Webcoding Paper

a cheat sheet

max

2024-12-13

Contents

The Word Wide Web

The World Wide Web (WWW), commonly known as the Web, is an information system where documents and other web resources are identified by Uniform Resource Locators (URLs, such as https://example.com/), which may be interlinked by hyperlinks, and are accessible over the Internet. The resources of the Web are transferred via the Hypertext Transfer Protocol (HTTP), may be accessed by users by a software application called a web browser, and are published by a software application called a web server. The World Wide Web is not synonymous with the Internet, which predated the Web in some form by over two decades and upon which technologies the Web is built.″

- invented by Tim Berners-Lee in 1989
- Browser wars
- W3C and WHATWG conflicts:
  - The had been publishing competing standards since 2012
  - W3C wanted to publish a ″finished″ version of HTML5
  - WHATWG wanted to continue working on a living standard for HTML

  - May 2019: 2 entities signed an agreement to work together on a single HTML version
- W3C = World Wide Web Consortium
- WAI = Web Accessibility Initiative

Web Accessibility and WAI

> The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect."–Tim Berners-Lee

- Accessible Websites provide accommodations that help individuals to individuals with visual, auditory, physical, and neurological disabilities overcome barriers

Web Development

- Process of developing websites or webpage on the internet or intranet
- Divided into:
  - Client-Sie Coding
  - Sever-Side coding
  - Database Technology

Client-Side stack

- HTML & CSS, JavaScript, JS Libraries, JS Frameworks
- Tools: Grunt, gulp, webpack, etc.

What is HTML used for?

- **Structure**
  - Text, Lists, Links, Images, Tables, Forms, Audio, Video

What is CSS used for?

- **Presentation and Layout**

A closer looks at Tags

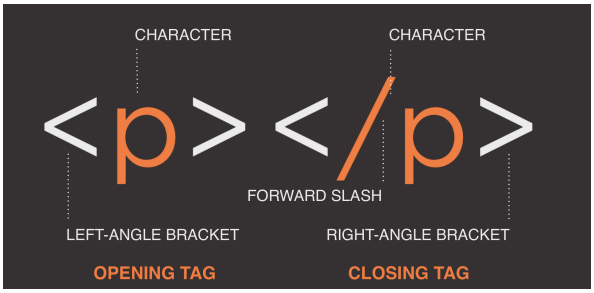


Figure 1: closer_look_at_tags.png

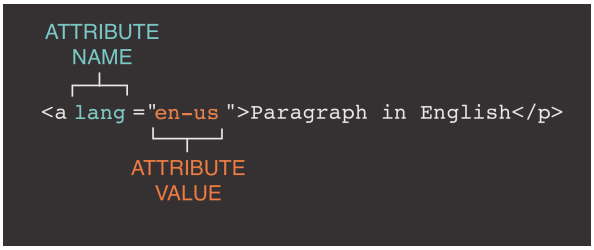


Figure 2: closer_look_at_attributes.png

Attributes tell us more about Elements

Body, Head and Title

```
<head>
  <title>This will show up in the Tab
      Name</title>
</head>
```

```
<body>
  <h1>Main Content</h1>
  <p>This is the Body</p>
</body>
```

Repeat

What is a self-contained unit of discourse in writing dealing with a particular point or idea? What HTML tag is used for this purpose?: Paragraph: `<p>`

HTML Elements between `<html></html>` →`<head>` (Metainformation), `<body>` (Contents)

How many section headings are there in html: 6 headings, (s1) 6 →s1 to s6

What is NOT the `<br>` tag used for: Making a new paragraph!!!!!

What is HTML used for: STRUCTURE THE CONTENTS (Text, Lists, Links, Images, Tables, Forms) →Giving meaning to the contents

What is CSS used for: Presentation + Layout

Opening: `<p>`closing `</p>`

Attributes:

`<a lang=de>`

Body, Head and Title

HTML Basics

Tags Describe Content

`<b></b>` Bold (not the same as `<strong>` because it gives the text a higher meaning) `<i></i>` Italic Dont use HTML to alter the presentation, use CSS for that. So dont use `<b>` or `<i>`.

Creating a page involves adding tags to content. These Tags are known as "markup" .

HTML Elements

Structural

- e.g Headings `<h1>` to `<h6>`
- Defining the main structure of the page
- (Use css to modify the statics)
- Paragraphs `<p>`: A paragraph consist of one or more sentences that form a self-contained unit of discourse. The Start of a paragraph is indicated by a new line. Text is easier to understand when it' s split up into units of texts. For example, a book may have chapters. Chapters can have subheadings. Under each heading will be one or more paragraphs.

Semantic

What is it? $ \ $

- Set of elemts, for example:
  - `<em>` tag adds emphasis
  - `<blockquote>` contains a quote

Why use it

- Provides extra information about your content
- Do NOT use it to alter presentation if those elements

Application

- Screen readers can add emphasis to words in `<em>`
- Search engines can find quotations in `<blockquote>`

Tags:

Strong and Emphasis

- `<strong>Beware</strong>` or `<em>think</em>`

Blockquote

```
<blockquote cite="https://example.com>
<p></p>
</blockquote>
```

The url: A URL that designates a source document or message for the information quoted. This attribute is intended to point to information explaining the context or the reference for the quote.

Inline quote:

```
<p>He said <q>lol</q>.</p>
```

Quoting a specific Paragraph. Quotation marks will be automatically set: He said "lol" .

Abbreviations & Acronyms

```
<p><abbr title="Professor">Prof</abbr>
    Müller...</p>
```

There is no `<acronym>` use `<abbr>` instead

Definitions and Citations

```
<p><cite>The Book</cite> by Someone</p>
```

(After agreement): Must contain title of Work, Not Author. The persons name is not the title. So the Author should not be in `<cite>`

```
<p> A <dfn>black hole</dnf> is a region of
    space from which nothinh, note even
    light, can escape.</p>
```

The **nearest parent** of the `<dfn>` tag must also contain the definition/explanation for the term inside `<dfn>`. Inside a Paragraph we have a Definition and some Text. (Visualize the Hierarchy). That Means that the whole paragraph is the definition of what′ s inside the `<dfn></dfn>` Tag. The nears parent would be `<p>`.

