Vienna University of Economics and Business

Seminar Paper

CSS: Concepts, Architecture, Nutshell Examples, Outlook

Author: Marina Wustinger

Matriculation Number: h11802099

Seminar aus BIS (Schiseminar) - 0082

Term: WS22/23

Submitted on 15th of December 2022

Instructor: ao.Univ.Prof. Mag. Dr. Rony G. Flatscher

Abstract

This seminar paper handles the styling language CSS, its history, key concepts, implementation and further provides examples of CSS in use. The stylesheet language offers an enormous amount of properties used to style a document. The newest versions even offer some features which can be used to make a web design responsive, assuring a flexible and adjustable style. After a short introduction into CSS and its beginnings, an overview of CSS concepts is given, followed by commonly used CSS architectures and the possibilities of implementing CSS into HTML. Next, some examples are used to illustrate CSS properties and features. Lastly, the paper is concluded with a brief summary as well as an outlook regarding the future of CSS. An understanding of HTML and its elements is needed to understand the examples.

After reading this paper the reader should have a basic knowledge of CSS and be familiar with the main concepts of it. Further they should be able to apply some common CSS properties to an HTML document.

Declaration of Authorship

I assure:

- to have individually written, to not have used any other sources or tools than referenced and to not have used any other unauthorized tools for the writing of this seminar paper.
- to never have submitted this seminar paper topic to an advisor neither in this, nor in any foreign country.
- that this seminar paper matches the seminar paper reviewed by the advisor.

Date: December 15th, 2022

Signature Marina Wustinger:

Table of Content

List of Tables

List of Figures

Codes

1 Introduction to CSS

CSS stands for Cascading Style Sheets, it is a “domain-specific, declarative programming language” (*The Evolution of CSS in 3 Decades*, n.d.), used in front-end web development to transform documents written in a markup language such as HTML into beautiful looking website by giving one the ability to control how the elements should be presented to the user in the bowser regarding their layout, style, etc. (*What Is CSS? - Learn Web Development | MDN*, 2022) Same as HTML, CSS is standardized by the World Wide Web Consortium – W3C. It as an open-source standard, independent and free to use. (Kumar, 2022)

The following chapters will present a short summary of the history of CSS, the key concepts, and lastly simple and more complex examples of CSS in use.

1.1 History

The history of CSS has its beginnings in 1994. The web was getting used more and more as an electronical publishing platform, but there was no way of styling the documents. So, Hakon Wium Lie invented a stylesheet language to solve the problem and Cascading Style Sheet – CSS – was born. (*A Brief History of CSS Until 2016*, n.d.)

The Arena web browser was the first browser that Lie tested CSS on. Later, in collaboration with Tim Berners-Lee and Robert Cailliau, they created CSS1 and CSS2. CSS soon became an official web standard, influencing look and accessibility of the world wide web. If not for CSS, the web would look very simple and assumably boring. In fact, the web would not be able to exist as known today. (*CSS History*, n.d.)

Without browsers, CSS would have simply been a sublime proposal. For this reason, browsers play a big role in the history of CSS, with the Microsoft Internet Explorer 3 being the first commercial browser to support CSS, followed by Netscape Navigator version 4.0 and later Opera. (*A Brief History of CSS Until 2016*, n.d.)

Over the years, W3C has been controlling and developing CSS features. They divided CSS into several profiles and levels. Profiles represent a subgroup of one or more levels. Now, the profiles listed are for mobile devices, television sets and printers. In 1996, CSS1 was released and after some corrections republished in 1999. CSS2 was

built on CSS1 and released in 1998, it supports various output media. (*CSS History*, n.d.)

The differences between CSS1, CSS2 and CSS3 can be seen in the features, added with each release. CSS1 first allowed the user to make changes to the colour, font style and size of the background and text. With CSS2, abilities to design the page layout were added. In 1999, CSS3 came with features, that allowed the user to choose from an even greater range of fonts and create presentations from documents. In addition, the user could now integrate rounded borders and multiple columns. W3C is continuously improving and adding new features to CSS, instead of releasing a completely new version. Hence, CSS4 is happening, however it will very likely never bear the name of CSS4. (*CSS History*, n.d.)

1.2 How CSS works in the Browser

In order for the browser to display the webpage, the document gets processed in a number of different stages.

1. The HTML document is loaded by the browser.
2. The HTML document is converted into a DOM – Document Object Model, which has a similar structure to a tree with each element, attribute, and pieces of text becoming a DOM node in the tree. Every DOM node can be defined by its relationship to the other nodes.
3. In this stage the browser will fetch most resources which are linked to by the HTML document, this can be embedded videos or images, and even other style sheets.
4. Now the browser analyses the previously fetched CSS and sorts it into different so called buckets in regard to their selector types. Based on the selectors, the browser will then apply the respective rules to the nodes in the DOM and attach the required styles to them. This step in this stage can also be called a render tree.
5. Next, the render tree will be laid out according to the structure it should appear in.
6. Lastly, the visual display of the webpage will be shown on the screen.
   (*How CSS Works - Learn Web Development* | *MDN*, 2022)

Should the browser encounter any CSS it does not understand, it simply moves on to the next bit of CSS, meaning it ignores anything it does not comprehend. This can happen when a user does not use the latest version of a certain browser, but also when

a browser does not support a new feature of CSS. (*How CSS Works - Learn Web Development* | *MDN*, 2022)

2 An Overview of CSS Concepts

The following chapters give a rough overview of the main CSS concepts, architectures, how CSS works in the browser and finally how CSS can make websites more accessible.

2.1 Syntax and Rules

“A CSS Syntax rule consists of a selector, property, and its value. The selector points to the HTML element where CSS style is to be applied. The CSS property is separated by semicolons. It is a combination of selector name followed by the property: value pair that is defined for the specific selector.” (GeeksforGeeks, 2022)

The Syntax, also referred to as rulesets or rules, therefore looks like the following: *selector { property: value;}*

There are over 100 different properties and an even larger number of values. Each property defines exactly which values are valid, in other words, what properties can be paired with which values. Should a value not be compatible with a certain property, the declaration will be deemed invalid, further leading to the CSS engine to completely ignore it. Properties and Values are case-sensitive and need to be separated using a colon *:* , while white spaces are usually ignored. (*Syntax - CSS: Cascading Style Sheets* | *MDN*, 2022)

This Syntax rule makes up the main building blocks of a stylesheet, but in order to use other information such as another external stylesheet, which needs to be imported or a character set, at-rules are needed. As the name already indicates, each statement starts with a @ directly followed by an identifier and a rule. (*Syntax - CSS: Cascading Style Sheets* | *MDN*, 2022)

The Syntax using at-rules could look like the following: *@import ‘somefile.css’;*

For a specific subset of at-rules, a specific group of statements is needed called nested statements. They only apply when a certain condition is met. Such a nested statement is for example *@media* which is used for targeting the styles to specific screens or simply when the document is printed. (*Syntax - CSS: Cascading Style Sheets* | *MDN*, 2022)

2.2 Selectors

CSS selectors select the HTML elements to which the styles are supposed to be applied to. There are five categories of selectors: basic selectors, combinators selectors, pseudo-class selectors, pseudo-element selectors and lastly attribute selectors. When selecting elements, more than one selector can be used. (*CSS Selectors - CSS: Cascading Style Sheets* | *MDN*, 2022)

2.2.1 Basic Selectors

The basic selectors include the most commonly used selectors, listed in Table 1.

