Cascading Style Sheets
CSS

Nathan Warner

Northern Illinois
University

Computer Science
Northern Illinois University
September 19, 2023
United States

Contents

1 Background Images

```
body {
    background-image: url(link or path);
}
```

And we have a couple properties we can use:

- **background-repeat:** Specifies how the background images are repeated. Possible values include:
  - repeat
  - repeat x
  - repeat y
  - **no-repeat**
- **background-size:** Specifies the size of the background images
  - auto
  - cover
  - contain (used with values such as px or %s)
- **background-position:** (x,y) Specifies the starting position of the background images.
  - left
  - right
  - center
  - top
  - bottom
  - center
- **background-attachment:** Specifies if the background image should scroll with the content or be fixed during scrolling.
  - **scroll:** The background will scroll along with the element.
  - **fixed:** The background is fixed relative to the viewport
  - **local:** The background will scroll along with the element's content.
- **background-orgin:** Specifies the positioning area of the background images.
  - **padding-box**: Default. The background is relative to the padding box.
  - **border-box**: The background is relative to the border box.
  - **content-box**: The background is relative to the content box.
- **background-clip:** Determines how far the background images and color extend within the element.
  - **border-box**: Default. The background extends to the outside edge of the border.
  - **padding-box**: The background extends to the outside edge of the padding.
  - **content-box**: The background extends to the edge of the content box.

2 Syntax

A CSS comprises of style rules that are interpreted by the browser and then applied to the corresponding elements in your document. A style rule is made of three parts:

- **Selector:** A selector is an HTML tag at which a style will be applied. This could be any tag like <h1> or <table> etc.
- **Property:** A property is a type of attribute of HTML tag. Put simply, all the HTML attributes are converted into CSS properties. They could be color, border, etc.
- **Value:** Values are assigned to properties. For example, color property can have the value either red or #F1F1F1 etc.

You can put CSS Style Rule Syntax as follows:

```
selector { property: value }
```

2.1 The Type Selectors

This is the same selector we have seen above. Again, one more example to give a color to all level 1 headings:

```
h1 {
 color: #36CFFF;
}
```

2.2 The Universal Selectors

Rather than selecting elements of a specific type, the universal selector quite simply matches the name of any element type:

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

2.3 The Descendant Selectors

Suppose you want to apply a style rule to a particular element only when it lies inside a particular element. As given in the following example, the style rule will apply to element only when it lies inside the <ul> tag.

```
ul em {
 color: #000000;
}
```

2.4 The class selector

You can define style rules based on the class attribute of the elements. All the elements having that class will be formatted according to the defined rule.

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

This rule renders the content in black for every element with class attribute set to black in our document. You can make it a bit more particular. For example:

```
h1.black {
 color: #000000;
}
```

You can apply more than one class selectors to a given element. Consider the following example:

```
<p class="center bold">
This para will be styled by the classes center and bold.
</p>
```

2.5 The ID Selectors

You can define style rules based on the id attribute of the elements. All the elements having that id will be formatted according to the defined rule.

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

This rule renders the content in black for every element with id attribute set to black in our document. You can make it a bit more particular. For example:

```
h1#black {
 color: #000000;
}
```

The true power of id selectors is when they are used as the foundation for descendant selectors. For example:

```
#black h2 {
 color: #000000;
}
```

In this example, all level 2 headings will be displayed in black color when those headings will lie within tags having id attribute set to black.

2.6 The Child Selectors

You have seen the descendant selectors. There is one more type of selector, which is very similar to descendants but have different functionality. Consider the following example:

```
body > p {
 color: #000000;
}
```

This rule will render all the paragraphs in black if they are a direct child of the <body> element. Other paragraphs put inside other elements like or <td> would not have any effect of this rule.