Author Details

```
<address>
  <p><a ="href"
      ="x@example.com">x@exampel.com</a></p>
  <p>Some Street 57, Somewhere</p>
</address>
```

The Contact Address element, provides contact information for a person or organisation. Can have `<p>`, but not necessary. Can also have `<br>` to make new lines (for addresses/streets)

Changes to Content

```
<p> It was the
    <del>worst</del><ins>Best</ins></p>
```

To show a change. Will be crossed out THE PAIR `del` and `ins` must be used always together. `<del>` is used to signalize something has been deleted and `ins` Indicates text that has been (newly) **added** to a document. To cross something out use: `<s>crossedout</s>` (Marks text that is **no longer accurate, relevant, or valid** but is retained for historical or contextual reasons.) There is also `<u>` to underline something but dont use it! `<del>Test</del><ins>Best</ins>` = TestBest

Bold and Italic `<b></b>` Old HTML Elements. Not good Practice Draw the attention to the elements contents wich are not otherwise granted special importance. Don′ t use it to alter the "looks" of a text. If that is what′ s wanted, use CSS.

`<i></i>` Has some more meaning, e.g. Idiomatic Text: Technical terms, taxonomical designations, Translations, Thoughts (terms on other languages?)

Superscript & Subscript Higer and lower `<sub></sub>, <sup></sup>` SUBnormalSUP

Whitespace is collapsed `This eb` will become this `This eb`

Line Breaks `<br>` Forces the browser to add a line break (if you use this to modify the presentation Layer, you are using it wrong, dont use it to create different paragraphs) Useful for addresses

Horizontal Rules `<p>Topic 1</p> <hr>` →Semantik Break, other topic. Has a Semantik Meaning `<p>Topic 2</p>`

Lists

Three types

1. Ordered
   - Numbers (added automatically), The orders matters (like a recipe)
   - The order of the items **is** important

```
<ol>
  <li>First</li>
  <li>Second</li>
</ol>
```

2. Unordered `ul`
   - Like Ingredients list. Order does not matter
   - Used to create navigation menus

```
<ul>
  <li>Milk</li>
  <li>Eggs</li>
</ul>
```

3. Defintion
   - Like a list of Definitions
   - Creates a list that is used to define terms

```
<dl>
  <dt>Sashimi</dt>
  <dd>Sliced raw fish</dd>
  <dd>Similar to sushi</dd>
</dl>
```

- The Term `Sashimi` has 2 definitions (like a dictonary)
- Dont use this as a Heading

Nested lists It is possible to have lists inside lists

```
<ul>
  <li>Mousses</li>
  <li>
    Pastries
    <ul>
      <li>Croissant</li>
      <li>Palmier</li>
    </ul>
  </li>
  <li>Tarts</li>
</ul>
```

What are Flow Contents?

→Can be used inside other Contents?

Questions

How many lists are ther: 3 In HTML5, which tag is used for acronyms? `<abbr>` Whats the difference between `<strong>` and `<b>` ? Strong has an emphasis (semantik/meaning)

Lecture 3

Special Entity Characters:

Char	Code
©	&copy;
<	&lt;
>	&gt;
&	&amp;

Using escapes can make it difficult to read and maintain source code, and can also significantly increase file size.

HTML Validation

Validation

Validation Problems

Some closing tags in HTML5 are optional Some of them are:

```
<html>
<head>
<body>
<dt>
<dd>
<li>
<p>
<td>
<th>
```

```
<ul>
    <li>(</li>)
    <li>(</li>)
</ul>
```

```
<p>
...
```

Pros

- Speeds up your pages
- Size?

Cons

- Bad rwadability
- Less maintainability
- Automatic tools might not work well. For example automatic format coding

!! Close, Format, Validate !!

Comments

```
<!-- comment -->
<!--
comment 2
-->
```

ID Attribute

Used to uniquely identify one element within a page. It is used by CSS and JavaScript. `<p id="pullquote">iridn</p>`

Class Attribute

- Used to distinguish more than one element as being
- used for a specific purpose.

`<p class="imprtant admittance">igirsgn</p>`

- Can have multiple values
- CSS Classes are applied in order of class

Default display value

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

Block Elements

- starts on a new line
- takes up the full width available

Inline Elements

- Does not start on a new line
- takes up as much width as necessary
- For example: `<em>, <b>`

Grouping Elements

div

- Groups elements into a block-level element
- (Takes the whole block/space). Eg. Background colour for a paragraph →We group those with div

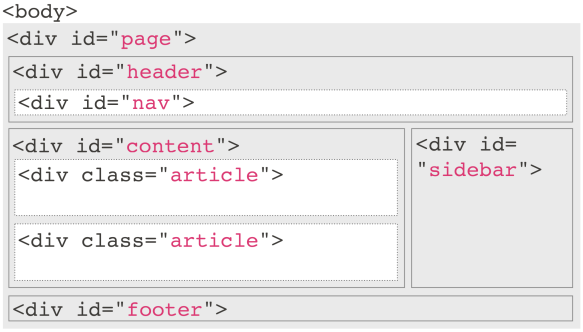
span

- Groups elements inline
- For example: When you have two words that need to be in one Group, so we can apply something to it (for example `<b>Two </b>, <b> Words</b>` →To apply the same css →`<span><b>Two</b> <b>Words</b>`)

```
<p> Some <span
    class="gallery">Something</span>
    something else.</p>
```

Structural semantic tags

`<header>, <nav>, <footer>, <main>, section, article, aside, ...`



same as `<div id="header">,...`

- Does it has to be an Article? Ask yourself if it makes sense if the object has it′ s own url →if yes it can be an article
- Header `<heaer></header>` Contains the headings
- nav element
- Main element `<main></main> contains the main content`
- Footer Element `<footer></footer>` contains the footer

- Section Element: indicates a portion or "section" of a document, like a chapter or topic
- Article Element: indicates an independent entry, like a blog posting, **that can stand on it' s own**
- Aside Element: indicates a sidebar or other tangential content
- Time Element: represents a date or time

Relative and absolute paths

images are void elements (dont need a closing tag)

Anchor Elements

- Anchor elements are denoted by the `<a>` tag, and are used to link to other pages, resources or content inside the same page
- Used for links to other sites or headings: `<a href="http://www.imdb.com">IMDB</a>`
- Adding `target="_blank"` will open it in another tab
- (Can take absolute and relative paths)
- To link to another resource on the same site: `<a href=" index.html" >Home</a>` (relative)
- Can also link to ID of html element `<a href="#top">Top</a>`
- E-Mail or telephone: `<a href="mailto:jon@example.org">Email Jon</a>` or `tel:+12435654`

Images

```
<img
    src="https://picsum.photos/id/1022"
    alt= "aurora boreal"
    title= "a picture of the northern
        lights"
>
```

- The alt text cannot be displayed
- The title gives a tooltip
- Is a void element (no closing tag, just `img`)

Cascading Style Sheet

What CSS does

Cascading Style Sheet (CSS) allows you to create rules that specify how the content od an element should appear => Control the design of web pages to make them more attractive

CSS Associates style rules

```
p {
    font-family: Arial;
}
```

`p` = selector `font-family: Arial;` = declaration

- Multiple elements can be addressed when listing them with `, →h1, h2, ... {}`
- It' s always `property: value;`

```
h1, h2, h3 {
    color: yellow;
    (property: value)
}
```

CSS Advantages

- Greater typography and page layout control
- Style is separate from structure
- Styles can be stored in a separate document and associated with many web pages
- Potentially smaller documents
- Easier site maintenance

Bringing CSS and HTML together

- Using inline styles
  - Using the `style` HTML attribute
  - Apply only to the specific element
- Using internal CSS (embedded styles)
  - Configured in the `<head>` element
  - Apply to the entire document
- Using external CSS
  - Styles are configured in a separate file
  - The `<link>` element is used at the `<head>` of the document to associate the page with the styles
- Using imported styles (`@import` directive) - not relevant for now...

