



# **Analysis of Cascading Style Sheets**

Institution:

Vienna University of Economics and Business

Institute for Management and Information Systems

Author:

Volkwin Haselbauer

Course:

Wirtschaftsinformatik-Projektseminar 4135

Prof. Rony Flatscher

Vienna, 14.06.2023

## Table of Contents

Abstract .....	II
List of Figures.....	III
1 Introduction .....	1
2 Definition.....	2
3 History .....	3
4 Basics of HTML .....	4
5 Syntax .....	5
5.1 Selector group .....	7
5.1.1 Selector types .....	7
5.1.2 Grouping and compounding. ....	10
5.1.3 Combinators .....	10
5.2 Properties .....	12
5.3 Values .....	12
5.4 Units .....	13
5.5 Value functions .....	14
5.6 Custom properties.....	14
6 Concepts .....	15
6.1 Cascading .....	15
6.2 Specificity .....	15
6.3 Inherence .....	16
6.4 Box Models .....	16
6.5 Layout methods.....	17
6.6 Media Queries.....	20
6.7 Accessibility.....	21
7 Architecture .....	23
7.1 Implementation in HTML.....	23
7.2 Implementation in XML .....	24
7.3 Best Practices .....	24
7.4 CSS methodologies.....	25
7.5 Frameworks .....	27
8 Outlook .....	28
9 Source Index .....	29
10 Appendix .....	33

## **Abstract**

Cascading Style Sheets (CSS) have become an integral part of modern web design. This thesis delves into the various aspects of CSS and its profound impact on responsiveness and accessibility.

The paper begins by providing an overview of the fundamental syntax and principles of CSS, exploring its selectors, properties, and concepts. A review of existing on CSS best practices and their impact on maintainability and scalability. It explores the application of CSS frameworks, advanced layout techniques, and animations to create visually appealing and engaging interfaces. The impact of CSS on responsive design is also examined, such as media queries.

Additionally, the thesis explores the influence of CSS on user experience. It investigates the role of CSS in improving readability and accessibility.

In conclusion, this thesis contributes to an understanding of CSS and its potential for elevating the styling of markup language documents. It gives an outlook of the further development of CSS.

## List of Figures

Figure 1: Syntax of a CSS rule .....	5
Figure 2: Result of the media query example.....	7
Figure 3: Example of an active hover state.....	8
Figure 4: Output of the ":first-child" pseudo class example.....	9
Figure 5: Output of the "::first-letter" pseudo element example.....	9
Figure 6: Output of the descendant combinat example.....	10
Figure 7: Output of the child combinator example .....	11
Figure 8: Output of the sibling combinator example .....	11
Figure 9: Output of the next siblings combinator example.....	12
Figure 10: CSS Box Model .....	17
Figure 11: Output of the flexible box layout example .....	18
Figure 12: Output of the grid example.....	19
Figure 13: Output of the grid example with grid lines .....	20

## 1 Introduction

Cascading Style Sheets, short CSS, is used in nearly every website in the modern World Wide Web (WWW) to shape its appearance („Usage Statistics of CSS for Websites, June 2023“, o. J.). It provides an agile and powerful way to design the web pages to adapt to different screen sizes and input types, which gets more important with the large number of mobile devices („CSS“, 2023).

The history of CSS starting with its release in 1996 by the World Wide Web Consortium (W3C) as CSS level 1, to the now modular CSS level 3 will most likely not have a successor but will be updated continuously by packages („A brief history of CSS until 2016“, o. J.).

The basic idea and structure of CSS using selectors and properties to assign values and thereby styles to elements of a document. Using different layout options, selecting specific screens, color schemes and making the web accessible for blind people, is all possible with CSS. It can be used in conjunction with Markup languages like HTML, where its vastly used for web development. Modules, like flexbox shifted the control of structuring the rendered document layout to CSS („CSS Snapshot 2023“, o. J.).

This paper aims to give an overview of the syntax of CSS, its concepts and the future of it. CSS is in development since 1996 and new modules are continuously added or improved. Therefore, an indexing of selectors, properties or other aspects would make a static paper like this obsolete in a short amount of time. Hence the common syntax and concepts are expounded with the help of HTML examples. Consequently, there is no claim to completeness. This paper mostly focuses on the use of CSS with HTML, as it was introduced for styling the web, but the implementation with XML is also covered. For this paper a basic knowledge of markup languages is recommended, even though there is a short introduction to HTML.

The examples in this paper are combined into one HTML and one CSS code section, to enable the reader to copy the code into a file and follow the examples in their own browser.

As a source served, to a big part, the official documentation of the W3C for CSS and the documentation on CSS from the Mozilla Corporation, developing company of the web browser Firefox.

## 2 Definition

Cascading Style Sheets is a language to style a document written in a markup language. Using CSS decouples the presentation of the document from its content, allowing a separate development and makes it possible to apply styles to similar documents, without repetition. Responsive designs, where the presentation is different on different devices and screen sizes, can also be done with CSS. Documents are also made accessible for people with disabilities, by optimizing it for screen readers or braille. It was developed by the W3C to standardize the styling of web pages, for a more accessible development and implementation in web browsers („Cascading Style Sheets, designing for the Web – Chapter 2: CSS“, o. J.; „CSS“, 2023; „CSS Snapshot 2023“, o. J.).

Markup languages (ML) structures the content, like text and images, of the document and adds semantic to make it useable for computers. This is achieved by the usage of tags and rules around them, to ultimately displaying the content in suitable manner. Hyper Text Markup Language (HTML) is one of the most common markup languages, with HTML documents web pages can be build. For data storage and distribution Extensible Markup Language (XML) is used, which has a stricter set of rules than HTML. Beside these two many more and combination of these have emerged („Extensible Markup Language (XML)“, o. J.; „HTML: The Markup Language (an HTML language reference)“, o. J.).

A User agent is the software that renders the document and enables the interaction with it, for the use of the WWW these are web browsers („User Agent Accessibility Guidelines (UAAG) 2.0“, o. J.).

The W3C further evolves CSS by having working groups develop new modules or improve existing modules. They are first published as working draft, this can then mature over candidate recommendations to proposed recommendations and to the final form a W3C recommendation. Developing companies of user agents may already implement modules that did not yet reach the W3C recommendation („W3C Process Document“, o. J.).

### 3 History

In the early 90s the WWW consisted of plain HTML documents, how these were displayed depended on which browser was used. There were many approaches and different philosophies on how and who should define the appearance of the web pages. Each browser handled the presentation differently with their own style sheet languages and customization options, to leave the styling to the users. But authors also wanted to have a word in how their documents are displayed („A brief history of CSS until 2016“, o. J.)

Håkon Wium Lie first proposed CSS in 1994 and further developed it with Bert Bos. Many other styling languages emerged, but CSS solved the problem the others could not. It enabled the author, reader, and browser to influence the style of the web page („A brief history of CSS until 2016“, o. J.).

In 1995 the first implementation of CSS was presented on a WWW conference. One year later the W3C released, alongside with HTML specifications, the CSS level 1 W3C Recommendation. The adaptation of browsers to support CSS fully was started by Microsoft with the Internet Explorer 3 and others slowly following („A brief history of CSS until 2016“, o. J.).

In 1998 CSS level 2 was published, with updated properties and the introduction of media types. Since 2000 there is CSS level 3, which is now updated with modules. This means changes are smaller and modules can be adopted by the browsers step by step („A brief history of CSS until 2016“, o. J.).

## 4 Basics of HTML

To understand the Syntax of CSS, a basic knowledge of ML is needed. Therefore, on the example of HTML Syntax MLs are explained.