2.7 The Attribute Selectors

You can also apply styles to HTML elements with particular attributes. The style rule below will match all the input elements having a type attribute with a value of text:

```
input[type="text"]{
 color: #000000;
}
```

2.8 Multiple Style Rules

You may need to define multiple style rules for a single element. You can define these rules to combine multiple properties and corresponding values into a single block as defined in the following example:

```
h1 {
    color: #36C;
    font-weight: normal;
    letter-spacing: .4em;
    margin-bottom: 1em;
    text-transform: lowercase;
}
```

2.9 Grouping Selectors

You can apply a style to many selectors if you like. Just separate the selectors with a comma, as given in the following example:

```
h1, h2, h3 {
    color: #36C;
    font-weight: normal;
    letter-spacing: .4em;
    margin-bottom: 1em;
    text-transform: lowercase;
}
```

3 Inclusion

There are four ways to associate styles with your HTML document. Most commonly used methods are inline CSS and External CSS.

3.1 Embedded CSS -The <style> Element

You can put your CSS rules into an HTML document using the <style> element. This tag is placed inside the <head>...</head> tags. Rules defined using this syntax will be applied to all the elements available in the document. Here is the generic syntax:

```
<style type="text/css" media="...">
Style Rules
.............
</style>
</head>
```

3.2 Inline CSS -The style Attribute

You can use style attribute of any HTML element to define style rules. These rules will be applied to that element only. Here is the generic syntax:

```
<element style="...style rules....">
```

3.3 Imported CSS -@import Rule

@import is used to import an external stylesheet in a manner similar to the <link> element. Here is the generic syntax of @import rule.

```
<head>
    <@import "URL / filename.css";
</head>

// or
<head>
    <@import url("URL");
</head>
```

3.4 Linking css to html file

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

4 Measuring Units

- % Defines a measurement as a percentage relative to another value, typically an enclosing element: p font-size: 16pt; **line-height**: 125%;
- cm Defines a measurement in centimeters.
- div margin-bottom: 2cm;
- em A relative measurement for the height of a font in em spaces. Because an em unit is equivalent to the size of a given font, if you assign a font to 12pt, each ”em” unit would be 12pt; thus, 2em would be 24pt.: p {letter-spacing: 7em;}
- ex This value defines a measurement relative to a font's **x-height**. The xheight is determined by the height of the font's lowercase letter x.: p {font-size: 24pt; line-height:3ex;}
- in Defines a measurement in inches. p {word-spacing: .15in;}
- mm Defines a measurement in millimeters.
- p {word-spacing: 15mm;}
- pc Defines a measurement in picas. A pica is equivalent to 12 points; thus, there are 6 picas per inch.: p {font-size: 20pc;}
- pt Defines a measurement in points. A point is defined as 1/72nd of an inch.: body {font-size: 18pt;}
- px Defines a measurement in screen pixels.: p {padding: 25px;}

5 Text

This chapter teaches you how to manipulate text using CSS properties. You can set the following text properties of an element:

Fonts

- **font-family:** Specifies the typeface to be used. (Given by font name)
- **font-size:** Sets the size of the font. (Given in units)
- **font-weight:** Sets the weight (or thickness) of the font characters (e.g., bold, normal, 100, 200, ..., 900).
  - Normal
  - Bold
  - Bolder
  - Lighter
  - Or numeric values
- **font-style:** Sets the style of the font (normal, italic, oblique).
  - Normal
  - Italic
  - Oblique
- **font-variant:** Selects a normal, or **small-caps** face from a font family.
  - Normal
  - **Small-caps**
- **line-height:** Sets the amount of space above and below inline elements.
  - Numbers
  - Numbers with units

Other:

- **Color:** used to set the color of a text.
- **Direction:** used to set the text direction.
  - ltr
  - rtl
- **Letter-spacing:** used to add or subtract space between the letters that make up a word.
- **Word-spacing:** used to add or subtract space between the words of a sentence.
- **Text-indent:** used to indent the text of a paragraph.
- **text-align:** used to align the text of a document.
  - left
  - right
  - center
  - justify
- **Text-decoration:** used to underline, overline, and strikethrough text.
  - underline

  - overline
  - **line-through**
  - none
  - **Styles:** ‘solid;‘, ‘wavy;‘, ‘dotted;‘, ‘dashed;‘. (given to ones above)
  - **Colors:** ‘red;‘, ‘blue;‘ (or any valid color value) (given to ones above)

- **text-transform:** used to capitalize text or convert text to uppercase or lowercase letters.
  - capitalize
  - uppercase
  - lowercase

- **text-shadow:** used to set the text shadow around a text.
  - **Eg:** 1px 1px 1px red; (horizontal-offset **vertical-offset** blur-radius color)

6 Images

6.1 Dimensions

- **width:** Specifies the width of the image.
- **Height:** Specifies the height of the image.
- **min-width:** Specifies the minimum width of the image.
- **min-height:** Specifies the minimum height of the image.
- **max-width:** Specifies the maximum width of the image.
- **max-height:** Specifies the maximum height of the image.