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

  <head>

    <meta charset="UTF-8" />

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

    <!-- this is external css. rel is
        required, specifies the
        relationship between
    current document and the linked
    resource -->

    <link rel="stylesheet"
        href="src/style.css" />

    <!-- this is internal/embedded css -->

    <style>
            h1 {
              color: red;
            }

    </style>

  </head>

  <body>

    <!-- this is internal css, should not
        be in body -->

    <style type="text/css">
            h1 {
              color: yellow;
            }

    </style>

    <!-- this is inline css -->

    <h1 style="color: white;">Hi friend,
        try change my color!</h1>

  </body>
</html>
```

-> Text will appear white, even if other statements say otherwise

The last mentioned style is preferred. If the external and embedded CSS modify the same element, the last in order will be used.

Why use External Stylesheets?

- Same CSS can be used for every page of website
  - DRY => Don′ t repeat yourself
- No need to copy style code into each webpage
- Changes to CSS automatically apply to the entire website
- Easier to for many people to create new pages in same style

CSS Selectors

- Universal: `* { }`
- type `h1, h2, h3 {}`
- Class `.note {} p.note {}`
- ID `#introduction {}`
- Child (direct descendants/first level) `li>a {}`
- Descendent (all) `p a {}`
- Adjacent sibling (only the first `p` after every `h1` at the same level) `h1+p {}`
- General sibling (same level of hierarchy and every p sibling of h1 →everything after h1 at the same level)`h1~p {}`

Specificity

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

`*` < selector < class < ID

When I use the ID to transform an object, it will overwrite the `*`, the normal selector (for example `p`) and the class (for example `#my-class`)

Inheritance

- Some styles are inherited (e.g.: fon-family)

```
body {
  font-family: Arial, sans-serif;
```

```
  color: #665544;
  padding: 10px;
}
.page {
  border: 1px solid #665544;
  background-color: #efefef;
  padding: inherit;
}
```

CSS colour representation

```
p {
  color: red; /*color name*/
  color: #ff0000; /*hex*/
  color: #f00; /*shorthand hex*/
  color: rgb(255, 0, 0); /*decimal rgb
      triplet*/
  color: rgba(255, 0, 0, 0.5); /*decimal
      rgb + opacity from 0 to 1*/
  color: hsl(0, 100%, 50%); /*hsl color*/
}
```

Foreground color

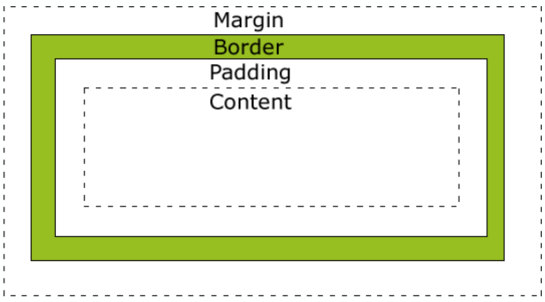
`color: DarkCyan`

Text

`background-color: DarkCyan`

Text

CSS The box Model



- Text will appear in content box
- Clear area around is padding
- border that goes around the padding and content
- followed by clear margin area

Some things we can do with boxes

- Set the dimensions of a box (width and height)
- Configure the margin, border and padding
- Style boxes:
  - Center web page contents
  - Apply shadows
  - Configure rounded corners
  - Hide/show boxes

CSS Dimensions

Unit	Purpose
px	fixed numbers of pixels
em	relative to the font size
%	percentage value of the parent element
vh/vw	relative to 1% of the viewport height/width

CSS Width and Height Properties

- width:
  - configures the width of an elements content
- min-width property:
  - Configures minimum width of an element
- max-width:
  - configures the maximum width of an element
- height
  - configures the height of an element (also has max and min height)

Important:

- When you set the width and height properties of an element with CSS, you just set the width and height of the content area. To calculate the full size of an element, you must also add padding, borders and margins. →content = height/width
- By default in the CSS box model, the width and height you assign to an element is applied only to the element′ s content box. If the element has any border or padding, this is then added to the width and height to arrive at the size of the box that′ s rendered on the screen. This means that when you set width

and height, you have to adjust the value you give to allow for any border or padding that may be added.
- The entire box (content+padding+border+margin = total height/width)
- Or setting the property `box-sizing` to `border-box`

!!! `box-sizing: border-box;`

Margin & Padding & Border

Margin

- The margin property
- Related properties: margin-top, margin-right, margin-left, margin-bottom
- Configures empty space between element and adjacent elements
- Syntax emaples: `margin: 0;`, `margin: 20px 10px;`, `margin: 10px 30px 20px;`, `margin 20px 30px 0 30px;`

Padding

- related properties: padding-top, padding-right, ...
- Configures empty space between the content of the HTML element (such as text) and the border
- Same syntax as in margin

Border

- Configures a border on the top, right, bottom, and left sides of an element
- Consists of: border-width, border-style, border-color
- Shorthand: `border: 2px solid #ff0000`
- Other styles: default, none, inset, outset, double, groove, ridge, solid, dashed, dotted

Display and Visibility

Change inline/block

```
li {
  display: inline;
}
li.coming-soon {
  display: none; /*removes it from the
      flow*/
  display: hidden; /*just makes it
      invisible*/
}
```

Styling

Centering page content with CSS