HTML is structured by elements, these elements consist of an opening tag, possible content, and the closing tag. Specific elements, empty elements, cannot have a content and consequently do not need a closing tag .

```
<tagname>Content</tagname>
```

HTML elements can be nested to group and structure the document. The basic structure of an HTML document is shown in the example code.

```
<!DOCTYPE html>
<html>
<head>
  <title>Document</title>
</head>
<body>
  <section>
    Content
  </section>
</body>
</html>
```

The “<!DOCTYPE html>” tag declares the document as a HTML document. Inside the “html” element the “head” and “body” elements are nested. In the “head” element meta data, like a page title, is placed, this content will not be displayed directly in the rendered document. The content of the “body” element will be rendered and contains the main information of the document. Elements that are nested in another element have a child-parent relationship, and all elements they or the parents are nested are ancestors. The parent is also an ancestor, but the parent’s parent is only an ancestor. If multiple elements have the same parent element they are siblings, if they are directly after another they are also direct or next siblings.

In the opening tag additional attributes, with or without a value, can be assigned to the element. There are predefined attributes, but custom attributes can be created. Important attributes for CSS use are the “id” and “class” attributes because they can be specifically targeted. The “id” attribute is for identification of



one unique element, for this reason the value assigned to this attribute also must be unique. The same "class" value can be assigned to multiple elements.

## 5 Syntax

The purpose of CSS is to define how elements of a document appear. To achieve this, the browser must know what characteristic of a specific element must look like. („Cascading Style Sheets, designing for the Web – Chapter 2: CSS“, o. J.)

CSS is used to style documents written in a ML, to achieve this there are CSS rulesets, or simply rules, and at-rules. Generally, in CSS whitespaces and line breaks are ignored and can be used to format the code into a readable manner. But there are exceptions, like keywords are not allowed to be separated. Also most parts of the rules are case-insensitive („CSS Syntax Module Level 3“, o. J.).

Rules have a group of selectors and a declarative block, in which properties are assigned values. The group of selectors can contain one or more selectors, separated by commas. After the selectors the declarations are wrapped in curly brackets, these delimit the start and end of the declaration block. („CSS Syntax Module Level 3“, o. J.)

selector group    declaration block  
**selector { property: value; }**

Figure 1: Syntax of a CSS rule

A CSS declaration consists of a property and a value, they are separated by a colon and the declaration is closed by a semicolon. In one declaration block multiple declarations can be stated. For specific properties a set of values is allowed, if an invalid value is given the declaration is ignored by engines. White spaces and line breaks are ignored, and declarations are case-insensitive. The last declaration in a block must not end with a semicolon, but it is recommended. („Attribute Selectors - CSS“, 2023; „CSS Syntax Module Level 3“, o. J.; „CSS WG Blog – Case-Insensitivity in CSS“, o. J.)

CSS can be directly written in the HTML element, using the "style" attribute in the elements tag. The declarations are directly assigned as the value of the attribute,

no curly brackets or selectors are required, as it is already linked to the element. („CSS Syntax Module Level 3“, o. J.)

The “!important” flag, consisting of the “!” delimiter and the “important” keyword, is placed between the value and the semicolon of the declaration. This marks the declaration as important and stopping normal, not important, declaration from overwriting it. The effect on the cascade is described in the chapter 5.1 Cascading. („!Important - CSS“, 2023)

Beside rulesets another kind of statements are at-rules, which start with an at-sign. After the at-sign the identifier defines the used at-rule, that can have different syntax, semantic and transmit various information. The at-rule continuous till it ends with a semicolon or the end of a block.

```
@identifier (RULE);
```

The syntax of the content of at-rule can be different for each identifier, it can be a statement, a declaration block, or a rule. Using this nesting of rules conditional styling can be applied. For example, the syntax of the “@media” rules, to show how at-rules can be constructed, as it is also content of this thesis in a later stage („CSS Syntax Module Level 3“, o. J.).

```
@media <media-query-list> {<stylesheet>}
```

Starting with the “@” sign and the keyword “media” followed by the media query list. This can consist of specific media types and media feature expressions, which can be brought in context with each other by logical operators. With the Module “Media Queries Level 4” media queries made the expression syntax simpler, by allowing comparison signs on range features. The media query is followed by the block of conditional rules. The rules will be applied if the media query fits the device and media features the document is opened („CSS Syntax Module Level 3“, o. J.).

This “@media” rule applies to devices that have a screen as a media type and the media feature of a width smaller than 500 pixels. If this is true for the user agent, then all paragraphs are colored red, as pictured in Figure 2.

```
@media screen and (width < 500px) {  
  p {  
    color: red;  
  }  
}
```

```
}
```

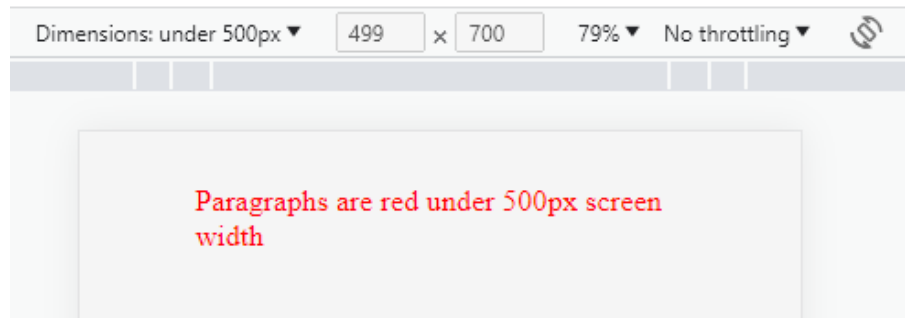


Figure 2: Result of the media query example

To use comments in the CSS code, start with slash and asterisk characters following the content of the comment and ending with asterisk and slash characters. They can be single or multi line. Comments can be used to add information exclude parts of the code from being rendered by the user agent („Comments - CSS“, 2023).

```
/* Comment content */
```

## 5.1 Selector group

Selectors are used to apply a declaration block to a specific element or elements. They prepended the declaration block. Different characteristics of elements can be used to target it, resulting in multiple selector kinds. Operators can be used to combine selectors. The selectors have different specificity, more in 6.2 Specificity.

### 5.1.1 Selector types

The universal selector, an asterisk character, selects all elements. This can be used to reset default values of specific properties for the whole document.

```
* { <declaration> }
```

The Type selector selects elements by their tag name, to use this selector the tag name is used with nothing prepended. In the example code the “<span>” elements are selected

```
span { <declaration> }
```

Elements can be assigned with a unique ID in the ML, with the ID selector a single element is selected. A hash character followed by the ID of the element, selects the element with this specific ID.

```
#id { <declaration> }
```

Class attributes can be assigned to elements in the ML, does not need to be unique. To select a class the class name is prepended with a period character.

```
.class { <declaration> }
```

Elements can be selected by other attributes assigned to them as well. Attribute Selectors are written in square brackets. It is possible to select elements that have the attributes and with the use of operators to select elements with a specific value of the attribute.

```
[attribute="value"] { <declaration> }
```

Pseudo classes allow selections based on a state of elements or other information, that are not directly or hardly selectable through combination of other selectors. There are different groups of pseudo classes depending on what aspect is selected with it. The pseudo class starts with a colon followed by an identifier for the proposed pseudo classes. Functional pseudo classes have brackets after the class name for the arguments. The element that a pseudo class is connected to is the "anchor element".

