks.com/snippets/css/a-guide-to-flexbox/)
FlexBox

Container's properties
- Display
- flex-direction
- flex-wrap
- justify-content
- align-items

Items' properties
- flex-grow
- flex-shrink
- align-self

Simple and well described tutorial: [https://css-tricks.com/snippets/css/a-guide-to-flexbox/](https://css-tricks.com/snippets/css/a-guide-to-flexbox/)
FlexBox

Container’s properties
- Display
- flex-direction
- flex-wrap
- justify-content
- align-items

Items’ properties
- flex-grow
- flex-shrink
- align-self

A nice game to master FlixBox: https://flexboxfroggy.com/

Simple and well described tutorial: https://css-tricks.com/snippets/css/a-guide-to-flexbox/
Positioning
Positioning

**Static**

Static positioning is the default that every element gets — it just means "put the element into its normal position in the document layout flow".

**Relative**

Relative positioning is very similar to static positioning, except that once the positioned element has taken its place in the normal layout flow, you can then modify its final position, including making it overlap other elements on the page.
Positioning

Absolute

An absolutely positioned element no longer exists in the normal document layout flow. Instead, it sits on its own layer separate from everything else. This is very useful: it means that we can create isolated UI features that don't interfere with the position of other elements on the page.

Fixed

whereas absolute positioning fixes an element in place relative to the `<html>` element or its nearest positioned ancestor, fixed positioning fixes an element in place relative to the browser viewport itself. This means that you can create useful UI items that are fixed in place, like persisting navigation menus.
HTML

```html
<h1>Static positioning</h1>
<p>I am a basic block element.</p>
<p class="positioned">I am a positioned element.</p>
<p>I am another static element</p>
```

CSS

```css
body{
    width: 500px;
}
p {
    background: aqua;
    border: 3px solid blue;
    padding: 10px;
    margin: 10px;
}
.positioned {
    position: static;
    background: yellow;
    top: 30px;
    left: 30px;
}
```
Relative positioning

I am a basic block element.

I am a positioned element.

I am another static element
Absolute positioning

I am a positioned element.

I am a basic block element.

I am another static element.
**Fixed positioning**

```html
<h1>Fixed positioning</h1>
<p>I am a basic block element.</p>
<p class="positioned">I am a positioned element.</p>
<p>I am another static element</p>
```

```css
body{
  width: 500px;
}
p {
  background: aqua;
  border: 3px solid blue;
  padding: 10px;
  margin: 10px;
}
.positioned {
  position: fixed;
  background: yellow;
  top: 50px;
  left: 30px;
}
```
Z-Index

Web pages also have a z-axis: an imaginary line that runs from the surface of your screen, towards your face. `z-index` values affect where positioned elements sit on that axis;

- Positive values move them higher up the stack.
- Negative values move them lower down the stack.
- By default, positioned elements all have a `z-index` of `auto`, which is effectively 0.
Relative positioning

HTML

```html
<h1>Relative positioning</h1>
<p>I am a basic block element.</p>
<p class="positioned">I am a positioned element.</p>
<p>I am another static element</p>
```

CSS

```css
body{
  width: 500px;
}
p {
  background: aqua;
  border: 3px solid blue;
  padding: 10px;
  margin: 10px;
}
.positioned {
  position: relative;
  background: yellow;
  top: 30px;
  left: 30px;
  z-index: -1;
}
```
Values and Units
<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;integer&gt;</code></td>
<td>An <code>&lt;integer&gt;</code> is a whole number such as 1024 or -55.</td>
</tr>
<tr>
<td><code>&lt;number&gt;</code></td>
<td>A <code>&lt;number&gt;</code> 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.</td>
</tr>
<tr>
<td><code>&lt;dimension&gt;</code></td>
<td>A <code>&lt;dimension&gt;</code> is a <code>&lt;number&gt;</code> with a unit attached to it, for example 45deg, 5s, or 10px. <code>&lt;dimension&gt;</code> is an umbrella category that includes the <code>&lt;length&gt;</code>, <code>&lt;angle&gt;</code>, <code>&lt;time&gt;</code>, and <code>&lt;resolution&gt;</code> types.</td>
</tr>
<tr>
<td><code>&lt;percentage&gt;</code></td>
<td>A <code>&lt;percentage&gt;</code> 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.</td>
</tr>
</tbody>
</table>
## Absolute length units