```
#container {
  margin-left: auto;
  margin-right: auto;
  width: 80%;
  /*margin: 0 auto; is the same*/
}
```

Box-shadow property

- Configure the horizontal offset, vertical offset, blur radius, and valid color value
- Example: `box-shadow: 5px 5px 5px red`

Rounded Corners

- Border-radius property
- Example: `h1 {border: 1px solid black; border-radius: 15px; }`
- `border-radius: 0 0 100% 0;` just like margin

Questions:

1. CSS treats each element as if it creates its own **BOX**
2. Whats the difference between total hight/width and hight/width? →total is everyhting (Padding, Margin, ...), width/hight alone is applied to the content alone (not the stuff arrounf)
3. Whats the difference between `display:none` and `visibility:hidden`?→`Display:none` removes the element (removes it, as it does not exists), 'visibility:hidden just does not show it (gaps would still stay)
4. Which selector would you use to only select ALL the CHILDREN sitting inside a `<div>`? →`div>* {}`

Typography Theory

Web typography refers to the use of fonts on the WWW

- "Type" is the user interface for conveying information
- Typography is about usability
- Typography is about emotion

Basics of typography

- Typeface
  - Set of symbols, numbers or letter that define the family
  - In CSS, typeface is qual to "font-family"
- Font
  - A specific instance of a typeface which includes a particular style weight => i.e. the font is the variation in weight and size of a typeface
  - Also referred as font face
  - A **font** is a specific implementation or style of a typeface, including weight, size, and style
- Example:
  - "Times" is a typeface (the font family)
  - "TimesBold" , "TimesRegular" , ...are variations of "Times" and therefore, font faces

Classification

- Serif: easier to read, with mire lines around letters
- Sans-Serif: without those extra lines (sans = without), clearer to read with lower resolution
- Monospace: used for code
- The Baseline is the line each letter sits on (like this)
- The descender is the line when for example an y ends on the lower end
- Cap Hight, is the normal height for uppercase letters
- X-Height, is the normal height for lower case letters
- Ascender, is the max upper line

Kerning

- Is the adjustment of space between two chars
- No Kerning, means that the next letter ends after the first one
- With kerning applied, the letters can overlap

Tracking

- Also referred as letter spacing
- is used to determine the (evenly) spacing between letters.
-

Leading

- is the space between baseline of the text
- Basic rule for digital: scale the line spacing as much as 120% of x-hight (single spacing)
- rule of thumb: 120%-180% is good for readability

Weight, Style and Stretch

WEIGHT	STYLE	STRETCH
Light	Normal	Condensed
Medium	*Italic*	Regular
Bold	*Oblique*	Extended
Black		

General tips

- use a maximum of 3 typefaces (heading, text, code)
- Pair fonts that belong to different classifications (sans + serif)
- Serif is better for printing and large screens
- Sans serif fonts are good for digital env
- Serif is more formal and traditional
- Sans serif tend to be more fun
- Use font size to create a visual hierarchy

Fonts and CSS

Choosing a typeface for your website

Specifying typefaces

```
body {
  font-family: "Courier New", Times, Serif;
}
```

The order is like this: Desired font →Font stack →Fallback font family

- Font family is inherited
- A browser will usually only display a font if it′ s installed on the user′ s computer

font-size

```
body {
  font-family: Georgia, Times, serif;
  font-size: 12px; /*em/%/vh...*/
}
```

- `rem` is the relative to the root html font size
- `em` is relative to it′ s parent

Own Fonts

- with this we can use fonts that are not installed on the computer

```
@font-face {
    font-family: ChunkFiveRegular;
    src: url(fonts/chunkfive.eot);}
    /*can have multiple sources for legacy
        support*/
}

h1, h2 {
    font-family: ChunkFiveRegular,
        Georgia, serif;
}
```

Subsetting

- Reduces filesize of font while throwing out other chars

Font-weight and Font-style: Bold/Italic

```
.credits {
  font-weight: bold; /*can be number*/
  font-style: italic;
}
```

Text Transform: Lower/Uppercase/Capitalize

```
h1 {
  text-transform: uppercase;
}
h2 {
  text-transform: lowercase;
}
.credits {
  text-transform: capitalize;
}
```

Text-decoration: Underline & Strike

```
.credits {
  text-decoration: underline;
}
a {
  text-decoration: none;
}
```

Line-height

```
p {
  line-height: 1.4em;
}
```

Letter & Word spacing

```
h1,
h2 {
  text-transform: uppercase;
  letter-spacing: 0.2em;
}
.credits {
  font-weight: bold;
  word-spacing: 1em;
}
```

Text Alignment

```
h1 {
  text-align: left;
}
p {
  text-align: justify;
}
.credits {
  text-align: right;
}
```

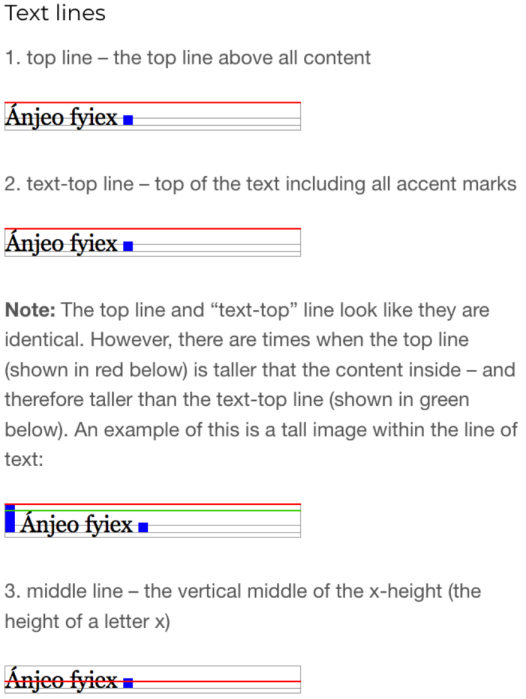
Vertical Alignment

- Commonly used with inline elements

```
#six-months {
  vertical-align: text-top;
}
#one-year {
  vertical-align: baseline;
}
#two-years {
  vertical-align: text-bottom;
}
```

`vertical-align` How not to use it

- Nice article that paints the different text lines and shows how the inline element (blue box) is aligned with its containing box



Text indenting

```
h1 {
  background-image: url(images/logo.gif);
  background-repeat: no-repeat;
  text-indent: -9999px;
}
.credits {
  text-indent: 20px;
}
```

- used to hide actual text and replace it with an image

Text shadow

```
.one {
  background-color: #eeeeee;
  color: #666666;
  text-shadow: 1px 1px 0px #000000;
}
p.two {
  background-color: #dddddd;
  color: #666666;
  text-shadow: 1px 1px 3px #666666;
}
p.three {
```

```
  background-color: #cccccc;
  color: #ffffff;
  text-shadow: 2px 2px 7px #111111;
}
```

First letter/First line pseudo-elements (part of element)

```
p.intro:first-letter {
  font-size: 200%;
}
p.intro:first-line {
  font-weight: bold;
}
p.intro::first-letter {
  font-size: 200%;
}
p.intro::first-line {
  font-weight: bold;
}
```

Pseudo Classes: visited, link (State)

```
a:link {
  color: deeppink;
  text-decoration: none;
}
a:visited {
  color: black;
}
a:hover {
  color: deeppink;
  text-decoration: underline;
}
a:active {
  color: darkcyan;
}
```

What′ s the difference between pseudo-class and pseudo-element?

- Pseudo-class selects element based on state,
- pseudo-element only selects a part of an element

Pseudo-classes vs Pseudo-elements

Pseudo class

Used to define a special state of an element

- Word preceded by a single colon (`:`)
- Possible to combine (chain) them together
- Refer to the element to which they are attached
- Examples: :active :hover :focus :link

Pseudo element

- Used to style specified parts of an element
- Word preceded by two colons (`::`)
- Only one pseudo-element is permitted in a given selector
- Combining pseudo-classes with pseudo-elements is allowed
- Examples: ::first-letter ::first-line ::after

Questions

Which one of the following elements is by defauult, an inline element?