This rule is applied to an element with the class "a", if the hover state is active, this is the case when the cursor is over the element, depicted in Figure 3. User action pseudo classes change the appearance depending on the user interacting with the document.

```
.a:hover { color: red; }
```

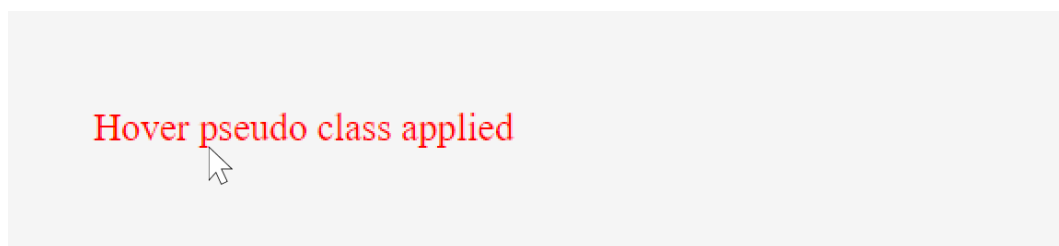


Figure 3: Example of an active hover state

Pseudo classes allows to select elements in the document that would not be able to be selected with other selectors and combinators, like selecting the first element among its siblings, as in the code example below. Utilizing tree-structural pseudo classes („Selectors Level 4“, o. J.).

```
.b:first-child { color: red; }  
  
<div>  
  <p class="b">First-child pseudo class applied</p>  
  <p class="b">This is the second child</p>  
</div>
```

First-child pseudo class applied

This is the second child

Figure 4: Output of the ":first-child" pseudo class example

Pseudo elements, similar to pseudo classes, are not directly in the document. They are abstracted parts of the originating element they are linked to. To style partitions of an element or add additional content, without altering the document tree. Pseudo-elements, denoted by double colons, provide a way to select and style certain parts of an element, such as the first letter or line, or even generate content itself. By utilizing pseudo-elements like ::before and ::after. Some pseudo elements can be used with a single colon in front, as it was standard before introduction of pseudo classes. This selector applies to the first letter of a element with the "c" class and enables styling of it separate from the rest of the element.

```
.c::first-letter { color: red; }
```

First-letter pseudo element applied

Figure 5: Output of the "::first-letter" pseudo element example

### 5.1.2 Grouping and compounding.

When a declaration block shall be applied to multiple selected elements, the selectors can be grouped into a selector list by using commas to separate them. If one of the selectors in the selector list is invalid, then the whole list would be invalid. With separate rules only the rule with the invalid selector would be invalid („Selectors Level 4“, o. J.).

Selecting elements, or a single element, based on multiple conditions, the selectors can be chained together without whitespaces, building a compound selector. Where the elements must match each selector to match the compound selector. When a type selector or universal selector is used it must be first in the sequence and there can only be one of them. Whitespaces are not allowed as they are used for the descendent combinator („Selectors Level 4“, o. J.).

### 5.1.3 Combinators

Combinators can be used to select more specific elements in reference to other elements in the document.

The descendant combinator selects the elements if any ancestor is a specific element. ancestor element is combined with the target selector by a space character („Selectors Level 4“, o. J.). In the following code example, the selector group matches both paragraphs.

```
.ancestor p { color: red; }  
  
<div class="ancestor">  
  <p>Descendant combined selector applied</p>  
  <div>  
    <p>Descendant combined selector also applied</p>  
  </div>  
</div>
```

Descendant combined selector applied

Descendant combined selector also applied

Figure 6: Output of the descendant combinat example

The child combinator selects the elements if the parent, direct ancestor, is a specific element. Parent element is combined with the target selector by a greater-than character („Selectors Level 4“, o. J.). Using the child combinator the selector group only matches the first paragraph as it is a direct ancestor.

```
.parent > p { color: red; }  
<div class="parent">  
  <p>Child combined selector applied</p>  
  <div>  
    <p>Child combined selector NOT applied</p>  
  </div>  
</div>
```

Child combined selector applied

Child combined selector NOT applied

Figure 7: Output of the child combinator example

To select a sibling element, meaning both elements have the same parent element, the sibling element is combined with the target element by a tilde character, the subsequent sibling combinator („Selectors Level 4“, o. J.). The selector group matches both paragraphs in the code example.

```
.sibling ~ p { color: red; }  
<p class="sibling">This is the starting sibling element</p>  
<p>Sibling combined selector applied</p>  
<p>Sibling combined selector applied</p>
```

This is the starting sibling element

Sibling combined selector applied

Sibling combined selector applied

Figure 8: Output of the sibling combinator example

If the target element shall be the immediate sibling of another element, the next sibling combinator, a plus character, is used to connect them and match the element („Selectors Level 4“, o. J.). In this example, only the second paragraph matches the selector, as its directly after the element with the “next-siblings” element.

```
.next-sibling + p { color: red; }  
  
<p class="next-sibling">This is the starting sibling element</p>  
<p>Next sibling combined selector applied</p>  
<p>Next sibling combined selector NOT applied</p>
```

This is the starting sibling element

Next sibling combined selector applied

Next sibling combined selector NOT applied

Figure 9: Output of the next siblings combinator example

Utilizing all the selectors, compounding, and combinators every element in the document and more can be selected to be styled. But it must be paid attention to the specificity and cascading, further described in chapter 5 “Concepts”.

## 5.2 Properties

CSS Properties are the characteristics of an element that define its behavior or style. As of May 2023 the W3C lists 574 properties from proposed to stable properties („Index of CSS properties“, o. J.). Important to note that some user agents do not support all the properties and for maximum accessibility, fallbacks must be used in these cases („W3C Process Document“, o. J.).

## 5.3 Values

CSS Values are the specific measurements, colors or settings assigned to properties, thereby defining how the element is styled and behaves („CSS Values and Units - CSS“, 2023).



The property defines what values are allowed, if incorrect values are inserted the declaration is deemed invalid and will be ignored. Some properties have special keywords or its own syntax that must be used as the value („CSS Values and Units - CSS“, 2023).

Properties that utilize keywords have a fix list of valid ones defined in the recommendations. For example, properties for layouts use keywords as a value, that describes the assigned behavior („CSS Values and Units - CSS“, 2023).

A URL can be assigned as a value inside the parentheses of the “url()” notation („CSS Values and Units - CSS“, 2023).

If the property is used to set a color, CSS has own color keywords. Also, the color spaces hexadecimal RGB, RGB and HSL can be used. These color spaces can be appended with the opacity making them the color spaces hexadecimal RGB with transparency, RGBA and HLSA („CSS Values and Units - CSS“, 2023).

To define sizes, numbers can be combined with multiple units in CSS. Absolute units, like pixel or millimeters, should be used with caution, as they are not responsive. Relative units specify the size in context to another element or characteristic. („CSS: em, px, pt, cm, in...“, o. J.).

Numbers, with decimal, or integers, without decimals, are available for specific properties. Some Properties can also be assigned percentages that represent a fraction of a base value („CSS Values and Units - CSS“, 2023).

## 5.4 Units

To give a dimension as a value, a number is appended with a unit. No whitespace or other character is allowed between the number and unit („CSS Syntax Module Level 3“, o. J.).

Length units are used to set the size of an element or of a part of the element. Relative length units are in relation to something else, for example the unit “rem” is relative to the font size of the root element. Another example is the viewport width and height, they are 1 percent of the user agent’s size. Using relative units to size the elements of a document, makes it accessible for a multitude of devices, without setting specific rules for each device („CSS Values and Units - CSS“, 2023).