<table>
<thead>
<tr>
<th>Unit</th>
<th>Name</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>cm</td>
<td>Centimeters</td>
<td>$1\text{cm} = \frac{96\text{px}}{2.54}$</td>
</tr>
<tr>
<td>mm</td>
<td>Millimeters</td>
<td>$1\text{mm} = \frac{1}{10}\text{th of 1cm}$</td>
</tr>
<tr>
<td>Q</td>
<td>Quarter-millimeters</td>
<td>$1\text{Q} = \frac{1}{40}\text{th of 1cm}$</td>
</tr>
<tr>
<td>in</td>
<td>Inches</td>
<td>$1\text{in} = 2.54\text{cm} = 96\text{px}$</td>
</tr>
<tr>
<td>pc</td>
<td>Picas</td>
<td>$1\text{pc} = \frac{1}{16}\text{th of 1in}$</td>
</tr>
<tr>
<td>pt</td>
<td>Points</td>
<td>$1\text{pt} = \frac{1}{72}\text{th of 1in}$</td>
</tr>
<tr>
<td>px</td>
<td>Pixels</td>
<td>$1\text{px} = \frac{1}{96}\text{th of 1in}$</td>
</tr>
</tbody>
</table>
Relative length units

<table>
<thead>
<tr>
<th>Unit</th>
<th>Relative to</th>
</tr>
</thead>
<tbody>
<tr>
<td>em</td>
<td>Font size of the parent element.</td>
</tr>
<tr>
<td>ex</td>
<td>x-height of the element's font.</td>
</tr>
<tr>
<td>rem</td>
<td>Font size of the root element.</td>
</tr>
<tr>
<td>lh</td>
<td>Line height of the element.</td>
</tr>
<tr>
<td>vw</td>
<td>1% of the viewport's width.</td>
</tr>
<tr>
<td>vh</td>
<td>1% of the viewport's height.</td>
</tr>
<tr>
<td>vmin</td>
<td>1% of the viewport's smaller dimension.</td>
</tr>
<tr>
<td>vmax</td>
<td>1% of the viewport's larger dimension.</td>
</tr>
</tbody>
</table>

[https://www.w3schools.com/cssref/css_units.asp](https://www.w3schools.com/cssref/css_units.asp)
Color

There are many ways to specify color in CSS, some of which are more recently implemented than others. The same color values can be used everywhere in CSS, whether you are specifying text color, background color, or whatever else.

- **Keywords:** `blue; black; white...`
- **Hexadecimal:** `#0083E0; #000000; #FFFFFF...`
- **RGB:** `rgb(0,0,0); rgb(255,255,255)...`
- **RGBA:** `rgba(0,131,224,0.5)...`
- **HSL:** `hsl(0,0%,0%); hsl(0,0%,100%)...`
- **HSLA:** `hsla(213, 100%, 50%, 0.5)...`
@rules

at-rules
An at-rule is a CSS statement that instructs CSS how to behave. They begin with an at sign, `@`, followed by an identifier and includes everything up to the next semicolon, `;` or the next CSS block, whichever comes first.

1. `@IDENTIFIER (RULE);`
The `@import` CSS at-rule is used to import style rules from other style sheets. These rules must precede all other types of rules, except `@charset` rules:

1. `@import url("fineprint.css") print;`
2. `@import 'custom.css';`
3. `@import "common.css" screen;`
4. `@import url('landscape.css') screen and (orientation:landscape);`
@media

The @media CSS at-rule can be used to apply part of a style sheet based on the result of one or more media queries. With it, you specify a media query and a block of CSS to apply to the document if and only if the media query matches the device on which the content is being used.

```css
@media (height > 600px) {
    body {
        line-height: 1.4;
    }
}

@media (400px <= width <= 700px) {
    body {
        line-height: 1.1;
    }
}
```
References

https://www.w3.org/Style/CSS/Overview.en.html
Questions?

Next: Introduction to Javascript