Name	Description	Syntax
Type Selector	Uses the HTML element/tag names to select the HTML elements.	element/tag_name { property: value;}
Universal Selector	Illustrated by an asterisk *, selects everything in the HTML document, which is why it can be used to make global changes, such as making the overall font bold.	* {property: value;}
Class Selector	Starts with a dot . and is followed by the class to be selected. Everything in the HTML document with that class applied to will be selected and styled accordingly.	.class_name {property: value;}

ID Selector	Starts with a # and selects the element with the ID, following after the # . Can be combined with a type selector.	#id_name {property: value;} element/tag_name#id_name {property: value;}

Table 1 – Basic Selectors (CSS Selectors - CSS: Cascading Style Sheets | MDN, 2022)

2.2.2 Combinators

In order to explain the relationship between two selectors, CSS uses combinators listed in Table 2, which indicate the relationship using certain characters between the two selectors. (*CSS Selectors - CSS: Cascading Style Sheets* | *MDN*, 2022)

Name	Description	Syntax
Descendant Selector	Indicated by a single space character and using the second selector selects those elements that have an ancestor element which matches the first selector.	ancestor_element descendant element {property: value;}
Child Selector	Specifically targets those elements selected by the second selector that are a direct child to the elements selected by the first using > .	Parent_element > child_element {property: value;}
Adjacent Sibling Selector	Selects only those elements selected by the second selector that are directly preceded by the	preceding_element + adjacent_element {property: value;}

	elements selected by the first selector using + .	
General Sibling Selector	Selects sibling elements that are not immediately following, so any elements selected by the second selector coming anywhere after the elements selected by the first selector using ~ .	sibling_element ~ following_sibling_element {property: value;}

Table 2 – Combinators (CSS Selectors - CSS: Cascading Style Sheets | MDN, 2022)

2.2.3 Pseudo-classes and Pseudo-elements

Pseudo-classes act like a class that has been applied to an element of the document, in order to reduce the number of classes in the markup, which provides a more flexible and maintainable code. The syntax looks like the following: *element:pseudo-class-name* They are used to specify a certain state for a targeted element, which could be *:active*, *:checked*, but also *:first-child*, *:last-child*, etc. (*Pseudo-classes and Pseudo-elements - Learn Web Development* | *MDN*, 2022)

In order to style only a specific part of a selected element a keyword, a so-called pseudo-element is added to the selector. The syntax looks like the following: *element::pseudo-element-name* (*Pseudo-classes and Pseudo-elements - Learn Web Development* | *MDN*, 2022)

2.2.4 Attribute Selectors

Attribute Selectors are used when selecting an element with a certain attribute or attribute value. There are many different types of attribute selectors, some commonly used ones are listed in Table 3. (*CSS Selectors - CSS: Cascading Style Sheets* | *MDN*, 2022)

Name	Description	Syntax

[attr]	Selects every element with specified attribute.	element/tag_name[attribute] {property: value;}
[attr=value]	Selects elements with a specified attribute and value.	Element/tag_name[attribute=”value”] {property: value;}
[attr~=value]	Selects elements with an attribute value including a specified word.	[attribute~=”value”] {property: value;} Element/tag_name[attribute~=”value”] {property: value;}

Table 3 – Attribute Selectors (CSS Selectors - CSS: Cascading Style Sheets | MDN, 2022)

2.2.5 Specificity

The specificity algorithm is used by a browser in order to decide which CSS declaration holds the most relevancy to an element. It also calculates the importance of a CSS selector when using more than one to determine which rule to apply first. ID selectors hold the most relevancy followed by class, attribute, and pseudo-class selectors and lastly by type and pseudo-element selectors. The specificity can be overwritten using *!important* but, if possible, this should be avoided. (*Specificity - CSS: Cascading Style Sheets | MDN*, 2022)

2.3 Units

When setting a property for an element or its content, CSS units can be used to indicate its size. There are two different categories: absolute and relative units.

2.3.1 Absolute Units

Using an absolute unit for a property signifies that its size will remain the same, no matter the size of the window or parent element, which means pages using absolute units will not scale when the size of the screen changes. These units are therefore preferred when the responsiveness is of no importance for a project. (Mitchell, 2020)

Absolute Units include cm (centimetre), mm (millimetre), in (inches), pt (points), px (pixels), with 1px being equivalent to 0.75pt and pc (picas), with 1pc being equivalent to 12pt. Pixels are the most common used absolute unit for screens, while cm, mm and in are more popular for print. (Mitchell, 2020)

2.3.2 Relative Units

When creating responsive sites, that scale according to the window size or parent element, relative units are very useful. In general, it is recommended to use relative units as the default in order to avoid later having to update the styles for different screens. Table 4 lists the three most common used relative units.

Relative Unit	Description	Use-Case Examples
%	Relative to parent element's value for that property	The child element should have x% of the parent's width as a margin.
em	Relative to the current font-size of the element	The font of a child element should be half the size of its parents' font-size.
rem	Relative to the font-size of the root	The font-size should be twice the size as the root element's font.

Table 4 – Relative Units (Mitchell, 2020)

2.4 The Box Model

Each element is presented as a box by the rendering engine of a browser. The box model describes how those elements regarding their content, padding, border, and margin of a box, as seen in Figure 1, are combined to construct the box that can be viewed on a page. The shorthand properties *padding*, *border* and *margin* can be used to size the box. (*Introduction to the CSS Basic Box Model - CSS: Cascading Style Sheets* | *MDN*, 2022)

The content area is limited by the content edge and displays any content like text or an image. If the element is a block element and the property *box-sizing* is set to the default value *content-box*, the content area can be sized using properties such as *width*, *max-width*, *height*, and *min-height*. (*Introduction to the CSS Basic Box Model - CSS: Cascading Style Sheets | MDN*, 2022)

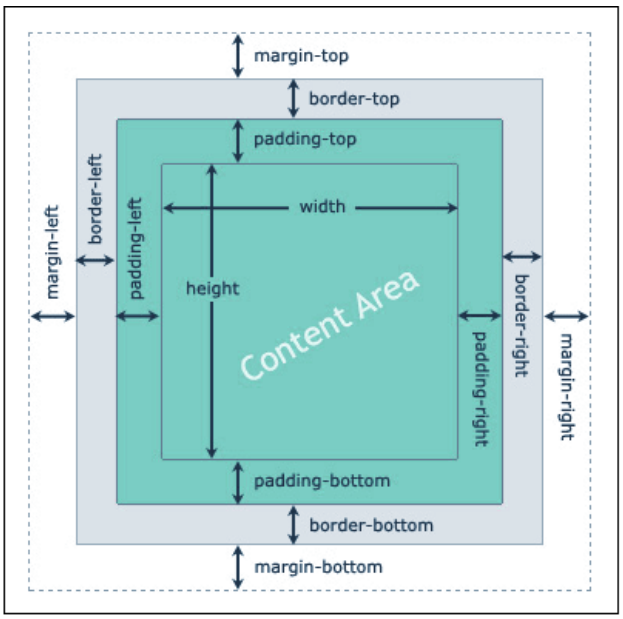

Figure 1 - The Box Model (https://static.javatpoint.com/csspages/images/css-box-model.png)

The content area is extended by the padding area which is limited by the padding edge and contains the padding of the element which creates space around an elements content. Padding properties such as *padding-top* and *padding-right* condition the thickness of the area. (*Introduction to the CSS Basic Box Model - CSS: Cascading Style Sheets | MDN*, 2022)