```
<h1>
  ,
  <li>
    ,
    <div>, <strong></strong></div>
  </li>
</h1>
```

Answer: `strong`

What font family has extra details on the ends of the main strokes of the letters? Sarif

What font family is clearer to read with lower resolutions? Sans-Sarif

What′ s the difference between pseud-class and pseudo-elements? Pseudo-class selects element based on state, pseudo-element only selects a part of an element

Images

Types of Graphics

Graphic types commonly used on web pages:

- GIF
- JPG
- PNG

- ...svg, webp

Gif

- Graphics Interchange Format
- Best used for line art and logos
- Maximum of 256 colours + transparent
- can be animated
- uses lossless compression
- can be interlaced

JPEG

- Joint Photographic Experts Group
- Best used for photographs
- Up to 16.7 million colours
- use lossy compression
- cannot be animated
- cannot be made transparent

PNG

- Portable Network Graphic
- Supports millions of colours
- Support levels of transparency
- Support interlacing
- use lossless compression
- combines the best of gif and jpeg

Image Optimization

- The process of creating an image with the lowest file size that still renders a good quality image - **balancing image quality and file size**
- Photographs taken with digital cameras are not usually optimized for the web
- Use a graphics application to:
  - Reduce image dimensions
  - Reduce size of the image file
- Just don′ t use a 4k Image if you don′ t really need it, it could harm others internet connection - You could compress it - or crop it Image Optimization:
- Reduce the file size of the image
- Reduce the dimensions of the image to the actual width and height of the image on the web page
- Reduce the quality of the image (Blur, Colours →Black and white)

Helps for Search engine *indexing* and user experience / usability

Other file formats

- WEBP (-30%)
- AVIF (-50%)

CSS

Controlling sizes of the image using CSS

```
img.large {
  width: 500px;
  height: 500px;
  /*you should not use the height because
      it could interfer with the ratio*/
}
img.medium {
  width: 250px;
  height: 250px;
}
.../* This does not change the image
    itself. CSS is only the presentation
    layer*/;
```

→Layout shift

- Specifying image sizes may help pages, for example, to load the content more smoothly. This is because the HTML and CSS code will often load before the images, and telling the browser how much space to leave for an image allows it to render the rest of the page without waiting for the image to download

Should we use absolute sizes or relative? (Open question) →Combination of both

Images without dimensions

- (in HTML) Omitted by developers when responsive design was introduced but still required...
- Include width and height
- Do not include units (always pixels)
- Why?
  - To reserve the required space of the boxes
  - Increase usability

Flexible images

→Cumulative layout shift

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

Predefine image space so If the Image loads later, the text won′ t get pushed around.

web.dev otpimize →It is inserted directly in the HTML

Responsive Images HTML 5.1 Picture Element

```
<picture>
  <source media="(min-width: 1200px)"
      srcset="large.jpg" />
  <!--default-->
  <source media="(min-width: 800px)"
      srcset="medium.jpg" />
  <source media="(min-width: 320px)"
      srcset="small.jpg" />
  <source src="fallback.jpg"
      alt="waterwheel" />
</picture>
```

Browser will decide which image to get →saves band width

or

```
<img
  src="fallback.jpg"
  sizes="100vw"
  srcset="large.jpg 1200w, medium.jpg
      800w, small.jpg 320w"
  alt="waterwheel"
/>
```

Can be used to set the design focus from images for Images that would get cropped badly.

Aligning Images using CSS

```
img.align-left {
  float: left;
  margin-right: 10px;
}
```

Can be created to wrap text around images. ONLY USE IF FLAOT 100% NEEDED! →BAD RESPONSIVENESS! Use grid or flags instead!

Centring images using css

```
img.align-center {
  display: block;
  margin: 0 auto; /* centering, also 0px =
      0 */
}

img.medium {
  width: 250px;
  height: 250px;
}
```

Remember: to center block level elements using an auto margin requires that we set the width of the Element manually!

css-tricks.com centering-css-complete-guide

Vertical Centering

- Cenetering things vertically is a bit tricker in CSS
- There are so many different ways of doing it depending on the situation (block vs inline, know height vs unknown, ...)

Background Images

```
p {
  background-image: url(./image.gif);
}
```

(no quotation marks in urls)

`background-image` Does it count as content? No →CSS is only used for style

`baclground-repeat: repeat-x/repeat-y/repeat/no-repeat` Is used to repeat the image so it fills the block

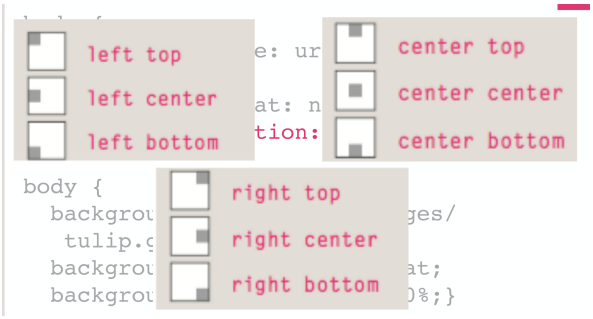
`background-attachment: scroll` is the default. The image will follow so you can scroll it away. With `fixed` it stays right in place

`background-position: left top / left center / left bottom` x and y . `50% 50%` =

`center center`, The top-left corner is 0 0 (only when the background is not repeated)

The `background-size` CSS property sets the size of the element′ s background image. The image can be left to its natural size, stretched, or constrained to fit the available space

```
/* first value: width of the image, second
    value: height */
background-size: 50% auto;
```



Shorthand notation: `background: #ffffff url(images/tulip.gif)no-repeat top right;}` (Sets color and image)

What selector do I use to set up a background image for the entire website?

→Body

Gradients