Absolute units, such as pixels (px), inches (in), and millimeters (mm), allow to define element sizes, positions, and other dimensions with a high level of accuracy. Unlike relative units that depend on the context or parent elements, absolute units maintain a consistent size regardless of the device or screen resolution. This attribute makes them particularly useful for creating designs that require precise positioning or alignment. However, it is important to note that the use of absolute units can present challenges when designing for different screen sizes or accommodating users with varying accessibility needs („CSS Values and Units - CSS“, 2023).

To define other dimensions there are also units for angle, time, frequency, flex, and resolution („CSS Values and Units - CSS“, 2023).

## 5.5 Value functions

Value functions provide a way of using calculation and data processing to return a value for a declaration. The syntax for a value function starts with the function name followed by parenthesis, in which the argument or arguments are written. Depending on the function arguments must be provided. As of May 2023, there are 12 categories of value functions, like math, transformation, or filter functions („CSS Value Functions - CSS“, 2023).

```
.value-function { font-size: max(20px, 1.5rem); }
```

In the example the “max()” value function is used to get the maximum value of the comma separated list to get a minimum font size of 20 pixel.

## 5.6 Custom properties

Custom properties, also referred to variables, can be used to store a value and insert the value in a declaration. They can only be used for the values of declarations („CSS Custom Properties for Cascading Variables Module Level 1“, o. J.).

A value can be assigned to a custom property by prepending the name of the with two dashes, name is case-sensitive. Then like a normal property, after a colon the value follows this the declaration is closed by a semicolon. As a value everything can be assigned besides some characters like the semicolon. custom properties also must be assigned inside a ruleset, and it follows the same cascading,

specificity, and inherence concepts as normal rulesets, see [chapter "5 Concepts"](#). To make a custom property available in the whole document it can be assigned in the `":root"` pseudo class, which would be the `"html"` element in a HTML document („CSS Custom Properties for Cascading Variables Module Level 1“, o. J.).

```
:root{ --example-color: #ff0000 }
```

To insert the value of a custom property as a value for a CSS property the `"var()"` value function is utilized. As an argument for the value function the custom property name including the double dashes is inserted. A second argument, separated by a comma a fallback can be included („CSS Custom Properties for Cascading Variables Module Level 1“, o. J.).

```
.variable-color { color: var(--example-color); }
```

## 6 Concepts

### 6.1 Cascading

Before CSS there was a dispute on who should determine the design of the web. Should the user agent, the author, or the user style the web. This is where the Concept of cascading steps in. The user agent has a default style sheet, which can be overruled by the styles a user wants to apply. The rules of the author are overruling both other origins. This would give the author full control over the design and completely neglect any choice of the user and user agent. Therefore, the order is reversed for statements with the `"!important"` flag, giving the user a say in how the document is ultimately displayed („CSS Cascading and Inheritance Level 4“, o. J.).

### 6.2 Specificity

The selectors used also have another influence than just selecting an element, they are ranked in specificity and define which ruleset is applied. The most specific style is the style attribute, inline styles, it is always applied if it exists. After this the Specificity is in descending order the ID selector, class selector and the element selector. If multiple rules with the same specificity are in the same origin, then the last ruleset is used („Specificity - CSS“, 2023).

### 6.3 Inherence

To make the styling of documents more structured some values are inherent from the ancestor elements. For example, if the font of the whole document has to be changed, it's not needed to select all elements with text, but the body element can be selected and all elements with text will inherit the font. This inherence of styling does not apply to all properties to mitigate unwanted behavior („CSS Cascading and Inheritance Level 4“, o. J.).

### 6.4 Box Models

The Box model defines how elements are sized and occupy space. Every element is displayed as a rectangle, consisting of four layers: content, padding, border, and margin. In the middle is the content of the element, like text or images. The padding is the space between the content and the border of the element. The space around the border, separating the element from other elements is the margin. The three components around the content can each be manipulated by different properties and all sides can be set independently („CSS Box Model Module Level 3“, o. J.).

With the box-sizing property we can change how the height and width of the element is calculated. The standard is the content box but there is also the padding box, border box and margin box. The difference between them is what is contained inside the defined size. For example, the width of the border box contains the content, padding and border but not the margin. This makes it possible to size the elements correctly without the need to calculate the other components of the box model („CSS Box Model Module Level 3“, o. J.).

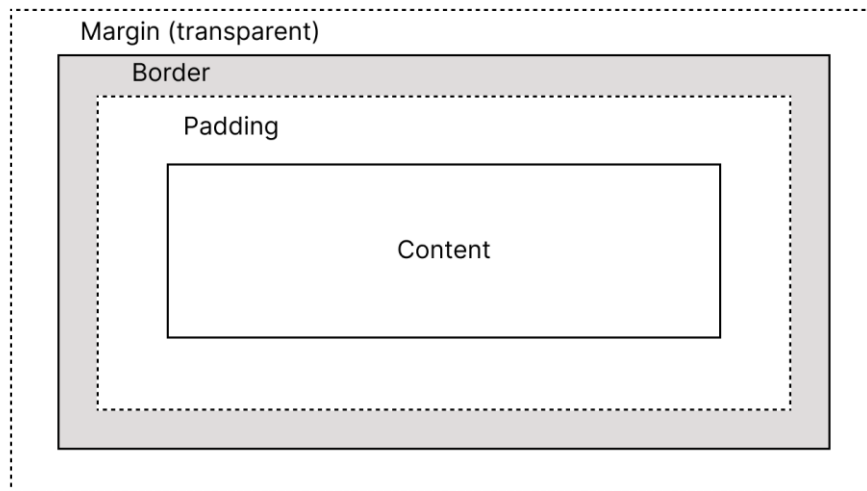


Figure 10: CSS Box Model

## 6.5 Layout methods

To arrange the elements of the document, in the desired way, there are different layout options. In the past the layout was a big part of the ML, where order and grouping were defined. With the newer CSS modules these options are now available in CSS and shifting more appearance decisions away from the content. These old layout options, like float and HTML tables, are now legacy method to style the appearance of a document („CSS Layout - Learn Web Development | MDN“, 2023; „Introduction to CSS Layout - Learn Web Development | MDN“, 2023).

For western languages where the writing flow is left to right then top to bottom, the block directions is vertical, the inline direction is horizontal. When languages with a vertical writing mode is displayed, like Chinese, then the directions are reversed. The normal flow is the default behavior of the elements, lining them up one after another, in the block direction. To make them line up in the inline direction the elements can be assigned the "inline" value to the "display" property („Block and Inline Layout in Normal Flow - CSS“, 2023).

With the CSS modules for Flex Layout Box and Grid, the display property to change to the layout method is now assigned to the parent element rather than each element. Meaning the element itself is not defining how it is displayed, which made

these systems possible and easily useable („Introduction to CSS Layout - Learn Web Development | MDN“, 2023).