6.2 Layout Properties

- **margin:** Controls the space outside the image.
  - top, right, bottom, left
- **padding:** Although not commonly used with inline images, padding can be used when the image is inside a block container.
  - top, right, bottom, left
- **display:** Defines how the image is displayed (e.g., block, inline, **inline-block**).
  - block
  - inline
  - **inline-block**

6.3 Positioning

- **Position:** Determines an element's positioning method.
  - **Static:** Default position.
  - **Relative:** Positioned relative to its normal position.
  - **Absolute:** Positioned relative to its closest positioned ancestor.
  - **Sticky:** Positioned based on the user's scroll.
  - **Fixed:** Positioned relative to the viewport.
- **top, bottom, left, right:** Determines the position of an element, works with **non-static** positioned elements.
- **clip-path:** Clips an element to a specific shape.
  - **circle:** Clips to a circle shape.
  - **ellipse:** Clips to an ellipse shape.
  - **inset:** Clips to a rectangle.
  - **polygon:** Clips to a custom polygonal shape.
- **object-fit:** Defines how an element, like an image, should fit its container.
  - **cover:** Scales the image to cover the container.

  - **fill:** Fills image within container
  - **contain:** Scales the image to fit within the container.
- **object-position:** Positions the image within its container.
  - **left, right, center:** Positions horizontally.
  - **top, bottom:** Positions vertically.

6.4 Filters

- **Filter:** Applies graphical effects to an element.
  - **blur:** Blurs the content.
  - **brightness:** Adjusts the brightness.
  - **contrast:** Adjusts the contrast.
  - **grayscale:** Converts to grayscale.
- **opacity:** Sets the transparency level of an element.

7 Pseudo classes

Concept 1: Pseudo-classes in CSS are used to define special states of an element that cannot be targeted using simple selectors. They are prefixed with a colon :. Here's an overview of some of the most commonly used pseudo-classes:

- **:hover** - Represents an element that is being hovered over by the mouse cursor. This is frequently used to indicate interactive elements or to provide visual feedback.
- **:active** - Represents an element (like a button) that is being activated (e.g., while it's being clicked). It's often used in combination with **:hover** for interactive elements.
- **:focus** - Represents an element that has received focus, often due to being clicked on or navigated to via the keyboard. It's crucial for accessibility.
- **:first-child** & **:last-child** - Represent the first and last child of a parent, respectively. These are handy for styling lists, navigation menus, and other grouped elements.
- **:nth-child(n)** - Used to target elements based on their position in a group or the overall set of siblings. It's versatile and can be used to style alternating rows in tables, for example.
- **:not(selector)** - Represents elements that do not match the given selector. It's useful for excluding specific elements from a general style rule.
- **:checked** - Represents elements like radio buttons or checkboxes that are in a checked state. This is especially useful in creating custom styled form elements using only CSS.
- **:disabled** - Represents form elements that are currently disabled. It's important for indicating **non-interactive** controls.
- **:valid** & **:invalid** - Represent form elements that have valid or invalid contents, respectively. They're helpful for **real-time** form validation feedback.
- **:required** - Represents form elements with the 'required' attribute, useful for styling mandatory fields.

8 Links

- The :link signifies unvisited hyperlinks.
- The :visited signifies visited hyperlinks.
- The :hover signifies an element that currently has the user's mouse pointer hovering over it.
- The :active signifies an element on which the user is currently clicking.
- The :focus targets a link when it has keyboard input focus, such as when navigated to via the Tab key.