The border area wraps around the previous two and contains the border of the element. Shorthand border properties determine the thickness of the border and should the property *box-size* be set *border-box*, it is possible to further define the border area's size using properties such as *width*, *min-width*, *height*, or *max-height*. The background of a box will extent to the out edge of the border, with the prerequisite that the box has a set background such as an image or colour. (*Introduction to the CSS Basic Box Model - CSS: Cascading Style Sheets | MDN*, 2022)

In order to separate the element from its neighbouring element the border area is extended by a margin area, which includes an empty space that is limited by the margin edge. The margin area can be sized using margin properties such as *margin-top* and

margin-left. If the margin is not clearly determined margin collapsing will occur, which combines the top and bottom margin of a block into one, which will be the size of the largest lone margin. (*Introduction to the CSS Basic Box Model - CSS: Cascading Style Sheets* | *MDN*, 2022)

2.5 CSS Layout

The browser represents each element as a box when laying out the document. There are two types, block boxes and inline boxes, which refer to how the box will act in regard to page flow and other boxes on the same page. Each box contains an outer display type of block, such as the HTML elements <h1> or <p>, or of inline, such as the HTML elements <a> or , and an inner display type. (*Introduction to the CSS Basic Box Model - CSS: Cascading Style Sheets* | *MDN*, 2022) CSS offers several layout options and techniques which can be changed using the *display* property with the layout option as the value. The following layout methods are the most commonly used ones.

2.5.1 Normal Flow

If no specific layout is specified the elements will lay out according to the normal flow of the webpage with regards to their margin, border, and padding, see chapter 2.4. The content of block level elements will fill out the available inline space of their parent element and the size of the block while the size of inline elements is limited to the size of the content, meaning the height and width cannot be set for inline elements apart from images. Using the *display* property, the size of inline elements can be controlled by setting it to *inline-block* which will cause it to act like a block level element. (*CSS Layout - Learn Web Development* | *MDN*, 2022)

2.5.2 Flexbox

Flexbox, a one-dimensional layout method, is used when laying out items in either a row or columns and cause items to stretch or shrink to fill out the available space. The flex model consists of a main-size axis running from left to right, also referred to as main start to main end and a cross-size axis running from top to bottom, also referred to as cross start to cross end. It provides the opportunity of vertically and horizontally centring blocks of content inside their parents. In order to use flexbox properties, the

parent element, also called container needs to have the property *display* set to *flex*. (*Flexbox - Learn Web Development* | *MDN*, 2022)

2.5.3 Grids

Grids, a two-dimensional layout method, lay out the content in rows and columns. This method will prevent elements from jumping around or changing their widths when moving from one page to the next. A container including the child elements which should be laid out in a grid needs to include the *display* property set to *grid*. Next, the *grid-template-column* property needs to be specified within the container to enable the grid layout by setting up the widths of the columns. Further gaps between the boxes can be set by the shorthand property *gap* which takes in first the gap between rows and columns. (*CSS Layout - Learn Web Development* | *MDN*, 2022)

2.5.4 Floats

To create multiple column layout the property *float* can be used. The element containing the *float* property will be taken out of the normal flow of a page and set to where the value specified which can be *left* or *right* for example. The remaining content of the page will simply wrap around the floated element and continue its normal flow. To stop the following element from wrapping around the floated one the *clear* property can be applied to the following element which will push it below the floated one. Should both elements be wrapped in one block for example a the *clear* property will not work. Instead, the *clearfix hack* needs to be used. This means that some generated content needs to be insert first after the block containing the *float* and the content wrapping around it and then *clear* needs to be set to *both*. An alternative to this is to use the *overflow* property and setting it to *auto*. (*CSS Layout - Learn Web Development* | *MDN*, 2022)

2.5.5 Positioning

The CSS *position* property defines the way an element is positioned with its values and the properties *left*, *right*, *top*, and *bottom* determining their final location in a document. The value *static* positions an element in relation to the normal flow of the given document. The value *relative* positions an element relative to its normal position with an offset determined by the values *left*, *right*, *top*, and *bottom*. This offset does not affect any other elements. If the element should be taken out of the flow of the

document, it is positioned *absolute*. Its position is relative to its closed positioned ancestor. This results in other elements being positioned as if it was not existent and therefore causing the element to potentially overlap with others. When the position of an element is set to *fixed* it will be positioned relative to the viewport causing it to stay in that exact place when scrolling along the page. An element can also be *sticky*, which means it is positioned according to ones scroll position. It is a mixture of relative and fixed positioning, meaning it will act as if relatively positioned up until a given offset position is reached, then it will stick in place. (*Position - CSS: Cascading Style Sheets | MDN*, 2022)

2.6 Custom Properties

CSS custom properties, also referred to as CSS variables, are defined by the developer and contain certain values which are to be reused in multiple places across the stylesheet. This is often the case for complex websites with large amounts of CSS with repeating values. CSS variables store these values in one single place which can then be referenced throughout the CSS. Custom property notation, such as *--main-color:white;* is used to specify the CSS variables and the function *var()* is used to access them which would look like the following: *color: var(--main-color);* The declaration always uses a double hyphen -- followed by a custom property name and a valid CSS property value. (*Using CSS Custom Properties (Variables) - CSS: Cascading Style Sheets | MDN*, 2022)

These custom properties also come with the benefit of sematic identifiers. For example, --main-headings-color would then substitute using #ff00ff later on which is easier to understand. (*Using CSS Custom Properties (Variables) - CSS: Cascading Style Sheets | MDN*, 2022)

2.7 ARIA and Automatic.CSS

ARIA stands for Accessible Rich Internet Applications and defines various roles and attributes, which have the purpose of making web content and applications more accessible for people with disabilities. (*ARIA - Accessibility | MDN*, 2022)

Automatic.css is a utility CSS framework, which applies these ARIA for CSS by providing utility classes and CSS variables, essentially offering automatic features to make a website more accessible to different users. (Geary, 2022a)

For example, the typography system, which refers to all text and headings being automatically responsive by using relative units, as they are scalable and can be of a great advantage when the user makes changes in their browser preferences for example to increase the default text size due to issues with their eyesight. Websites using absolute units opposed to relative ones will not respond according to those changes, as they are static units. (Geary, 2022a)

Many users also rely on colour contrast in order to see the information on a website. Automatic.css uses a strong colour system with auto-generated shades, which also gives you the ability to adjust them along with the lightness values and saturation. (Geary, 2022b)

Another way this CSS framework provides accessibility is by hiding certain elements from regular users while preserving those exact elements for accessibility users. For example, a certain icon might be enough for a regular user, a label naming the icon is not necessarily needed for them, but for those users using a screen reader, labelling icons is an absolute must. Automatic.css is equipped with a *.hidden-accessible* class which uses certain techniques to hide the content from specific users while at the same time providing it for those using a screen reader. (Geary, 2022b)

The Automatic.css framework comes with many more features giving one the ability to build an accessible website, such as automatic grids, variable hooks, and a spacing system. (Geary, 2022a)

3 CSS Architecture

A CSS architecture can be regarded as a set of guidelines and is essential for every project in order to work efficiently, keep the stylesheet with a consistent, reusable, and maintainable code. It gives CSS a clear structure and makes working on a project much more efficient especially once the project starts to grow. There are many different architectures but every single one tries to achieve efficiency in slightly different ways. Some of the most used architectures include OOCSS – Object Oriented CSS, BEM – Block Element Modifier, ACSS – Atomic CSS and SMACSS – Scalable and Modular Architecture for CSS. (Petrovic, 2022)