```
#gradient {
  background-image:
      linear-gradient(#336666, #66cccc);
}
```

Recap

How can we center block level elements using CSS?

- margin: 0 auto;
- (max-)width: 80%

By default, the width and height properties only take into account the size of the content (box model). How can we change these behaviours to include the margins and paddings?

- (usally widt = 100% + padding + border)
- `box-sizeing: border_box`

What CSS property and value do we need to make a background image to remain at the same position even when the user scrolls the page?

- background-attachment: fixed;

What are the differences of Pseudo Classes and Pseudo elements:

Positioning

Key Concepts in Positioning elements

- In containers (Main Container = Body)

Positioning methods:

- Define exactly where element boxes will appear:
  - Relative to where they would ordinarily be (normal flow)
  - In relation to a parent element
  - relative to the viewpoint
- position CSS property - Values:
  - Static (default)
  - Relative
  - Sticky
  - Absolute
  - Fixed
- We also have float (but it is not a position value)

Normal Flow: Position Static

- Every block-level element appears on a new line
- In order they are coded in the web page document
- Default value; the element is rendered in normal flow

Relative Positioning (position: relative)

- This moves an element from the position it would be in normal flow, shifting it to the top, right, bottom, or left of where it would have been places
- (relative to itself)
- There will be space from where the element normally would be (normal flow)

- Can lead to overlapping
- `position: relative`

```
p.example {
  position: relative;
  top: 10%; /*moves it 10% down from the
      top (relative to itself or
      containing (absolute) */
  left: 100px; /*shifts it 100px to the
      right (100px are added on the left
      side)*/
  /*other props: right / bottom */
}
```

Absolute Positioning (position: absolute)

- This positions the element in relation to its containing element (parent)
- It is taken out of normal flow, meaning that it does not affect the position of any surrounding elements
- can also leads to overlapping (stuff is handled if it wasn′ t there)
- !! If you want to place a object inside an element, you need to make the containing element a different position (eg. relative)
- `position: absolute`

Fixed Positioning

(Sort of Absolute)

- Positions the element in relation to the browser window/viewport (not the parent container)
- Do not affect the position of surrounding elements
- They do not move when the user scrolls up or down (it stays at the same spot)
- `position: fixed`

Floating Elements

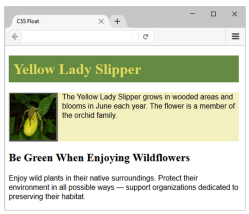
- This is a form of relative positioning
- Allows to take the element out of the normal flow (also removes it from the normal flow)
- Specifies that an element should be placed along the left or right side of its container
- Position the element to the far left or right of a containing box
- The floated element becomes a block-level element around which other content can flow
- DO NOT USE TO CREATE LAYOUTS
- Maybe you don′ t need floats anymore for creating layouts,
  - Better approaches such as flexbox or grid layout

```
img {
  flow: right;
  /*or*/
  flow: left;
}
```

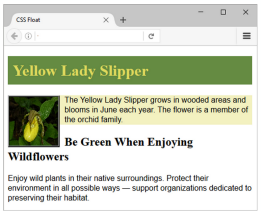
clear property

- Useful to "clear" or terminate a flow
- Values are left, right and both

The h2 text is displayed in normal flow.



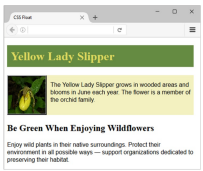
`clear: left;` was applied to the **h2**. Now the h2 text displays AFTER the floated image.



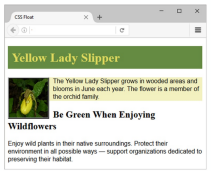
overflow property

- Configures the display of elements on a web page
- Useful to "clear" or terminate a float before the end of a container element
- Values are `auto, hidden and scroll`

The background does not extend as far as you'd expect.



`overflow: auto;` was applied to the container **div.** Now the background extends and the h2 text displays AFTER the floated image.



Sticky

- Combines relative and fixed positioning
- Treated as relatively until its containing block crosses a specified threshold (e.g. top)
- Then treated as fixed (stuck) until meeting the opposite edge of its containing block
- VERY INTERESTING

Overlapping elements

z-index

Defines the stack

- z-index: 0 is the normal screen, 1 is over the normal screen...`z-index: 10;`

Layouts

Mobile Design Quick Checklist

- Small screen size
- Bandwidth issues
- Single-column layout
- Maximize contrast
- Optimize images for mobile display
- Descriptive alternate text for images
- Avoid display of non-essential content

Layout Strategies

- Fixed layouts: Stay put at a specific pixel width regardless of the size of the browser
- Fluid (or liquid) layouts: resize proportionally when the browser window resized
- Elastic Layouts: resize proportionally based on the size of the text
- Hybrid layouts: combine fixed and scalable areas

Fixed Layouts

Fixed layouts are created at a specific pixel width

Advantages:

- Predictable and offers better control over line length
- Easier to Design and produce
- Pixel values are accurate at controlling size and positioning of elements Disadvantages:
- Left over space on large screens (big gaps around the edge of a page)
- Designed for an specific screen resolution
- User needs to scroll the page horizontally on small screens

Liquid Layouts

In liquid page layouts the page area and columns within the page get wider or narrower to fill the available space in the browser window

Advantages:

- No Horizontal scrollbars
- No unnecessary spaces around the page (large screen)
- Users can control the width of the window and content

Disadvantages:

- The design can look very different then you intended on different screen sizes
- Hard to read on large screens
- Words may be squashed in small screens

Elastic Layouts

In elastic layout, the dimensions of the containers scale with users text size. It works by sizing all elements with em′ s Advantages:

- Provides a consistent layout experience while allowing flexibility in the text size
- More control over line lengths than liquid and fixed layouts

Disadvantages:

- Is much more difficult to create
- the extra bit of usability it brings may not always seem worth it
- Not as useful for addressing device and browser size variety

Hybrid Layouts

Combination of all of them (advantages of all of them).

Advantages:

- More flexible
- Responsive design
- share the advantages of the 3 approaches

Disadvantages:

- It can be hard to implement
- Share the disadvantages of the 3 approaches

Layout Modules

CSS3 has different layout modules: - Block, for sections in a webpage - Inline, for text - Table, for

two-dimensional table data - Positioned, for explicit position of an element - Flex, to design flexible responsive layout structure without using float or positioning. Single dimension. - Grid, it offers a grid-based layout system, with rows and columns without having to use floats and positioning. Two dimensions.

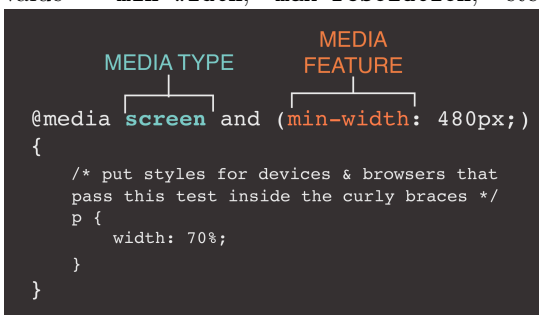
Layout Patterns

common patterns

- mostly fluid
- column drop
- layout shifter
- tiny tweaks
- off canvas

Media Queries

- Allow designers to deliver styles based on media type:
  - `print`, `screen, handheld, braille`, projection, `screen`, tty, `tv` and `all`
- Can evaluate specific media features
  - `Device-width`, `orientation`, and `resolution`, etc.
- Properties (media features) can be tested for a minimum or maximum value - `min-width`, `max-resolution`, etc