9 Borders

- The **border-width** specifies the width of a border.
  - light
  - medium
  - thick
- The **border-style** specifies whether a border should be solid, dashed line, double line, or one of the other possible values.
  - solid
  - dotted
  - dashed
  - double
  - groove
  - ridge
  - inset
  - outset
  - hidden
- **border-bottom-style** changes the style of bottom border.
- **border-top-style** changes the style of top border.
- **border-left-style** changes the style of left border.
- **border-right-style** changes the style of right border.
- The **border-color** specifies the color of a border.
- **border-left-color** changes the color of left border.
- **border-right-color** changes the color of right border.
- **border-top-color** changes the color of top border.
- **border-bottom-color** changes the color of bottom border.
- **border-radius**: Used to round the corners of an element
- **border-top-radius**: Used to round the corners of an element
- **border-right-radius**: Used to round the corners of an element
- **border-bottom-radius**: Used to round the corners of an element
- **border-left-radius**: Used to round the corners of an element
- **Border-image-source**

9.1 The border-style Property

The **border-style** property allows you to select one of the following styles of border:

- **none:** No border. (Equivalent of **border-width**:0;)
- **solid:** Border is a single solid line.
- **dotted:** Border is a series of dots.
- **dashed:** Border is a series of short lines.
- **double:** Border is two solid lines.
- **groove:** Border looks as though it is carved into the page.
- **ridge:** Border looks the opposite of groove.
- **inset:** Border makes the box look like it is embedded in the page.
- **outset:** Border makes the box look like it is coming out of the canvas.
- **hidden:** Same as none, except in terms of **border-conflict** resolution for table elements.

You can individually change the style of the bottom, left, top, and right borders of an element using the following properties:

- **border-bottom-style** changes the style of bottom border.
- **border-top-style** changes the style of top border.
- **border-left-style** changes the style of left border.
- **border-right-style** changes the style of right border.

9.2 The border-width Property

- **border-bottom-width** changes the width of bottom border.
- **border-top-width** changes the width of top border.
- **border-left-width** changes the width of left border.
- **border-right-width** changes the width of right border.

9.3 Border Properties Using Shorthand

```
1  <p style="border:4px solid red;">
2      This example is showing shorthand property for border.
3  </p>
```

10 Margins

The **margin** is the space outside an element, separating it from other elements. The **margin** properties are used to create space around elements, outside of any defined borders.

- The **margin** specifies a shorthand property for setting the margin properties in one declaration.
  - top right bottom left
- The **margin-bottom** specifies the bottom margin of an element.
- The **margin-top** specifies the top margin of an element.
- The **margin-left** specifies the left margin of an element.
- The **margin-right** specifies the right margin of an element.

11 Lists

- The **list-style-type** allows you to control the shape or appearance of the marker.

  UL

  - **disc (default)**
  - **circle**
  - **square**
  - **none**

  OL

  - **decimal (default)**
  - **decimal-leading-zero**
  - **lower-roman**
  - **upper-roman**
  - **lower-alpha or lower-latin**
  - **upper-alpha or upper-latin**
  - **none**

- The **list-style-position** specifies whether a long point that wraps to a second line should align with the first line or start underneath the start of the marker.
  - **outside**
  - **inside**
- The **list-style-image** specifies an image for the marker rather than a bullet point or number.
- The **list-style** serves as shorthand for the preceding properties.
  - list-style-type list-style-position list-style-image
- The **marker-offset** specifies the distance between a marker and the text in the list.

12 Paddings

Padding refers to the space between the content of an element and its border

- The **padding-top** specifies the top padding of an element.
- The **padding-left** specifies the left padding of an element.
- The **padding-right** specifies the right padding of an element.
- The padding serves as shorthand for the preceding properties.
- The **padding-bottom** specifies the bottom padding of an element.
  - top right bottom left

13 Outlines

Outlines are very similar to borders, but there are few major differences as well:

- An outline does not take up space.
- Outlines do not have to be rectangular.
- Outline is always the same on all sides; you cannot specify different values for different sides of an element.

You can set the following outline properties using CSS.

- The **outline-width** property is used to set the width of the outline.
  - **thin**
  - **medium**
  - **thick**
  - **numeric**
- The **outline-style** property is used to set the line style for the outline.
  - **solid**
  - **dotted**
  - **dashed**
  - **double**
  - **groove**
  - **ridge**
  - **inset**
  - **outset**
  - **hidden**
- The **outline-color** property is used to set the color of the outline.
- The **outline-offset** property specifies the amount of space between an outline and the edge or border of an element.
- The outline property is used to set all the above three properties in a single statement.