3.1 Commonly Used Architectures

Object Oriented CSS applies object-oriented concepts to CSS and follows two key principles, the first being the separation of structure and skin. In this context structure refers to the styles impacting the layout, while skin refers to any styles impacting the elements, such as colours or font styles. The second principle, the separation of container and content, indicates that you should avoid location-specific styles. (Petrovic, 2022)

Block Element Modifier or BEM makes use of a consistent naming convention for the creation of style classes applied to elements in the HTML document. This makes it possible to write independent CSS blocks which provide reusable code. The architecture specifically recommends the following naming convention: *block-name_element-name--modifier-name*. An example could look like the following: *.fruits__price--large* It is important to remember to only use BEM with classes, not with IDs. (Holden, 2021)

The focus of Atomic CSS lies in creating various small CSS utility classes, which can then be used in the HTML document. ACSS has similarities to OOCSS, as it also proposes separating duplicated property-value pairs, but ACSS definitely uses a more extreme approach to this technique by suggesting the creation of styles for even the smallest attainable level. Further, ACSS also recommends using properties in the naming convention of the selectors, which means while you might use *.skin* in OOCSS, in ACSS you would use *.color-skin* for example. (Holden, 2021)

While all of the previous architectures focused on class names, SMACSS puts its aim more on CSS folder/file organization. SMACSS promotes breaking the CSS down based on five CSS rules: 1.Base, 2.Layout, 3.Module, 4.State, 5.Theme. The categorization of styles based on these five rules helps identify patterns and further outline better practices for each pattern, which as a result makes up less and easier to maintain code and provides the user with a better and more consistent experience. (Holden, 2021)

3.2 Which Architecture Should be Used?

Now, the question that remains is, what architecture should be chosen? The answer is it really does not matter. It is recommended to simply choose the architecture one is the most comfortable with, the one preferred and if working in a team, the one every member agrees on in order to keep the format consistent. The most important aspect of CSS architectures simply is to maintain a consistent style. Whichever architecture is chosen, it is highly recommended to describing the architecture used in the documentation of the project, so the guidelines can be more easily enforced and kept up, and new team members can catch up quickly. (Holden, 2021)

4 Implementation of CSS in HTML

There are three ways of how CSS can be included in an HTML document, inline styles, embedded styles, and external stylesheets.

4.1 Inline Styles

When using inline styles, the style rules are applied by directly adding the CSS rules into the starting tag of an HTML element and are defined using the *style* attribute as seen in Code 1.

```
<h1 style="color:green; font-size:40px; font-style:sans-serif;">H1</h1>
```

Code 1

This way of implementing CSS in HTML is regarded as a bad practice, as the content of the document and the presentation are being mixed which can cause issues with maintaining the code. But it is sometimes used when defining a single unique style for one specific element. (*How to Include CSS in HTML Pages - Tutorial Republic*, n.d.)

4.2 Embedded Styles

Embedded Styles, also referred to as internal styles solely affect the document, they are inserted in. They are defined using the <style> element within the <head> section of the HTML document as seen in Code 2. Since this embedding of styles makes it impossible to share styles between documents, this method is not recommended either. (*How to Include CSS in HTML Pages - Tutorial Republic*, n.d.)

```
<head>
  <style>
    body {
      background-color: magenta;
    }
    p {
      color: black;
      font-size: 20px;
    }
  </style>
</head>
```

Code 2

4.3 External Stylesheets

The option of using external stylesheets is usually of an advantage when making changes to several pages, as it enables you to change the look of an entire website by

only changing one singular file. Additionally, it provides great reusability. The external stylesheet is implemented into the HTML document in the <head> section by using the <link> tag as done so in Code 3. The *href* attribute takes in the address of the file. (*How to Include CSS in HTML Pages - Tutorial Republic*, n.d.)

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

Code 3

Another option of implementing an external stylesheet is by importing it using the *@import* statement. Code 4 shows how this can be done by including *@import* in the <style> section of the header of an HTML document. After the *@import* statement, the <style> element may include some other CSS rules, which shows the use of embedded styles. (*How to Include CSS in HTML Pages - Tutorial Republic*, n.d.)

```
<head>
   <style>
      @import url("externalsheet.css");
   </style>
</head>
```

Code 4

4.4 How to Create an External Stylesheet Using Chrome and Visual Studio Code

The first step is to download and install Chrome by following the instructions on the Google Chrome Website. Next, download Visual Studio Code, also referred to as VS Code and not to be confused with Microsoft's Visual Studio, for the needed operating system also following the instructions of their website.

After downloading and installing the tools, open up VS Code and choose the option *Open Folder* on the starting page listed under *Start* and open the folder to work in. Next create a new file and save it to the folder using any name adding the ending *.html* to create an HTML document. The next step is to fill the document with some basic code to create a very simple looking website. A helpful shortcut in VS Code in order to be provided with an HTML skeleton is *!+Tab*. As a following step, some HTML elements need to be added for the purpose of later styling them with CSS.

To create a CSS file, create another new file in VS Code preferably in the same folder, giving it any name while adding *.css* to the end. Now, the file needs to be linked to the

HTML document using the <link> element in the head section of the document, see Chapter 4.3.

The web page can then be viewed in the browser by opening up the HTML document from the file explorer. The page will need to be refreshed after any changes have been made and saved to the HTML document or stylesheet.

5 Nutshell Examples

The following examples will be done using an external stylesheet, the web browser Chrome, and the code editor Visual Studio Code. While Chrome and VS Code are the preferred tools for most web developers, any other browser and code editor will work as well.

5.1 Simple Examples – Travel Bucket List

The following examples will be done using the very simple HTML document which shows a list of travel destinations seen in the Appendix, Code 28, and will show how to edit it using differently CSS styling properties. Due to the fact that these examples are focused on the styling of the document, the submit button will not actually function, as this would exceed the abilities of CSS. The finished and complete stylesheet can be found in the appendix, Code 29.

5.1.1 How to Apply and Change the Colour of Text and Background

This simple example shows how to apply colour to the <h1> and <button> element. In the external stylesheet the <h1> element is selected, and the styling property *color* is defined, which applies a colour to the element.

```
h1 {
    color: green;
}
```

Code 5

The value describing the colours in Code 5 is written in a human-readable style using the colour names. The value could also be written using RGB/RGBA or hexadecimals.

The RGB Value uses the formula rgb(red, green, blue), with each parameter (red, green, blue) defining the intensity of a colour between 0 and 255. For example, the colour green could be display using *rgb(0,255,0)*. The RGBA Value is an extension to RGB with an alpha channel and uses the formula rgba(red, green, blue, alpha) with the alpha parameter specifying the opacity of a colour with a number between 0.0 and 1.0. (Groves, n.d.)

Using hexadecimals values is the most common used technique in CSS to specify colours. Hexadecimal values are preceded with a # symbol followed by six hex numbers between 0-9 and a-f. As a reference: the colours white and black have the syntax *#000000* and *#ffffff*. (Groves, n.d.)