```
@media screen and (min-width: 480px) and
    (max-width: 800px) {
  /* put styles for devices & browsers
      that pass this test inside the curly
      braces */
  p {
    width: 70%;
  }
}
```

Can be chained together with `and`

You can also use the `media` tag

```
<head>
    <link> rel="stylesheet" media="screen
        and ...</link>
</head>
```

Questions

What form of absolute positioning positions the element in relation to the browser window (not the parent container)? →Position: Fixed

Which CSS at-rule allows you to link to typefaces that are not installed on the users computer? →Font-face?

When an element is floated, which display value is the element set in? →Block

When using te float property, which other property are you most likely to need? →Width

Flexbox

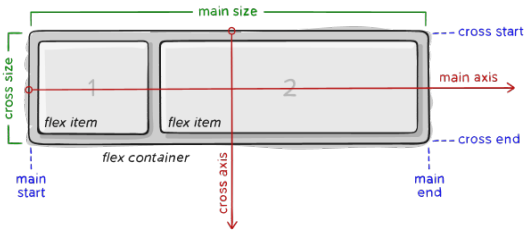
Flexbox

> The Flexbox layout is a layout module in CSS3, made to improve items size, alignment, directions and order in a container.

Flexbox Terminology

- everything is in the `flex-container`
- `main axis` and `cross axis` (could be horizontal and vertical)
- `main start` and `main end` (could be left and right (horizontal))
- `cross start` and `cross end` (vertical)
- Everything will be placed from main start to main end
- All of the children of the `flex-container` are `flex items`
  - they have `main size` and `cross size`
- To use flexbox layout, we need to set the display property on

the parent element (container)



Display: flex

- The is `flex` and `inline-flex` →the `flex-container` only takes up as much space as it needs

```
.flex-container {
  display: flex;
  display: -webkit-flex; /*Safari*/
}
.flex-inline-container {
  display: inline-flex;
  display: -webkit-flex; /*Safari*/
}
```

Flexbox properties

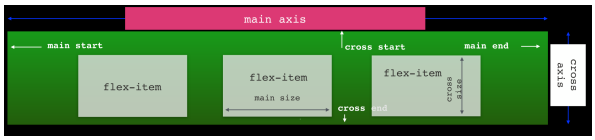
There are several properties related to flexbox. We can group these in:

- Properties for the **container**
- Properties for the **items**

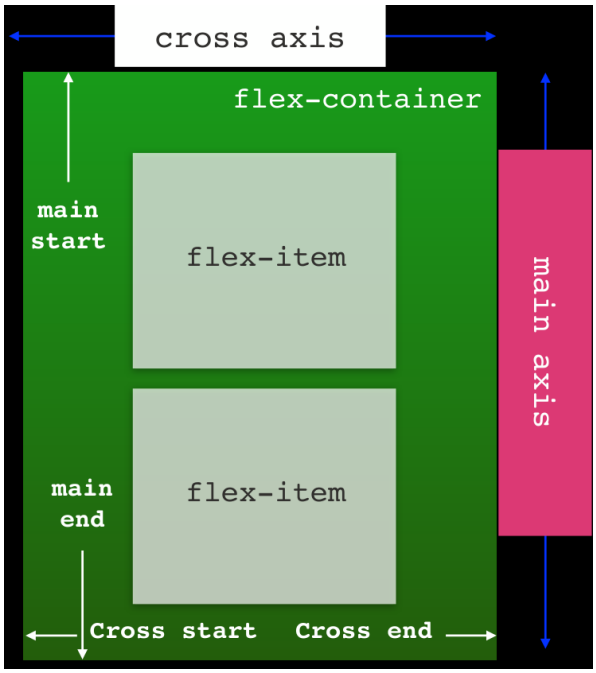
Container Properties

flex-direction

- The `flex-direction` tells the browser wether the element are laid vertically or horizontally
- With row direction, items are stacked from left to right. With `row-reverse`, items stack from right to left



- If the `flex-direction` is set to column then the elements are laid from top to bottom.
- `column-reverse` does the opposite




```
.flex-container {
  flex-direction: row; /* row-reverse /
      column / column-reverse */
  -webkit-flex-direction: row; /*Safari*/
}
```

- if we don' t specify a direction, the default is **row**

Flex-wrap

- By default, a flex container will have its items placed on a single line, so flex-items will adjust to fit the container
- The flex-wrap property is used to tell the container that the items can wrap around when there' s no space for them (similar to float)
- To use wrap: `flex-wrap: wrap;` or `flex-wrap: wrap-reverse;`
- `flex-wrap: nowrap;` is the default

flex-flow

- `flex-flow` is a shorthand property that lets us set the flex-direction and the flex-wrap values in one line

```
.flex-container {
  flex-flow: row wrap;
```

```
}
```

justify-content The justify-content property aligns items along the main axis

```
.flex-container {
  justify-content: flex-start; /*
      flex-start / flex-end / center /
      space-between / space-around /
      space-evenly */
}
```

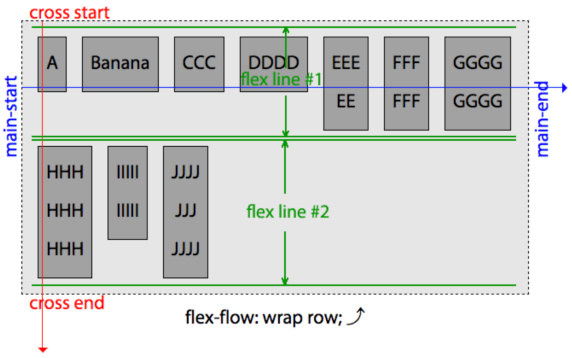
align-items

- The property `align-items` works pretty much like `justify-contents`, but along the cross direction of the flow (cross axis), on a flow line
- `align-items: stretch;` is the default

```
.flex-container {
  align-items: flex-start; /* flex-start /
      flex-end / center / baseline /
      stretch */
}
```

align-content The `align-content` property modifies the behaviour of the flex-wrap property. It is similar to align-items and justify-content, but instead of aligning flex items, it aligns flex lines. (only for multi-line flexboxes possible). The `align-content: stretch;` is the Default

```
.flex-container {
  align-content: flex-start; /* flex-end /
      center / space-between /
      space-around / space-evenly /
      stretch */
}
```



Item Properties

The item properties are for items inside the flex container.

order Flex-items can be ordered inside a container, without changing the html code, with the order property. The default is 0 `order: 0;`

```
.flex-item {
  order: 1;
}
```

flex-grow We can decide the size of flix-items in relation to other items by using the flex-grow property. flex-grow property is strongly discouraged by the authors of the specification itself.