14 Dimensions

You have seen the border that surrounds every box i.e. element, the padding that can appear inside each box, and the margin that can go around them. In this chapter, we will learn how to change the dimensions of boxes.

We have the following properties that allow you to control the dimensions of a box.

- The **height** property is used to set the height of a box.
- The **width** property is used to set the width of a box.
- The **line-height** property is used to set the height of a line of text.
- The **max-height** property is used to set a maximum height that a box can be.
- The **min-height** property is used to set the minimum height that a box can be.
- The **max-width** property is used to set the maximum width that a box can be.
- The **min-width** property is used to set the minimum width that a box can be
- The **box-sizing** property Determines how the total width and height of an element is calculated. The default value is content-box, where width and height only apply to the content, excluding padding and borders. If set to border-box, the width and height include the content, padding, and borders.
  - **content-box** where width and height only apply to the content, excluding padding and borders. (default)
  - **border-box** the width and height include the content, padding, and borders.
- The **Display** property affects how elements occupy space.
  - **inline**: takes up as much space as necessary
  - **block** take up the full available width

15 Scrollbars

There may be a case when an element's content might be larger than the amount of space allocated to it. For example, the given width and height properties do not allow enough room to accommodate the content of the element.

CSS provides a property called overflow, which tells the browser what to do if the box's contents is larger than the box itself. This property can take one of the following values:

- **Overflow**: tells the browser what to do if the box's contents is larger than the box itself.
  - **visible** Allows the content to overflow the borders of its containing element.
  - **hidden** The content of the nested element is simply cut off at the border of the containing element and no scrollbars is visible.
  - **scroll** The size of the containing element does not change, but the scrollbars are added to allow the user to scroll to see the content.
  - **Auto** The purpose is the same as scroll, but the scrollbar will be shown only if the content does overflow.
- **overflow-x**: Controls the behavior of content overflow in the horizontal direction
- **overflow-y**: Controls the behavior of content overflow in the vertical direction

16 Visibility

A property called visibility allows you to hide an element from view. You can use this property along with JavaScript to create very complex menu and very complex webpage layouts.

You may choose to use the visibility property to hide error messages that are only displayed if the user needs to see them, or to hide answers to a quiz until the user selects an option.

The visibility property can take the values listed in the table that follows:

visibility:

- visible The box and its contents are shown to the user.
- hidden The box and its content are made invisible, although they still affect the layout of the page.
- collapse This is for use only with dynamic table columns and row effects

We also have **Display**: this is useful when we want to hide something but not take up any space

Display

- **none**: does not take up any space

17 Positioning

CSS helps you to position your HTML element. You can put any HTML element at whatever location you like. You can specify whether you want the element positioned relative to its natural position in the page or absolute based on its parent element.

Now, we will see all the CSS positioning related properties with examples.

17.1 Relative Positioning

Relative positioning changes the position of the HTML element relative to where it normally appears. So ”left:20” adds 20 pixels to the element’s LEFT position.

You can use two values top and left along with the position property to move an HTML element anywhere in an HTML document.

- Move Left - Use a negative value for left.
- Move Right - Use a positive value for left.
- Move Up - Use a negative value for top.
- Move Down - Use a positive value for top.

Note:-

You can use the bottom or right values as well in the same way as top and left.

17.2 Absolute Positioning

An element with position: absolute is positioned at the specified coordinates relative to your screen **top-left** corner.

You can use two values top and left along with the position property to move an HTML element anywhere in HTML document.

- Move Left - Use a negative value for left.
- Move Right - Use a positive value for left.
- Move Up - Use a negative value for top.
- Move Down - Use a positive value for top.

Note:-

You can use bottom or right values as well in the same way as top and left.

17.3 Fixed Positioning

Fixed positioning allows you to fix the position of an element to a particular spot on the page, regardless of scrolling. Specified coordinates will be relative to the browser window.

You can use two values top and left along with the position property to move an HTML element anywhere in the HTML document.

- Move Left - Use a negative value for left.
- Move Right - Use a positive value for left.
- Move Up - Use a negative value for top.
- Move Down - Use a positive value for top

Note:-

You can use bottom or right values as well in the same way as top and left.