Therefore Code 5 could also look like Code 6.

```
h1 {
    color: rgba(72, 159, 181, 0.925);
}
```

Code 6

In the HTML document, Code 28 there are certain elements with the class *must* applied to them. Those cities are the ones considered an absolute must to be visited. In Code 7 they are therefore specifically selected using the class selector and coloured differently using hexadecimals.

```
.must {
    color: #82c0cc;
}
```

Code 7

The overall colour of the whole HTML document can be changed by selecting the <html> element and setting *background-color* to the desired colour as seen in Code 8.

```
html {
    background-color: #ede7e3;
}
```

Code 8

5.1.2 How to Style a Document Using Common Text Properties

The next example shows how to style a HTML document using some of the most commonly used text properties, starting with *text-align*. This property is used to set the alignment of the inline-level content horizontally inside a block element. In Code 9 all elements are selected using the universal selector * and centred using *text-align* and the value *center*.

```
* {
    text-align: center;
}
```

Code 9

The bullet points of the list can be centred with the list, by selecting *ul* and setting the *display* property to *inline-block*, or they can simply be removed completely by setting the property *list-style-type* to *none* as seen in Code 10.

```
ul {
    list-style-type: none;
}
```

Code 10

The needed text property to set the weight or boldness of a text font is *font-weight* in combination with a value using keywords such as *bold* or *lighter* or using numbers from 100 to 900 as used in Code 11 where the text in the <p> and <ul> elements is set to become bolder. It is important to note that using *font-weight* also comes with the limitation of the used *font-family*.

```
p {
    font-weight: 600;
}
ul {
    font-weight: bolder;
}
```

Code 11

Another text property often used is *text-decoration*, which controls the appearance of decorative lines on text, this can include underlines, overlines or even a line going through the text. The colour can be specified as well as the line style and the thickness. The order of the values does not matter, they simply need to be separated with a white space. This text property is most commonly used to remove default underlines such as the ones that come with <a> elements by setting *text-decoration* to *none*. In Code 12 *text-decoration* was used to cross off those cities already visited, which the class *done* has been applied to.

```
.done {
    text-decoration: line-through black;
}
```

Code 12

When setting the height of a line box, the property *line-height* is used to specify the distance between lines. Used on block-level elements, it sets the minimum height of line boxes, while it sets the height used to calculate the line box height for non-replaced inline elements. None-replaced inline elements are those whose content is displayed and not replaced with something else, for example the content within a <p> element will be displayed while the content within an <img> element will be replaced with the

actual image. The height is specified using either a number, a length, a percentage, or a keyword such as *normal*. In general, it is more common to use unitless numbers. In Code 13 the number 2.5 is used to set the height of lines of the unordered list elements to make them appear further apart.

```
ul {
    line-height: 2.5;
}
```

Code 13

The horizontal space between letters can be set using the text property *letter-spacing*. Using positive values, the characters will spread apart, while using negative values will bring them closer together. In Code 14 this is done by spacing out the letters of the <h1> element by 5 pixels.

```
h1 {
    letter-spacing: 5px;
}
```

Code 14

To change the overall font size, the property *font-size* with a value is needed, which can either be a keyword such a *smaller* or *x-large*, a length or a percentage, which are relative to the font size of the element. The difference to the *font-weight* property is that it does not increase the thickness of characters, only the size. In Code 15 this can be seen as the font size of the content within the <p> and <ul> elements is increased while the font weight has been made bolder, see Code 11. The font size of the <ul> and <h1> elements has also been increased to match it to the text in <p>.

```
p {
    font-weight: 600;
    font-size: 150%;
}

ul {
    font-weight: bolder;
    font-size: 120%;
}

h1 {
    font-size: 300%;
}
```

Code 15

5.1.3 How to Change the Font Style

Changing the font style hugely depends on the fonts built into the browser being used, which is why it is recommended to rely on common browser or website fonts as those

are the ones most users will have in their browser. The property used to change the font is *font-family* with the font style as the value. In Code 16 the font of the overall document is changed to be *Courier New* in combination with a font stack which lists the fonts in the order of how they should be applied if the first one is not available. In other words, the font stack acts as a backup.

```
* {
    font-family: 'Courier New', Courier, monospace;
}
```

Code 16

5.1.4 How to Use Box Model Properties

The <form> element and those nested in the element at the bottom of the HTML document, Code 28, can be styled using properties first introduced in Chapter 2.4. To start out, the *box-sizing* property should be set to *border-box* for the whole document. This will result in the browser regarding any padding and border added, effectively decreasing the elements width by the value specified for padding and border.

```
* {
    box-sizing: border-box;
}
```

Code 17

The Code 18 shows how the <input> elements can now be styled using an attribute selector to select them. The properties *width* and *height* are used to increase the size of the input boxes. The *margin* is set to 5 pixels to increase the distance between the elements just a little. *padding* is not necessarily needed for this example. The shorthand *border* property is used to set the width of the border, the colour and also the style, while setting the *outline* property to *none* to stop the border from changing when the input box is selected. Further, the *border-radius* property is set to round off the boxes by 10 pixels.

```
input[type="text"] {
    font-size: 120%;
    width: 300px;
    height: 50px;
    margin: 5px;
    border: 2px solid #16697a;
    outline: none;
    border-radius: 10px;
}

input[type="submit"] {
    font-size: 120%;
    font-weight: bolder;
```

```
    width: 120px;
    height: 50px;
    margin: 5px;
    border: 2px solid #16697a;
    background-color: #16697a;
    border-radius: 10px;
}
```

Code 18

5.1.5 How to Put a Shadow on an Element

In Code 19 the boxes created by the <input> elements are given a slight shadow using the *box-shadow* property. The value defined for the property includes four numbers followed by a colour. The first number sets the horizontal offset, a positive number will create a shadow on the right side of the box, while a negative one will put it on the left side. The second number sets the vertical offset, here a positive number puts the shadow below a box and a negative one above it.

The blur radius is set by the third number and determines how blurry a shadow will be. Setting the number to 0 will result in a very sharp shadow, while a higher number will result in a more blurred shadow. The higher the number, the more the shadow will also extend, meaning if the horizontal offset is 10px and the blur radius is set to 10px the total of the shadow will be 20px.

The last number, the spread radius is in comparison to the other three optional. The size of the shadow will increase when using positive numbers, while using negative ones reduces it.

For shadows it is most common to use a more transparent colour by setting the opacity to for example 0.4 or less. Using the keyword inset as a value before specifying the four numbers will create an inner shadow.

```
input[type="text"] {
    box-shadow: 0 0 10px 3px rgba(0, 0, 0, 0.35);
}

input[type="submit"] {
    box-shadow: 0 0 10px 3px rgba(0, 0, 0, 0.35);
}
```

Code 19

5.1.6 How to Create a Hover Effect

The Code 20 demonstrates how to create a hover effect on the submit button. First, the element needs to be selected using an ID selector followed by the pseudo-class

:hover, which creates the hover effect. Then the element can be styled by setting a *background-color* which the button will change to when hovered over. Further, the cursor will change to a hand pointer when hovering over by setting the *cursor* property to *pointer*.

```
#add:hover {
    background-color: #82c0cc;
    cursor: pointer;
}
```

Code 20

5.1.7 How to Position Elements

In Code 21 the <form> element is positioned by using the *position* property, see Chapter 2.5.5 with the value *sticky* making it stick to the bottom of the page where it will stay and therefore overlap with the other elements when scrolling up the page.

```
form {
    position: sticky;
    bottom: 10px;
}
```

Code 21

5.2 More Complex Examples – Advent Calendar

The next examples will use the following HTML document and styles as seen in the Appendix, Code 30, and Code 31, and will show how to create an advent calendar with six boxes in each row, which change when hovered over and how to make it responsive to different screen sizes. The finished and complete stylesheet can be found in the Appendix, Code 32.

5.2.1 How to Create a Layout Using Flexbox

The HTML document, Code 30 has four containers which are selected and set to *flex* in Code 22, further some sizing properties are applied to the containers.

The direction of the main axis is specified using the property *flex-direction* which can be set to either *column* or *row*, as done so in Code 22. Items can also be laid out using reverse directions values *column-reverse* or *row-reverse*.