Flex layout box is a powerful and flexible CSS layout module designed for one-dimensional layouts, either horizontally or vertically. It allows easy alignment, distribution, and reordering of elements within a container. Flex layout box provides properties to control the positioning, spacing, and alignment of elements across the main and cross axis. It is well-suited for building navigation bars, card layouts, and flexible content containers and simplifies the creation of responsive layouts („Basic Concepts of Flexbox - CSS“, 2023; „CSS Flexible Box Layout Module Level 1“, o. J.). In the code example the outer element is assigned a “display”-value of “flex” to initialize the flex layout box. The default direction is horizontal, named “row”. With the “justify-content” the child elements, the flex items, are positioned on the start of the main axis. The “align-items” property places the items in the center of the cross axis. This is displayed in the Figure 11.

```
.flex-container {  
  display: flex;  
  justify-content: start;  
  align-items: center;  
}  
  
<div class="flex-container">  
  <div class="flex-item"></div>  
  <div class="flex-item"></div>  
</div>
```

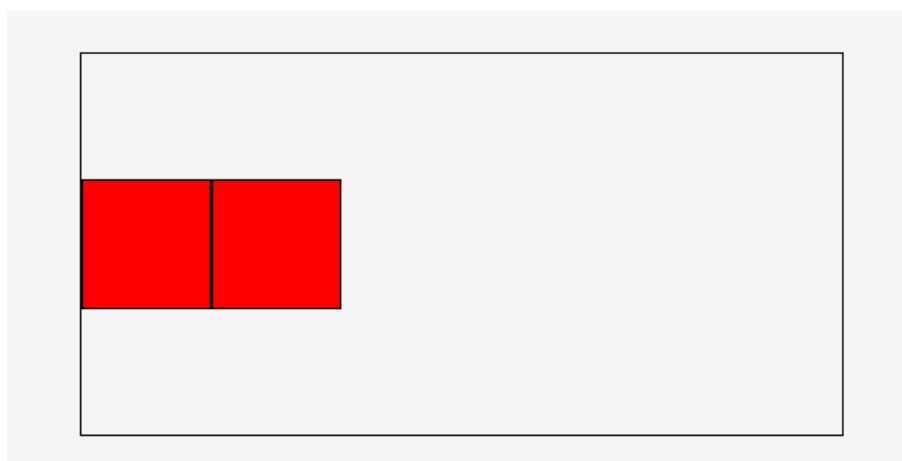


Figure 11: Output of the flexible box layout example

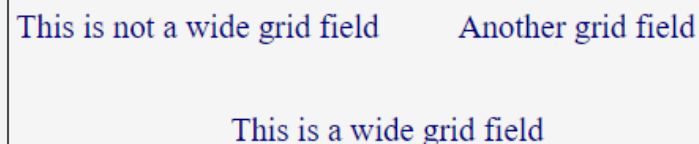
The CSS grid is still only a candidate recommendation of the W3C, but already supported by the major user agents. It divides the element in rows and columns to create the grid structure where the children can be positioned. The grid can be

defined by counts of columns or rows, and also by assigning specific grid areas. How the children are aligned in the grid field can be defined for the whole grid and specific for each child element. The child elements can span over multiple row and columns, make the CSS grid a versatile layout system with predictable behavior and responsiveness. Further separating the appearance of the document from the content, by removing the need of tags for additional grouping of elements („CSS Grid Layout - CSS“, 2023; „CSS Grid Layout Module Level 2“, o. J.). In the following example a grid with 2 columns is defined and the content of the grid fields is centered. The grid items are automatically placed in the grid fields. The last grid item is set to span over to grid columns. Figure 12 shows how the grid is rendered, in Figure 13 the grid lines are visible.

```
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr;
  justify-items: center;
  align-items: center;

.grid-item-wide {
  grid-column: span 2;
}

<div class="grid-container">
  <p>This is not a wide grid field</p>
  <p>Another grid field</p>
  <p class="grid-item-wide">This is a wide grid field</p>
</div>
```



This is not a wide grid field      Another grid field

This is a wide grid field

Figure 12: Output of the grid example

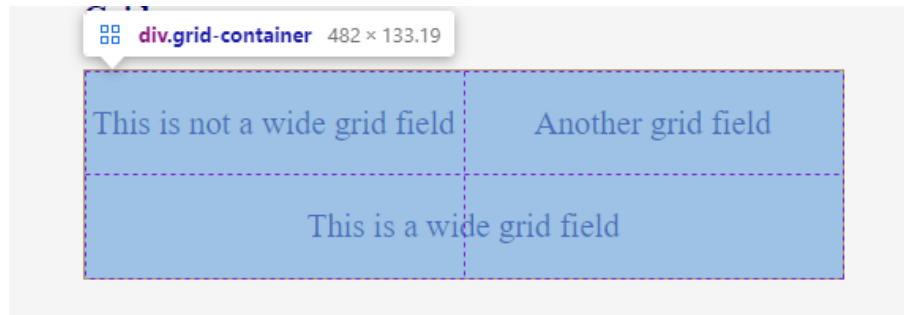


Figure 13: Output of the grid example with grid lines

With these two modules and the normal flow of the documents, complex illustrations of documents can be achieved, leaving out complex document structures. They are not a substitute for each other and should be used in conjunction with each other, as each method has its own advantages, disadvantages, and applications.

## 6.6 Media Queries

In CSS level 2 with media types, it was possible to define different styles to output media types. CSS level 3 extended this concept by also filtering the different characteristics and capabilities of the devices, by introducing the media queries. With them it's possible to adapt the style to the individual screen sizes, resolutions, and other. Making it a popular technique for responsive designs. This approach eliminates the need for separate websites or stylesheets for different devices and enables a single codebase to cater to diverse devices („Media Queries - CSS“, 2023; „Media Queries Level 5“, o. J.).

As described in the chapter “5 Syntax”, the media query starts with the “@” sign follow by the identifier “media”. Then a media type can be selected and a media feature expressions can follow. Logical operator can also be used to build the desired query („Media Queries Level 5“, o. J.).

The media query is only true if the media type and the media expressions are all matching the device. Only then the rulesets in the media query will be applied to the document.

Media queries also enable the concept of breakpoints, which are specific screen sizes at which the layout or styles change. By defining breakpoints and adjusting the design accordingly, the document can smoothly adapt to different screen sizes.



## 6.7 Accessibility

*The Web is fundamentally designed to work for all people, whatever their hardware, software, language, location, or ability. When the Web meets this goal, it is accessible to people with a diverse range of hearing, movement, sight, and cognitive ability.*  
(„Accessibility - W3C“, o. J.)

Accessibility means the making the document usable for all users, with all kinds of limitations. ML are semantic and therefore a good basis for accessibility and CSS can assist, making them even more inclusive.

In the document we can define alternative content for images, videos, and audio. For images an alternative text can be defined if it's not solely for decorations purposes. This gives visually impaired who rely on a screen reader and people with devices without the possibility of displaying images, to still gain the full information of the document. Subtitling videos makes it accessible for users not able to hear the audio, visitors with hearing problems or others not being able to play the audio. Both videos and audio should also be accompanied by a transcription („Accessibility - W3C“, o. J.). Modern search engines take the accessibility of websites in account when ranking them, nudging developers to build inclusive websites (Moreno & Martinez, 2013).

Besides making the content itself accessible there are also style considerations to make. The colors of the text and the background must have a significant contrast between each other, making the text easy to read for all users. The standard luminosity contrast ration between foreground and background should be at minimum 4.5:1 („Understanding Success Criterion 1.4.3: Contrast (Minimum) | WAI | W3C“, o. J.).

Using relative units to define the font size has the advantage over absolute units that the user settings can be respected, by referencing to the user's preferences. To set the font size percentages can be used as they reference to the parent. The unit "em" is the size of the font, inherited by the parent element. Both can get complicated to use if they are nested with lots of changes in font sizes. The "rem" unit solves this problem, it is relative to the root font size. If a user changes his settings to a bigger font size to make reading easier, then the document, using relative font sizes adapts to this preference. Another characteristic making reading

of a text more pleasant besides font size is the width of the text. A paragraph wider than around 60 to 70 characters makes it harder for the reader to skip to the next line. For good accessibility the text should be comfortable to read and adapt user preferences („Accessibility - W3C“, o. J.).

For users that are not able to use the mouse and must rely on the navigation via keyboard, elements that are focused must be marked to be easily identified („Accessibility - W3C“, o. J.).