18 Pseudo-elements

Pseudo-elements in CSS are used to style specific parts of an element that you can't easily target with regular CSS selectors. They allow you to insert and style content in an element without changing the actual HTML.

- **::before**: Used to insert content before the content of an element.
- **::after**: Used to insert content after the content of an element.
- **::first-line**: Targets only the first line of an element. It's commonly used in typography to style the first line differently from the rest of the content.
- **::first-letter**: Targets the first letter of a block-level element. This is often used for "drop caps" effects in editorial design.
- **::selection**: Used to change the appearance of selected text in an element.
- **::placeholder**: Targets the placeholder text in form elements like <input> and <textarea>.
- **::marker**: Used to style the marker of list items (li) in ordered (ol) and unordered (ul) lists.

19 Z-index and stacking

The **z-index** property in CSS determines the stack order of positioned elements (those with position values other than static). An element with a higher z-index will generally appear on top of an element with a lower one.

20 Display Styles

In css we have couple different display styles:

- inline
- block
- inline-block
- flex
- grid

For now we will be discussing the simpler ones, which are the default for different kinds of elements. These are *inline, block* and *inline-block*

Inline

This display style will force the element to take up only the space that it needs, and other elements can appear on the same line.

The elements that default to a inline display style are:

- links and spans

Along with a few others

Block

This display style will force a newline a newline above and below, and take up the entire width of the page by default.

The elements that default to a block display style are:

- Headers
- divs
- sections
- preserved text
- lists and list items
- blockquotes

Along with a few others.

inline-block

Inline-block behaves the same way as inline, where it will attempt to take up the least space possible. However, we are allowed to change the width and height as needed.

Note:-

We cannot change the height and width of *inline* displays, we must convert to either *inline-block* or block

21 Flex box

When you declare display: flex or display: inline-flex on an element, it becomes a flex container. Its direct children automatically become flex items.

21.1 Main Axis and Cross Axis

The main axis is the primary axis along which flex items are laid out. It's determined by the flex-direction property. The cross axis is perpendicular to the main axis.

flex-direction

- **flex-direction**: row; (default): main axis is horizontal.
- **flex-direction**: column;: main axis is vertical.

21.2 Properties for the Flex Container

- **flex-direction**: Determines the direction of the main axis.
  - **row (default)**
  - **row-reverse**
  - **column**
  - **column-reverse**
- **flex-wrap:** By default, flex items will all try to fit onto one line. You can change that with this property.
  - **nowrap (default)**
  - **wrap**
  - **wrap-reverse**
- **justify-content: Aligns items along the main axis.**
  - **flex-start (default)**
  - **flex-end**
  - **center**
  - **space-between**
  - **space-around**
  - **space-evenly**
- **align-items:** Aligns items along the cross axis.
  - **stretch (default)**
  - **flex-start**
  - **flex-end**
  - **center**
  - **baseline**
- **align-content:** Aligns lines of items when there's extra space on the cross-axis.
  - **stretch (default)**
  - **flex-start**
  - **flex-end**
  - **center**
  - **space-between**
  - **space-around**

21.3 Properties for the Flex Items

- **order:** By default, all items have an order of 0. You can use positive or negative integers to change the order of individual items.
- **flex-grow:** Determines the grow factor of an item. By default, it's 0.
- **flex-shrink:** Determines the shrink factor of an item. By default, it's 1.
- **flex-basis:** Defines the default size of an item. Default value is auto.
- **flex:** This is a shorthand for flex-grow, flex-shrink, and flex-basis. The default is 0 1 auto.
- **align-self:** This allows the default alignment (or the one set by align-items) to be overridden for individual flex items.
  - **auto**
  - **flex-start**
  - **flex-end**
  - **center**
  - **baseline**
  - **stretch**

Note:-

Making a parent into a flex box automatically makes its children flex boxes

22 Grid display

The grid display style is similar to the flex display style, except that grids layout their items in 2 dimensions. Whereas flexbox only uses a single dimension. By specifying *display: grid;* we can active the grid display style.