When a fixed height and width are set for the boxes the flex items might overflow their container, eventually destroying the layout. To keep them from overflowing, the

property *flex-wrap* can be set to *wrap*, which will cause any overflowing items to move to the next line.

Flexbox also offers properties used to align items along the main and cross axis. *justify-content* specifies how the content or items are distributed across the main axis and can be used with the values such as *flex-start* the default, *center*, or *space-evenly* which spaces out the items evenly according to the available space. To distribute the space between items along the cross axis the property *align-items* is used with values such as *stretch* the default, *flex-end* or *center* which centres the items on the cross axis. If the behaviour of one specific item in a container should be changed, the property *align-self* can be used with the same values as *align-items*.

```
#container1,
#container2 {
    display: flex;
    width: 90%;
    height: 100%;
    margin: 3% auto;   /*margin: top and bottom = 3%, automatically calculate left and right = auto*/
    flex-direction: row;
    justify-content: space-evenly;
    align-items: center;
}
```

Code 22

To center the content within the flex items, flexbox properties are applied to the elements in Code 23.

```
section div {
    justify-content: center;
    align-items: center;
}
```

Code 23

5.2.2 How to Create a Transition

The shorthand CSS *transition* property is used to change the state of an element by specifying the transition for it. States can be defined using pseudo-classes. The property takes in up to four values, with the first being the property to which the transition should be applied to, for example the *background-color* or also *all*. The next value specifies the duration of the transition defined by a number of seconds. Next, the timing function can be defined using values such as *ease-in* or *steps(4, end)* which will display the function in four steps ending the animation with the last one. Lastly, the value for the delay can be added, causing a delay defined by a number of seconds to the transition effect. (*Transition - CSS: Cascading Style Sheets | MDN*, 2022)

In Code 24 the transition is applied to all properties of the elements and to the *opacity* property of the <img> elements with an effect duration of three and four seconds.

```
section div {
    transition: all 3s;
}

img {
    transition: opacity 4s;
}
```

Code 24

Code 25 sets the other state of the and <img> elements using the pseudo-class :*hover*. Now, when hovering over the elements the images will appear.

```
section div:hover {
    color: transparent;
    box-shadow: none;
    background-color: #ffffff;
}

img:hover {
    opacity: 1;
}
```

Code 25

5.2.3 How to Rotate an Element

This example shows how the CSS property *transform* can add a rotation to an element. The transform property gives one the ability to rotate, skew, scale, or translate a transformable element which include all those whose layout is controlled by the box model with some small exceptions. In Code 26 *transform* was used with the function *rotate()* which causes the elements to rotate by 360 degrees when hovered on.

```
section div:hover {
    transform: rotate(360deg);
}
```

Code 26

5.2.4 How Make the Design Responsive Using Media Queries

In Code 27 specific media features are targeted by using media queries to create a responsive design. Media queries were introduced in CSS3 and can be applied using the at-rule *@media* followed by *only screen* which will prevent older browsers, not supporting media queries with media features, from applying the specified styles, and

lastly one or more conditions which must be true in order for certain styles to be included. The defined minimum and maximum widths in Code 27 are commonly used breakpoints for widths of devices and make up the condition needed to be true in order for the sizes of the elements to change. The Code 27 will therefore result in a much more responsive webpage with , <h1> and <img> elements changing their sizes accordingly.

```
@media only screen and (max-width: 480px) {
    section div {
        width: 60px;
        height: 60px;
        font-size: 2em;
        margin-left: 2px;
        margin-right: 2px;
    }
    h1 {
        font-size: 3em;
    }
    img {
        width: 0.5em;
        height: 1em;
    }
}

@media only screen and (min-width:481px) and (max-width: 768px) {
    section div {
        width: 100px;
        height: 100px;
        font-size: 2em;
        margin-left: 2px;
        margin-right: 2px;
    }
    h1 {
        font-size: 4em;
    }
    img {
        width: 1em;
        height: 1.5em;
    }
}

@media only screen and (min-width:769px) and (max-width: 1024px) {
    section div {
        width: 170px;
        height: 170px;
        font-size: 3em;
        margin-left: 2px;
        margin-right: 2px;
    }
    img {
        width: 1.5em;
        height: 2em;
    }
}
```

Code 27

6 Conclusion and Outlook

Hakon Wium Lie's goal in 1994 was it to create a stylesheet language to give people the possibility of styling documents, such as HTML documents and in 1996 the first version of CSS was released. The World Wide Web Consortium has since take on the job of updating and developing new features, which make web design more responsive and accessible to all.

CSS generally follows a syntax of a *property:value* pair preceded by one or more selectors, except for at-rules which are often used for importing other external stylesheets or sizing pages according to certain media features. CSS offers absolute and relative units, although relative units are preferred when creating responsive designs.

Before starting any project, the rules of the architecture used should be set, to ensure a consistent style of the CSS code. Every architecture aims at making work more efficient, while keeping the style consistent and the code reusable in their own slightly different way.

The nutshell examples have shown an easy way of creating simple responsive web design which could be further extended using a scripting language such as JavaScript to make the design dynamic and program the behaviour of the web page. Further frameworks such as Bootstrap could be used to build web pages, making the web development process much faster and pleasant while giving the developer the opportunity of implementing advanced CSS features. Using frameworks can also increase the compatibility with several browsers and the different versions of them.

Over the past few years, accessibility and especially accessible web design have gained great attention. The framework automatic.css was developed to cater to the needs of impaired users enabling developers to create accessible website by providing them with new CSS features. Automatic.css and ARIA are already paving the way to an overall accessible web design, now the question that arises is: how will other features of CSS be adopted or developed to fit to the needs of special users?

References

A brief history of CSS until 2016. (n.d.). Retrieved October 26, 2022, from https://www.w3.org/Style/CSS20/history.html

ARIA - Accessibility | *MDN*. (2022, October 3). Retrieved November 23, 2022, from https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA

CSS History. (n.d.). Retrieved October 26, 2022, from https://simplecss.eu/css-history-brief-overview.html

CSS layout - Learn web development | *MDN*. (2022, September 9). Retrieved December 7, 2022, from https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout

CSS selectors - CSS: Cascading Style Sheets | *MDN*. (2022, November 5). Retrieved November 24, 2022, from https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors?retiredLocale=de

Flexbox - Learn web development | *MDN*. (2022, November 23). Retrieved December 9, 2022, from https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox

Geary, K. (2022a, February 28). *Why Automatic.css is Different From Other Frameworks*. Automatic.css. Retrieved November 23, 2022, from https://automaticcss.com/why-automatic-css-different/

Geary, K. (2022b, August 7). *How Automatic.css Makes Websites More Accessible* | *ACSS*. Automatic.css. Retrieved November 23, 2022, from https://automaticcss.com/accessibility-features/

GeeksforGeeks. (2022, June 8). *CSS Syntax and Selectors*. Retrieved November 8, 2022, from https://www.geeksforgeeks.org/css-syntax-and-selectors/

Groves, D. (n.d.). *Defining Colors in CSS*. Retrieved November 24, 2022, from http://web.simmons.edu/%7Egrovesd/comm244/notes/week3/css-colors

Holden, I. (2021, February 2). *CSS Architecture Style Guides For Frontend Developers*. HackerNoon. Retrieved October 17, 2022, from https://hackernoon.com/css-architecture-style-guides-for-frontend-developers-lj28332a

How CSS works - Learn web development | MDN. (2022, September 13). Retrieved October 23, 2022, from https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/How_CSS_works