Some users may prefer the website without many animations because it could make them feel uncomfortable or trigger other unwanted reactions. Media queries can be used with the media feature “prefers-reduced-motion” to apply styles to the document that does not utilize animations, that are not necessary („Accessibility - W3C“, o. J.).

Beside accessibility for users with disabilities, accessibility concerns users with different devices. This was already attended in the section “6.6 Media Queries”.

Implementation of these CSS techniques significantly contribute to improving the accessibility of documents, ensuring that they are usable and inclusive for all users. Adhering to accessibility guidelines and standards such as the Web Content Accessibility Guidelines (WCAG) is essential to provide an inclusive user experience.

## 7 Architecture

CSS Architecture refers to the organization, structure, and integration of CSS code and stylesheets. A well-designed CSS architecture promotes maintainability, scalability, and code reusability, making it easier to manage and update styles across coherent documents.

### 7.1 Implementation in HTML

In HTML there are three ways to implement CSS:

- Inline style
- Internal CSS
- External CSS

The inline style is directly written in the HTML element as a "style" attribute and the declarations as its value. As it is inside the element there is no need for a selector, making the inline styles also the most specific styling. It is not possible to use @-rules and therefore also no media queries with inline styles. This method of using CSS is bad practice because it neglects a basic idea of CSS, separation between content and styling. Code may be repeated often, making maintenance and changes of the styling more time consuming. Sometimes it has to be used when changing the styling of out of the box solutions, where the CSS file is not editable („How CSS Is Structured - Learn Web Development | MDN", 2023).

For internal CSS, or embedded CSS, the CSS code is in the HTML document but not at each element, like with inline style. Between the "<head>" tags of the HTML document a "<style>" element contains the CSS code, with full possible use of rulesets and @-rules. However, if multiple documents utilize the same styling the CSS will repeat again, making this only a suitable option if it's a single page with this styling. Else similar problems with maintaining as with inline styles emerge. Again, if the CSS file cannot be edited, this may be a valid option („How CSS Is Structured - Learn Web Development | MDN", 2023)..

External CSS contains all the CSS code in a different file, a ".css" file. It is possible to link one CSS file to multiple HTML document, but also linking multiple CSS files to HTML documents. With a "<link>" element in the "<head>" element we can reference the CSS. The relation attribute "rel" specifies that a stylesheet is linked, the hypertext reference attribute "href" provides the path to the stylesheet.

This implementation of CSS in HTML is normally the most valid option if no other limitations apply. This makes it possible to reduce repetitive code and simplify the changes and maintenance of the styling. While still being able to split specific code for single documents. („How CSS Is Structured - Learn Web Development | MDN“, 2023)

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

## 7.2 Implementation in XML

CSS can be implemented in XML, but its own style sheet language with XSL, that has advantages over CSS for XML. While HTML can utilize inline styles, most user agents do not interpret them with XML. Internal style sheets are also not supported with XML, there is a workaround to use them. But it can have problems and there is no specification. The usage of external style sheets is possible with “xml-stylesheet” processing instruction, which has to be the first tag in the document. As with HTML there can be multiple style sheets referenced.(„How to add style to XML“, o. J.)

```
<?xml-stylesheet href="my-style.css"?>
```

## 7.3 Best Practices

CSS was evolved, there did also evolve some best practices, some of them will be explained in this section.

CSS ignores all whitespaces and line breaks, therefore these can be used to bring the code in a readable format. It is possible to put the whole rule in one line, or to break it up into multiple lines („Organizing Your CSS - Learn Web Development | MDN“, 2023).

To maintain an overview of the code, it can be structured in logical section. This structure could start with the general styles and getting more specific to single elements or parts of the document, the later in the CSS file lines („Organizing Your CSS - Learn Web Development | MDN“, 2023).

Comments can help to mark these sections for easier navigation in the code. Also adding comments to some rules is helpful if they are not self-explanatory or

referencing the source of the code if it is not self-written lines („Organizing Your CSS - Learn Web Development | MDN“, 2023).

The used selector only should be as specific as necessary, this way they can be used in multiple cases while only applying to the specific elements. Some times rules have to be overruled which is easier if the first rule is not to specific lines („Organizing Your CSS - Learn Web Development | MDN“, 2023).

Beside the possibility of marking logical sections in the CSS code, it also can be useful to split the code in multiple CSS files. When there are specific stylings that only apply to one page or set of subpages, then this file can be, additional to the general CSS file, be imported in these documents. Making it not only more organized but avoid the download of unnecessary code lines („Organizing Your CSS - Learn Web Development | MDN“, 2023).

Custom properties, CSS variables, can be useful if for example colors are often reused. Then the variable can be assigned one time and be used as the value. Custom properties promote consistency, reusability, and maintainability in CSS codebases. Additionally reducing the amount of code duplication and improves the efficiency. They can be modified using JavaScript, promoting dynamicity and interactivity lines („Organizing Your CSS - Learn Web Development | MDN“, 2023).

Utilizing these practices needs consistency, to bring a positive effect. When naming classes, it should also follow a consistent schema („Organizing Your CSS - Learn Web Development | MDN“, 2023).

#### **7.4 CSS methodologies**

To mitigate the need to develop an own system when writing CSS, one of the many existing methodologies can be selected and still be adapted to personal needs if needed. These methodologies range from big classes for components to classes with only one declaration. As always there is no right way to use CSS, only personal, team, and project bases preferences. These methodologies have the advantage that many developers know them, therefore code written with them is accessible and understandable („Organizing Your CSS - Learn Web Development | MDN“, 2023).

One of the commonly used methodologies is the Object-Oriented CSS. The concept of OOCSS is the creation of classes that can be applied to different objects, which share styles, to combine them to the desired style. It is based on two principles, first to separate structure and skin, meaning to create own classes for appearance and ones for the structure, not using the type selector. The second principle being separate containers and content, not using combined selectors by relation, making the classes used for them more versatile. OOCSS reduces code duplication, thereby file size, and increases flexibility for style application („Organizing Your CSS - Learn Web Development | MDN“, 2023; Sullivan, o. J.).

SMACSS, short for Scalable and Modular Architecture for CSS, provides a modular approach to structuring and organizing CSS for enhanced scalability and maintainability. Styles are categorized into five categories. The base category is for styling general properties, like typography. In the layout the basic structure of the page is defined. Modules are reusable components with specific styling. To handle dynamic changes and variations of elements, such as the hover state, the state stylings are used. for different themes the corresponding theme category exists. It based on single responsibility, styles are focused on single modules. Using these categories, separates the modules from the basic design , thereby making it more maintainable and extendable. SMACSS provides a clear structure and naming convention making the code consistent („Organizing Your CSS - Learn Web Development | MDN“, 2023; „Scalable and Modular Architecture for CSS“, o. J.).

The Block-Element-Modifier, BEM, naming convention is popular for writing maintainable code and standardizing naming classes, to enhance reusability and structure. BEM breaks down the design into modular components and dividing each into three parts. A standalone component that has its own functionality and styling is called the block, they are named using lowercased words. Elements are the inter part of the block and cannot exist without being tied to a block, they are connected to the block using double underscores. For defining different states or variation of block or elements the modifier is used, represented by double dashes. This can lead to longer class names, but the advantages are that the class names are self-explanatory, readable, and understandable. Making

collaboration in teams simpler and consistent („Organizing Your CSS - Learn Web Development | MDN“, 2023; Strukchinsky, o. J.).