22.1 Defining Grid Columns and Rows

- **grid-template-columns**: Ex: 200px 200px 200px (this gives three columns of length 200px)
- **grid-template-rows**: Ex: 200px 200px 200px (This gives three rows of length 200px)

22.2 Using repeat function

We can use the repeat function, ex:

```
display: grid;
grid-template-columns: repeat(4, 100px);
```

22.3 Automatically define column and row sizes

- **grid-auto-rows**: Ex: grid-auto-rows: 150px; (Any rows that are not defined will be 150px)
- **grid-auto-columns**

22.4 Using the minmax function

For the auto-rows and auto-columns property, we can use the minmax function to specify what we want the minimum and maximum size

Example:

```
display: grid;
grid-auto-rows: minmax(100px, auto);
```

22.5 Gap between Grid Items

- **grid-row-gap:** Defines the size of the gap between grid rows.
- **grid-column-gap**: Defines the size of the gap between grid columns.
- **grid-gap:** Shorthand for row-gap and column-gap.

22.6 Grid template areas

- **grid-template-areas: Assigns names to specific areas of the grid layout.**
- **grid-area: References a name given in grid-template-areas.**

Example:

```
.grid-container{
    display: grid;
    grid-template-columns: repeat(2, 200px);
    grid-auto-rows: minmax(auto, 150px);
    grid-template-areas:
    "name1 name1"
    "name2 name3"
}

.grid-item-1 {
    grid-area: name1;
}

.grid-item-2 {
    grid-area: name2;
}

.grid-item-3 {
    grid-area: name3;
}
```

22.7 Placing Items

The following properties define where an item should start and end.

- **grid-column-start**
- **grid-column-end**
- **grid-row-start**
- **grid-row-end**
- **grid-column**: shorthand (start end) (Ex: 1 / -1) (Ex: span 3)

Note:-

we can use a value of -1 for the end to signify that is should take up the entire row. We can also use span

22.8 The fraction unit (fr)

The fr unit represents a fraction of the available space in the grid container. For example, if you define grid-template-columns: 1fr 2fr;, the first column will take up one-third and the second will take up two-thirds of the available space.

22.9 Grid alignment

To align our grid containers, we have:

- **justify-content**: This property aligns the entire grid (i.e., all the items together) along the row (inline) axis
  - start
  - end
  - center
  - stretch
  - space-around
  - space-between
  - space-evenly
- **align-content**: This property aligns the entire grid along the column (block) axis
  - **start**
  - **end**
  - **center**
  - **stretch**
  - **space-around**
  - **space-between**
  - **space-evenly**

To align the grid items, we have:

- **justify-items**: This property aligns the grid items along the row (inline) axis inside their respective grid areas.
  - stretch
  - start
  - end
  - center
- **align-items**: This property aligns the grid items along the column (block) axis inside their respective grid areas.
  - stretch
  - start
  - end
  - center

Note:-

Must specify height and width for align-content

23 Transparency and acrylic transparency

To achieve transparency in an element, one common method is to adjust its opacity property, for instance, setting it to 50%. However, a notable drawback of this approach is that it affects not just the element itself but also all of its child elements, making them equally transparent. A more effective solution is to define the element's background color using the RGBA color model. This method involves specifying the Red, Green, and Blue (RGB) values and adding an Alpha channel, which controls the transparency level independently, without altering the transparency of the child elements. **Concept 2:**

```
background-color: rgba(106, 106, 106, 0.5); /* blueish with 50% opacity */
```

Then, to achieve an acrylic look, we can specify the 'backdrop-filter' property with a 'blur' value.

```
backdrop-filter: blur(10px); /* Apply a blur effect to the background */
```

24 Subsequent-sibling combinator

Concept 3: The subsequent-sibling combinator is made of the "tilde" character that separates two sequences of simple selectors. The elements represented by the two sequences share the same parent in the document tree and the element represented by the first sequence precedes (not necessarily immediately) the element represented by the second one.

Consider the following example:

```
 <ul>
  <li class="b">1st</li>
  <li class="a">2nd</li>
  <li>3rd</li>
  <li class="b">4th</li>
  <li class="b">5th</li>
</ul>
```

```
.a ~ .b {
  background-color: powderblue;
}
```

`.a ~ .b` matches the 4th and 5th list item because they:

- Are `.b` elements
- Are siblings of `.a`
- Appear after `.a` in HTML source order.