How to Include CSS in HTML Pages - Tutorial Republic. (n.d.). Retrieved November 23, 2022, from https://www.tutorialrepublic.com/css-tutorial/css-get-started.php

Introduction to the CSS basic box model - CSS: Cascading Style Sheets | MDN. (2022, November 30). Retrieved November 22, 2022, from https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Introduction_to_the_CSS_box_model

Kumar, S. (2022, May 13). *History of CSS*. The Crazy Programmer. Retrieved November 26, 2022, from https://www.thecrazyprogrammer.com/2021/11/history-of-css.html

Mitchell, J. (2020, September 3). *CSS Units Explained*. DigitalOcean Community. Retrieved November 22, 2027, from https://www.digitalocean.com/community/tutorials/css-css-units-explained

Petrovic, B. (2022, September 14). *Widely Used CSS Architectures and How They Function*. CQL. Retrieved November 22, 2022, from https://www.cqlcorp.com/insights/widely-used-css-architectures-and-how-they-function/

position - CSS: Cascading Style Sheets | MDN. (2022, November 29). Retrieved December 9, 2022, from https://developer.mozilla.org/en-US/docs/Web/CSS/position?retiredLocale=de

Pseudo-classes and pseudo-elements - Learn web development | MDN. (2022, November 30). Retrieved November 22, 2022, from https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors/Pseudo-classes_and_pseudo-elements

Specificity - CSS: Cascading Style Sheets | MDN. (2022, October 30). Retrieved November 23, 2022, from https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity

Syntax - CSS: Cascading Style Sheets | MDN. (2022, September 28). Retrieved October 17, 2022, from https://developer.mozilla.org/en-US/docs/Web/CSS/Syntax

transition - CSS: Cascading Style Sheets | MDN. (2022, September 28). Retrieved December 9, 2022, from https://developer.mozilla.org/en-US/docs/Web/CSS/transition?retiredLocale=de

Using CSS custom properties (variables) - CSS: Cascading Style Sheets | MDN. (2022, September 28). Retrieved December 14, 2022, from https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties

What is CSS? - Learn web development | MDN. (2022, September 13). Retrieved October 17, 2022, from https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS

Appendix

Code 28 defines the HTML document used for the Nutshell Examples of Chapter 5.1. It represents a list of cities when viewed in the browser.

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

<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Travel Bucket List</title>
   <link rel="stylesheet" href="travel_list.css">
</head>

<body>

   <h1>My Travel Bucket List</h1>
   <div>
      <p> These are some cities I would love to visit one day!</p>
   </div>
   <hr>
   <ul>
      <li>Barcelona, Spain</li>
      <li class="must done">London, UK</li>
      <li>Tokyo, Japan</li>
      <li class="must">Honolulu, Hawaii</li>
      <li>Helsinki, Finland</li>
      <li class="done">Paris, France</li>
      <li class="must done">New York City, USA</li>
      <li>Toronto, Canada</li>
      <li class="must">Edinburgh, Scotland</li>
      <li class="must">Auckland, New Zealand</li>
      <li class="done">Rome, Italy</li>
      <li>Seoul, South Korea</li>
      <li>Oslo, Norway</li>
      <li class="done">Galway, Ireland</li>
      <li class="must">Amsterdam, Netherlands</li>
   </ul>
   <form>
      <input type="text" name="city" id="name" placeholder="City, Country">
      <input type="submit" value="Add City" id="add">
   </form>

</body>

</html>
```

Code 28 - HTML Document for chapter 5.1

Code 29 defines the finished stylesheet for Chapter 5.1 and contains all the styles applied to the HTML document, Code 28.

```
html {
   background-color: #ede7e3;
}
```

```css
* {
    box-sizing: border-box;
    text-align: center;
    font-family: 'Courier New', Courier, monospace;
}

ul {
    list-style-type: none;
    font-weight: bolder;
    line-height: 2.5;
    font-size: 180%;
}

h1 {
    color: rgba(72, 159, 181, 0.925);
    letter-spacing: 5px;
    font-size: 370%;
}

p {
    font-weight: 600;
    font-size: 180%;
}

.done {
    text-decoration: line-through black;
}

.must {
    color: #82c0cc;
}

/* styling the <input> elements in the <form> element */
input[type="text"] {
    font-size: 120%;
    width: 300px;
    height: 50px;
    margin: 5px;
    border: 2px solid #16697a;
    outline: none;
    border-radius: 10px;
    box-shadow: 0 0 10px 3px rgba(0, 0, 0, 0.35);
}

input[type="submit"] {
    font-size: 120%;
    font-weight: bolder;
    width: 120px;
    height: 50px;
    margin: 5px;
    border: 2px solid #16697a;
    background-color: #16697a;
    border-radius: 10px;
    box-shadow: 0 0 10px 3px rgba(0, 0, 0, 0.35);
}

/* adding the hover effect */
#add:hover {
    background-color: #82c0cc;
    cursor: pointer;
}
```