In contrast is the methodology of atomic CSS, which promotes granular writing style. Making small and distinctive utility classes instead of longer component-based classes. Its target is to make styling of individual elements more flexible and reduce the file size while maximizing the code reusage. The styles are broken down into atomic classes, containing only one property. The class names represent the property value pair they contain, making them intuitive. By combining different utility classes with another complex layouts and design can be styled without the need of writing specific classes for components. Improving performance, through reduced load time and file size, and flexibility. But it leads to a higher number of classes on each element, making the HTML more complex and verbose („Organizing Your CSS - Learn Web Development | MDN“, 2023).

## **7.5 Frameworks**

Utilizing the methodologies frameworks have emerged to simplify the writing of CSS code. CSS frameworks are predesigned conventions and classes that can be used to enhance styling efficiency. They provide a consistent style through the document and often even offer complete components to implement. Frameworks save setup time, because they often include completely responsive layout systems, providing optimal presentation on different devices. CSS frameworks like Bootstrap and Foundation have become widely popular and widely adopted in the web development community. These frameworks offer comprehensive documentation, support, and community resources. This extensive support network makes it easy to adopt these frameworks and provides many out-of-the-box solutions („Organizing Your CSS - Learn Web Development | MDN“, 2023).

## 8 Outlook

The evolution of CSS modules in the last 20 years enhanced the capabilities while still making it simpler to style documents. New challenges for CSS are emerging and the W3C is trying to recommend new modules to overcome them („All CSS specifications“, o. J.; „CSS current work & how to participate“, o. J.). There are several key areas developed can be anticipated.

Styling the layout of a document has vastly improved with the flex layout box and grid modules. The new multi column layout module is a stable proposed recommendation from this year. Further improvements of these layout methods can be expected, as they are all just level one modules, and the grid module is still a candidate recommendation. Development of new modules would also be possible since the system, of the parent element defining the position of the child, could be adopted for different layout methods than flex layout box and grid („CSS current work & how to participate“, o. J.).

The W3C is trying to enable more sophisticated and dynamic styling. Modern systems generating HTML documents dynamically and only partly refreshing the document, needs appropriate CSS properties for optimal styling of these components and states („All CSS specifications“, o. J.).

With a staggering number of different devices and therefore screen sizes, the need for responsiveness rises. The next module of media queries is already a working draft, making them even more versatile. And also further improving CSS for accessibility, to include as many people as possible, with additional media features to test for preferences („Media Queries Level 5“, o. J.). Other modules for improved accessibility are on the way to become recommendations. For example, the speech module, to define how speech synthesizer render specific text („CSS Speech Module Level 1“, o. J.).



## 9 Source Index

A brief history of CSS until 2016. (o. J.). Abgerufen 14. Juni 2023, von <https://www.w3.org/Style/CSS20/history.html>

Accessibility—W3C. (o. J.). Abgerufen 29. Mai 2023, von <https://www.w3.org/standards/webdesign/accessibility>

All CSS specifications. (o. J.). Abgerufen 31. Mai 2023, von <https://www.w3.org/Style/CSS/specs.en.html>

Attribute selectors - CSS: Cascading Style Sheets | MDN. (2023, April 22). Abgerufen 9. Mai 2023, von [https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute\\_selectors](https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute_selectors)

Basic concepts of flexbox - CSS: Cascading Style Sheets | MDN. (2023, Mai 24). Abgerufen 14. Juni 2023, von [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_flexible\\_box\\_layout/Basic\\_concepts\\_of\\_flexbox](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout/Basic_concepts_of_flexbox)

Block and inline layout in normal flow - CSS: Cascading Style Sheets | MDN. (2023, Mai 29). Abgerufen 14. Juni 2023, von [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_flow\\_layout/Block\\_and\\_inline\\_layout\\_in\\_normal\\_flow](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flow_layout/Block_and_inline_layout_in_normal_flow)

Cascading Style Sheets, designing for the Web – Chapter 2: CSS. (o. J.). Abgerufen 9. Mai 2023, von <https://www.w3.org/Style/LieBos2e/enter/>

Comments - CSS: Cascading Style Sheets | MDN. (2023, April 16). Abgerufen 14. Juni 2023, von <https://developer.mozilla.org/en-US/docs/Web/CSS/Comments>

CSS Box Model Module Level 3. (o. J.). Abgerufen 14. Juni 2023, von <https://www.w3.org/TR/css-box-3/>

CSS Cascading and Inheritance Level 4. (o. J.). Abgerufen 14. Juni 2023, von <https://drafts.csswg.org/css-cascade-4/#cascade>

CSS: Cascading Style Sheets | MDN. (2023, April 16). Abgerufen 14. Juni 2023, von <https://developer.mozilla.org/en-US/docs/Web/CSS>

CSS current work & how to participate. (o. J.). Abgerufen 31. Mai 2023, von <https://www.w3.org/Style/CSS/current-work.en.html>

CSS Custom Properties for Cascading Variables Module Level 1. (o. J.). Abgerufen 13. Juni 2023, von <https://www.w3.org/TR/css-variables-1/>

CSS: em, px, pt, cm, in.... (o. J.). Abgerufen 13. Juni 2023, von <https://www.w3.org/Style/Examples/007/units.de.html>

CSS Flexible Box Layout Module Level 1. (o. J.). Abgerufen 14. Juni 2023, von <https://www.w3.org/TR/css-flexbox-1/>

CSS Grid Layout - CSS: Cascading Style Sheets | MDN. (2023, Mai 29). Abgerufen 14. Juni 2023, von [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_grid\\_layout](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout)

CSS Grid Layout Module Level 2. (o. J.). Abgerufen 14. Juni 2023, von <https://drafts.csswg.org/css-grid/>

CSS layout—Learn web development | MDN. (2023, Februar 23). Abgerufen 14. Juni 2023, von [https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS\\_layout](https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout)

CSS Snapshot 2023. (o. J.). Abgerufen 14. Juni 2023, von <https://www.w3.org/TR/CSS/>

CSS Speech Module Level 1. (o. J.). Abgerufen 31. Mai 2023, von <https://www.w3.org/TR/css-speech-1/>

CSS Syntax Module Level 3. (o. J.). Abgerufen 9. Mai 2023, von <https://www.w3.org/TR/css-syntax-3/>

CSS value functions - CSS: Cascading Style Sheets | MDN. (2023, April 27). Abgerufen 13. Juni 2023, von [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Functions](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Functions)

CSS values and units - CSS: Cascading Style Sheets | MDN. (2023, März 30). Abgerufen 13. Juni 2023, von [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Values\\_and\\_Units](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Values_and_Units)

CSS WG Blog – Case-Insensitivity in CSS. (o. J.). Abgerufen 9. Mai 2023, von [https://www.w3.org/blog/CSS/2007/12/12/case\\_sensitivity/](https://www.w3.org/blog/CSS/2007/12/12/case_sensitivity/)

Extensible Markup Language (XML). (o. J.). Abgerufen 14. Juni 2023, von <https://www.w3.org/XML/>

How CSS is structured—Learn web development | MDN. (2023, März 2). Abgerufen 29. Mai 2023, von [https://developer.mozilla.org/en-US/docs/Learn/CSS/First\\_steps/How\\_CSS\\_is\\_structured](https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/How_CSS_is_structured)

How to add style to XML. (o. J.). Abgerufen 29. Mai 2023, von <https://www.w3.org/Style/styling-XML.en.html>

HTML: The Markup Language (an HTML language reference). (o. J.). Abgerufen 14. Juni 2023, von <https://www.w3.org/TR/2012/WD-html-markup-20121025/spec.html>