25 Responsive Design

Concept 4: Responsive design in frontend web development is a strategy aimed at creating websites that provide an optimal viewing and interaction experience across a wide range of devices, from desktop computers to mobile phones. The goal of responsive design is to ensure that a website is easily navigable, readable, and functional regardless of the device's screen size, orientation, or resolution. This approach is fundamental in modern web development due to the diverse array of devices used to access the internet.

A major aspect of responsive design is ensuring that our webpages do not break when screen size is changed. For example a webpage that is created and looks perfect on a 1920x1080 desktop monitor might completely break when the web browsers size is changed or when the webpage is viewed on a smart device such as a cellphone or tablet.

25.1 Html meta tag

The first and simplest way to integrate responsiveness into our webpages is including a simple HTML meta tag in our documents.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

25.2 @media

Concept 5: The @media tag in css allows us to write css that only applies to certain device sizes

```

@media only screen and (max-width: 600px) {
    // targets devices with a screen width of 600 pixels or less, often used for
    ↪ smartphones.
}

@media only screen and (min-width: 786px) {
    // targets screens with a minimum width of 786 pixels, typically used for
    ↪ tablets
}

@media only screen and (min-width: 786px) and (min-width 1024px) {
    // targets screens with a width between 786 and 1024 pixels  which typically
    ↪ includes tablets in both portrait and landscape modes.
}


@media only screen and (orientation: portrait) {
    targets devices that are in portrait mode
}

@media only screen and (orientation: landscape) {
    targets devices that are in portrait mode
}
```

25.3 Hover capabilities

Concept 6: With the increasing number of touch devices, you can target devices based on whether they are capable of hover interactions

```
@media (hover: hover) {
    targets devices that support hover functionality.
}
```

25.4 Targeting Smartphones

The media query we use to target smart phones such as IPhones is

```
/* Portrait smartphones */
@media only screen and (max-width: 600px) and (orientation: portrait) {
    /* Styles */
}

/* Landscape smartphones */
@media only screen and (max-width: 600px) and (orientation: landscape) {
    /* Styles */
}
```

25.5 Targeting Tablets

```
/* Portrait tablets */
@media only screen and (min-width: 601px) and (max-width: 1024px) and
↪  (orientation: portrait) {
    /* Styles */
}

/* Landscape tablets */
@media only screen and (min-width: 601px) and (max-width: 1024px) and
↪  (orientation: landscape) {
    /* Styles */
}
```

25.6 Desktop browser size changes

One common problem one will experience when building webpages is that, when uses increase or decrease the size of their web browser, the page may break

Designing a webpage for desktop use while accommodating users who resize their browser window is an important aspect of responsive web design. The goal here is to ensure that your website remains functional and visually appealing across a range of window sizes. This can be achieved through a combination of fluid layouts, flexible images, and media queries.

The next couple subsections will focus on this problem

25.7 Fluid Layouts

Fluid layouts use relative units like percentages or vw / vh rather than fixed units like pixels for widths. This makes your layout more flexible and able to adapt to different screen sizes.

```
.container {
    width: 80%;
    margin: auto;
}
```

In this example, the container will always be 80% of the viewport width, regardless of the screen size.

25.8 Flexible Images

Images should also be able to scale within their containing elements. This can be done by setting their max-width to 100% and height to auto.

```
img {
    max-width: 100%;
    height: auto;
}
```

25.9 Media queries

Media queries can also be used to adapt to changes in the size of the **window**, but not necessarily size changes in the actual web browser itself.

25.10 Responsive font sizing

You can use vw in combination with clamp(). The clamp() function enables you to set a minimum size, preferred size, and maximum size. This way, your font size will scale with the viewport but won't get too small or too large.

```
font-size: clamp(16px, 2vw, 24px);
```

25.11 Background-image across viewports

```
body {
    background: url("./figures/back.jpeg") no-repeat center center fixed;
    background-size: cover;
    margin: 0;
    padding: 0;
    height: 100vh;
    width: 100vw;
    /* overflow: hidden; Maybe we do want a scrollable webpage */
}
```