```
/* positioing the <form> element */
form {
    position: sticky;
    bottom: 10px;
}
```

Code 29 - The finished style sheet for chapter 5.1

Code 30 defines the HTML document used for the Nutshell Examples of Chapter 5.2. It represents the skeleton of an advent calendar when viewed in the browser.

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

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="advent_calendar_app.css">
    <title>Advent Calendar</title>
</head>

<body>
    <h1>Advent Calendar</h1>
    <section id="container1">
        <div>
            <h4 class="day">1</h4>
            <img src="https://content.mycutegraphics.com/graphics/christmas/dog-wearing-santa-hat.png"
                alt="A Dog Wearing a Santa Hat">
        </div>
        <div>
            <h4 class="day">2</h4>
            <img src="https://content.mycutegraphics.com/graphics/christmas/popcorn-christmas-tree.png"
                alt="Christmas Tree with Popcorn">
        </div>
        <div>
            <h4 class="day">3</h4>
            <img src="https://content.mycutegraphics.com/graphics/christmas/elf-hat-red-green.png"
                alt="Red and Green Elf Hat">
        </div>
        <div>
            <h4 class="day">4</h4>
            <img src="https://content.mycutegraphics.com/graphics/christmas/christmas-owl.png"
                alt="Owl Wearing a Santa Hat">
        </div>
        <div>
            <h4 class="day">5</h4>
                <img  src="https://content.mycutegraphics.com/graphics/christmas/christmas-gift-with-big-
bow.png"
                alt="Christmas Gift with Big Bow">
        </div>
        <div>
            <h4 class="day">6</h4>
            <img src="https://content.mycutegraphics.com/graphics/christmas/boy-santa-christmas-list.png"
                alt="Boy Holding a Christmas Wish List">
        </div>
    </section>
    <section id="container2">
        <div>
            <h4 class="day">7</h4>
```

```
            <img  src="https://content.mycutegraphics.com/graphics/christmas/cute-reindeer-head.png"
alt="Reindeer">
      </div>
      <div>
        <h4 class="day">8</h4>
            <img  src="https://content.mycutegraphics.com/graphics/christmas/gingerbread-man.png"
alt="Gingerbread Man">
      </div>
      <div>
        <h4 class="day">9</h4>
        <img src="https://content.mycutegraphics.com/graphics/christmas/snowman-in-santa-hat.png"
          alt="Snowman Wearing a Santa Hat">
      </div>
      <div>
        <h4 class="day">10</h4>
        <img src="https://content.mycutegraphics.com/graphics/christmas/polar-bear-christmas-tree-in-
snow.png"
          alt="Polar Bear Next to Christmas Tree in the Snow">
      </div>
      <div>
        <h4 class="day">11</h4>
            <img  src="https://content.mycutegraphics.com/graphics/christmas/christmas-brown-bear-
drinking-cocoa.png"
          alt="Bear Wearing Santa Hat and Drinking Cocoa">
      </div>
      <div>
        <h4 class="day">12</h4>
        <img src="https://content.mycutegraphics.com/graphics/christmas/christmas-gift-blue.png"
          alt="Blue Christmas Gift">
      </div>
    </section>
    <section id="container3">
      <div>
        <h4 class="day">13</h4>
          <img  src="https://content.mycutegraphics.com/graphics/christmas/christmas-penguins-and-
christmas-tree.png"
          alt="Penguins Next to Christmas Tree">
      </div>
      <div>
        <h4 class="day">14</h4>
        <img src="https://content.mycutegraphics.com/graphics/christmas/reindeer-in-snow.png"
          alt="Reindeer in the Snow">
      </div>
      <div>
        <h4 class="day">15</h4>
        <img src="https://content.mycutegraphics.com/graphics/christmas/elf-hat-green.png" alt="Green
Elf Hat">
      </div>
      <div>
        <h4 class="day">16</h4>
          <img  src="https://content.mycutegraphics.com/graphics/christmas/penguin-eating-christmas-
cookies.png"
          alt="Penguin Eating Christmas Cookies">
      </div>
      <div>
        <h4 class="day">17</h4>
        <img src="https://content.mycutegraphics.com/graphics/christmas/christmas-tree-with-gifts.png"
          alt="Christmas Tree with Presents Underneath">
      </div>
      <div>
        <h4 class="day">18</h4>
        <img src="https://content.mycutegraphics.com/graphics/christmas/christmas-bear-with-gift.png"
```

```
            alt="Bear Holding Christmas Gifts">
        </div>
    </section>
    <section id="container4">
        <div>
            <h4 class="day">19</h4>
            <img src="https://content.mycutegraphics.com/graphics/christmas/elves-with-reindeer.png"
                alt="Elves With Reindeer">
        </div>
        <div>
            <h4 class="day">20</h4>
             <img src="https://content.mycutegraphics.com/graphics/christmas/dog-with-christmas-gift-and-
candy-cane.png"
                alt="Dog with Christmas Gift and Candy Cane">
        </div>
        <div>
            <h4 class="day">21</h4>
            <img src="https://content.mycutegraphics.com/graphics/christmas/christmas-gift-pink.png"
                alt="Pink Christmas Gift">
        </div>
        <div>
            <h4 class="day">22</h4>
            <img src="https://content.mycutegraphics.com/graphics/christmas/festive-christmas-tree.png"
                alt="Festive Christmas Tree">
        </div>
        <div>
            <h4 class="day">23</h4>
                <img  src="https://content.mycutegraphics.com/graphics/christmas/christmas-cat-with-candy-
cane.png"
                alt="Cat Wearing Santa Hat and holding Candy Cane">
        </div>
        <div>
            <h4 class="day">24</h4>
             <img src="https://content.mycutegraphics.com/graphics/christmas/santa-claus.png" alt="Santa
Claus">
        </div>
    </section>
</body>

</html>
```

Code 30 - The HTML Document for chapter 5.2

Code 31 defines the styles applied to the HTML document, Code 30. These styles where not specifically explained in Chapter 5.2 as similar styles have been applied in Chapter 5.1. For explanations of the styles in Code 31, refer to Chapter 5.1.

```
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

html {
      background:  url(https://media.istockphoto.com/id/1176728520/vector/winter-christmas-seamless-
pattern-background-vector-
illustration.jpg?s=612x612&w=0&k=20&c=Txhf7q_IKxRW8C0rtpsG8Um07V-Sge0umxocdLCwZ_8=);
}

h1 {
```

```
    font-size: 6em;
    font-family: "Copperplate Gothic Light", fantasy;
    text-align: center;
    color: #bc4749;
    margin-top: 2%;
}

img {
    width: 2.5em;
    height: 3em;
    opacity: 0; /*0 -> image is fully opaque*/
}

.day {
    position: absolute;
}

section div {
    width: 200px;
    height: 200px;
    background-color: #23856d;
    border: 3px solid #006f57;
    box-shadow: inset -0.3em -0.3em 0.6em rgba(0, 0, 0, 0.4), inset 0.3em 0.3em 0.6em rgba(0, 0, 0,
0.4);
    color: white;
    margin-left: 3px;
    margin-right: 3px;
    border-radius: 5%;
    margin-top: 2%;
    font-size: 4em;
}
```

Code 31 - CSS for chapter 5.2

Code 32 defines the finished stylesheet for Chapter 5.2 and contains all the styles applied to the HTML document Code 30. When displayed in the browser, the finished advent calendar with all styles applied to it can be seen. Due to the media queries, the calendar will scale according to the screen size.

```
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

html {
      background:  url(https://media.istockphoto.com/id/1176728520/vector/winter-christmas-seamless-
pattern-background-vector-
illustration.jpg?s=612x612&w=0&k=20&c=Txhf7q_IKxRW8C0rtpsG8Um07V-Sge0umxocdLCwZ_8=);
}

h1 {
    font-size: 6em;
    font-family: "Copperplate Gothic Light", fantasy;
    text-align: center;
    color: #bc4749;
    margin-top: 2%;
}
```

```
img {
    width: 2.5em;
    height: 3em;
    opacity: 0;
    /*0 -> img is fully opaque*/
    transition: opacity 4s;
}

/* positioning the numbers of each calendar door */
.day {
    position: absolute;
}

/* creating the layout and overall style of the calendar */
#container1,
#container2,
#container3,
#container4 {
    display: flex;
    width: 90%;
    height: 100%;
    margin: 3% auto;
    flex-direction: row;
    justify-content: space-evenly;
    align-items: center;
}

section div {
    width: 200px;
    height: 200px;
    display: flex;
    background-color: #23856d;
    border: 3px solid #006f57;
    box-shadow: inset -0.3em -0.3em 0.6em rgba(0, 0, 0, 0.4), inset 0.3em 0.3em 0.6em rgba(0, 0, 0,
0.4);
    color: white;
    margin-left: 3px;
    margin-right: 3px;
    border-radius: 5%;
    margin-top: 2%;
    font-size: 4em;
    justify-content: center;
    align-items: center;
    transition: all 3s;
}

/* creating the hover effect */
section div:hover {
    color: transparent;
    box-shadow: none;
    background-color: #ffffff;
    transform: rotate(360deg);
}

img:hover {
    opacity: 1;
}

/* media queries */
@media only screen and (max-width: 480px) {

    section div {
```

```
        width: 60px;
        height: 60px;
        font-size: 2em;
        margin-left: 2px;
        margin-right: 2px;
    }

    h1 {
        font-size: 3em;
    }

    img {
        width: 0.5em;
        height: 1em;
    }
}

@media only screen and (min-width:481px) and (max-width: 768px) {

    section div {
        width: 100px;
        height: 100px;
        font-size: 2em;
        margin-left: 2px;
        margin-right: 2px;
    }

    h1 {
        font-size: 4em;
    }

    img {
        width: 1em;
        height: 1.5em;
    }
}

@media only screen and (min-width:769px) and (max-width: 1024px) {

    section div {
        width: 170px;
        height: 170px;
        font-size: 3em;
        margin-left: 2px;
        margin-right: 2px;
    }

    img {
        width: 1.5em;
        height: 2em;
    }
}
```

Code 32 - The finished style sheet for chapter 5.2