!important - CSS: Cascading Style Sheets | MDN. (2023, März 12). Abgerufen 11. Juni 2023, von <https://developer.mozilla.org/en-US/docs/Web/CSS/important>

Index of CSS properties. (o. J.). Abgerufen 10. Mai 2023, von <https://www.w3.org/Style/CSS/all-properties.en.html#list>

Introduction to CSS layout—Learn web development | MDN. (2023, Februar 23). Abgerufen 14. Juni 2023, von [https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS\\_layout/Introduction](https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Introduction)

Media queries - CSS: Cascading Style Sheets | MDN. (2023, Mai 25). Abgerufen 14. Juni 2023, von [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_media\\_queries](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_media_queries)

Media Queries Level 5. (o. J.). Abgerufen 31. Mai 2023, von <https://www.w3.org/TR/mediaqueries-5/#descdef-media-prefers-reduced-motion>

Moreno, L., & Martinez, P. (2013). Overlapping factors in search engine optimization and web accessibility. *Online Information Review*, 37(4), 564–580. <https://doi.org/10.1108/OIR-04-2012-0063>

Organizing your CSS - Learn web development | MDN. (2023, Mai 9). Abgerufen 31. Mai 2023, von [https://developer.mozilla.org/en-US/docs/Learn/CSS/Building\\_blocks/Organizing](https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Organizing)

Scalable and Modular Architecture for CSS. (o. J.). Abgerufen 31. Mai 2023, von <http://smacss.com/>

Selectors Level 4. (o. J.). Abgerufen 10. Mai 2023, von <https://drafts.csswg.org/selectors/>

Specificity - CSS: Cascading Style Sheets | MDN. (2023, Mai 17). Abgerufen 14. Juni 2023, von <https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity>

Strukchinsky, V. (o. J.). BEM — Block Element Modifier. Abgerufen 14. Juni 2023, von <http://getbem.com/>

Sullivan, N. (o. J.). Object Oriented CSS. Abgerufen 31. Mai 2023, von GitHub website: <https://github.com/stubbornella/oocss/wiki/Home>

Understanding Success Criterion 1.4.3: Contrast (Minimum) | WAI | W3C. (o. J.). Abgerufen 29. Mai 2023, von <https://www.w3.org/WAI/WCAG21/Understanding/contrast-minimum.html>

Usage Statistics of CSS for Websites, June 2023. (o. J.). Abgerufen 14. Juni 2023, von <https://w3techs.com/technologies/details/ce-css>

User Agent Accessibility Guidelines (UAAG) 2.0. (o. J.). Abgerufen 14. Juni 2023, von <https://www.w3.org/WAI/UA/2011/ED-UAAG20-20110525/#def-user-agent>

W3C Process Document. (o. J.). Abgerufen 14. Juni 2023, von <https://www.w3.org/2020/Process-20200915/#maturity-levels>

## 10 Appendix

The HTML code follows for all examples in this paper.

```
<!-- code.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <link rel="stylesheet" href="/">
  <style> /* Internal CSS initialised by the style-tag */

      /* ----- EXAMPLE CSS CODE -----*/

      /* Media query */

      @media screen and (width < 500px) {
        p {
          color: red;
        }
      }

      /* Selectors */

      /* -Universal selector */
      *{
        color: navy;
      }

      /* -Type selector */
      span {
        color: red;
      }

      /* -ID selector */
      #id {
        color: red;
      }

      /* -Class selector */
      .class {
        color: red;
      }

      /* -Attribute selector */
      [attribute] {
        color: red;
      }
      [attribute="value"] {
        color: red;
      }
    </style>
</head>
</html>
```

```
/* -Pseudo class */

.a:hover {
  color: red;
}

.b:first-child {
  color: red;
}

/* -Pseudo elements */

.c::first-letter {
  color: red;
}

/* -Grouping and Compounding */

.grouping1, .grouping2 {
  color: red;
}

p.compound {
  color: red;
}

/* -combinators */

.ancestor p {
  color: red;
}

.parent > p {
  color: red;
}

.sibling ~ p {
  color: red;
}

.next-sibling + p {
  color: red;
}

/* Value function */

.value-function {
  font-size: max(20px, 1.5rem);
}

/* Custom Property */

:root{
  --example-color: #ff0000
```

```
}
.variable-color {
  color: var(--example-color);
}

/* Layout methods */
/* Flexbox */

.flex-container {
  display: flex;
  justify-content: start;
  align-items: center;
}

/* Grid */

.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr;
  justify-items: center;
  align-items: center;
}
.grid-item-wide {
  grid-column: span 2;
}

</style>
</head>
<body>
  <section>

    <h1>Syntax</h1>
    <h2>at-rule</h2>
    <p>Paragraphs are red under 500px screen width </p>

    <h2>Selectors</h2>

    <p>Universal selector applied</p>

    <div>
      <span>Type selector applied</span>
    </div>
    <p id="id">ID selector applied</p>
    <p class="class">Class selector applied</p>
    <p attribute="value">Attribute selector applied</p>

    <p class="a">Hover pseudo class applied</p>

    <div>
      <p class="b">First-child pseudo class applied</p>
      <p class="b">This is the second child</p>
    </div>
  </section>
</body>
</html>
```

```
<p class="c">First-letter pseudo element applied</p>

<div>
  <p class="grouping1">"grouping1" class applied</p>
  <p class="grouping2">"grouping2" class applied</p>
</div>

<div>
  <p class="compound">"p"element with "compound" class</p>
  <p class="">"p"element, no "compound" class</p>
</div>

<div class="ancestor">
  <p>Descendant combined selector applied</p>
  <div>
    <p>Descendant combined selector also applied</p>
  </div>
</div>

<div class="parent">
  <p>Child combined selector applied</p>
  <div>
    <p>Child combined selector NOT applied</p>
  </div>
</div>

<div>
  <p class="sibling">This is the starting sibling element</p>
  <p>Sibling combined selector applied</p>
  <p>Sibling combined selector applied</p>
</div>

<div>
  <p class="next-sibling">This is the starting sibling
element</p>
  <p>Next sibling combined selector applied</p>
  <p>Next sibling combined selector NOT applied</p>
</div>

<h2>Value Function</h2>
<p class="value-function">This font size is the bigger option of
20px or 1.5rem</p>

<h2>Custom Properties</h2>
<p class="variable-color">The color of the custom property is
applied</p>

<h1>Concepts</h1>
<h2>Layout methods</h2>
<h3>Flexbox</h3>
<div class="flex-container">
  <div class="flex-item"></div>
  <div class="flex-item"></div>
</div>
```



```
<h3>Grid</h3>
<div class="grid-container">
  <p>This is not a wide grid field</p>
  <p>Another grid field</p>
  <p class="grid-item-wide">This is a wide grid field</p>
</div>

</section>
</body>
</html>
```

The CSS code for implementing a separate file and to display the examples in this paper correctly.

```
/* style.css */

/*: helper styles to display the examples correctly */
h1 {font-size: 2.6rem;}
h2 {font-size: 1.9rem;}
h3 {font-size: 1.3rem;}
p, span {font-size: 1.3rem;}
section > p, section > span, section > div, h2 {margin-bottom: 6rem;}
body {border: solid 1px black; padding: 3rem; max-width: 60ch; margin-
inline: auto; background-color: whitesmoke;}
.grid-container {
  width: 100%;
  border: 1px solid black;
  display: grid;
}
.flex-container {
  width: 100%;
  height: 15rem;
  border: 1px solid black;
}
.flex-item {
  width: 5rem;
  height: 5rem;
  background-color: red;
  border: 1px solid black;
}
```