```
.flex-item {
  flex-grow: 0; /* Default */
}
```

flex-shrink flex-shrink specifies how much the item will shrink in relation to others. Is also not recommended to use.

```
.flex-item {
  flex-shrink: 0; /* Default */
}
```

flex-basis flex-basis takes the initial value of the item, exactly as using width or height.

The flex-basis determines how the flex growth and shrink factors are implemented

```
.flex-item {
  flex-basis: 80px;
}
```

→The flex item will start out at 80px and can grow to fill up the available space →The flex item will start out at 80px but then shrink to fit the space available with the other items being at least min-content sized.

flex (shorthand) flex-grow, flex-shrink and flex-basis can be written together in the shorthand `flex`. It is recommended to use the shorthand notation!

```
.flex-item {
  flex: 0 1 350px; /* [ <flex-grow>
      <flex-shrink>? || <flex-basis> ] |
      none */
}
```

align-self `align-self` is used when you need to override align-items for a specific flex-item.

```
.flex-item {
  align-self: auto; /* Default */
  /* auto | flex-start | flex-end | center
      | baseline | stretch */
}
```

Grid layout

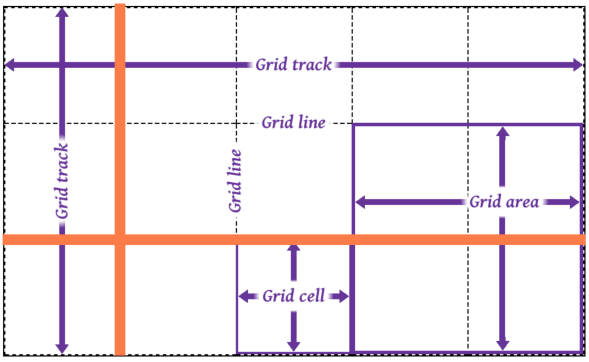
Grid Layout definition

The CSS Grid Layout is a layout module in CSS3 that offers a grid-based layout system which aims to make it easier to design web pages without having to use floats and positioning.

- Grid layout is a generalised layout system (2 dimensions)
  - rows
  - columns
- Grid allows pieces of the design:
  - to be laid out independently of their document source order
  - to overlap pieces of the layout

Terminology

- The most fundamental unit is the grid line
- by defining the placement of one or more grid lines,
- you implicitly create the rest of the grids components



- A grid track is a continuous run between two adjacent grid lines
  - In other words, columns and rows
- A grid cell is any space bounded by four grid lines, with no grid lines running through it
  - in other words, something similar to a table cell
  - Grid cells cannot be directly addressed with CSS grid properties
- A grid area is any rectangular area bounded by four grid lines and made up of one or more grid cells
  - Grid areas are directly addressable by CSS grid properties

Grid Properties

Working with grid layouts implies working with grid lines.

Grid tracks, cells, and areas are entirely constructed of grid lines, and more importantly, do not have to correspond to grid items.

Properties for the Grid Container	Properties for the Grid Item
display	
grid-template-columns	grid-column-start
grid-template-rows	grid-column-end
grid-template-areas	grid-row-start
grid-template	grid-row-end
grid-column-gap	grid-column
grid-row-gap	grid-row
grid-gap	grid-area
justify-items	justify-self
align-items	align-self
place-items	place-self
justify-content	

Grid container props

Display: grid To use the grid layout, we need to set the display property on the parent element (container)

```
.grid-container {
  display: grid; /*inline-grid;*/
}
```

Placing Grid lines

grid-template-columns & grid-template-rows

- Used for placing column and row grid lines
- Formal syntax: `none | <track-list> | <auto-track-list>`

```
.grid-container {
  display: grid;
  grid-template-rows: 20% 1fr 20%;
  grid-template-columns: 100px max-content
      1fr;
  /* none / auto / max-content /
      min-content / length / initial /
      inherit; */
  /*shorthand:*/
  grid-template: 20% 1fr 20% / 100px
      max-content 1fr;
  /*rows then columns*/
}
```

Fixed-Width (inflexible) Grid Tracks

- Grid tracks are a fixed width (e.g.: pixels or ems)
- Percentages also count as fixed-width here
- "fixed-width" means the grid lines are placed such that the distance between them does not change due to changes of content within the grid track

```
.grid-container {
  display: grid;
  grid-template-columns: 200px 50% 100px;
  /*(50% of the grid container)*/
}
```

Flexible Grid Tracks Based on the

1. amount of space not consumed by inflexible tracks
2. actual content of the entire track

`fr` (fractional unit): divide up whatever space is available by some fraction

```
.grid-flexible-container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
}
/* or */
.grid-flexible-mix-container {
  display: grid;
  grid-template-columns: 200px 50% 100px
      1fr 1fr;
  /* the last two fr are really small and
      occupie the remaining space from the
      container*/
}
```

- `minmax(min,max)`: defines a size range, greater than or equal to min, and less than or equal to max
- `min-content`: the largest minimal content contribution of the grid items occupying the grid track
- `max-content`: largest maximal content contribution of the grid items occupying the grid track Example:

```
.grid-minmax {
  display: grid;
  grid-template-rows: max-content
      minmax(3em, 100%) min-content;
```

```
}
```

Repeating Grid lines:

- `repeat()`: to set up a bunch of grid tracks of the same size. `repeat(5, 1fr)`
- also possible:
  - to repeat patterns `repeat(5, 2em 1fr)` (repeats 5x)
  - to repeat concat grid tracks `repeat(5, 2em 1fr)100px` (after the last element from the repetition there will be new one with 100px)

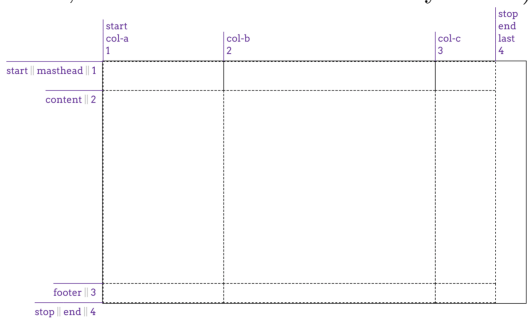
Auto-filling tracks

- `auto-fill` can be used to set up a simple pattern and repeat it until the grid container is filled

```
.grid {
  grid-template-rows: repeat(auto-fill,
      5em);
  grid-row: 2;
}
```

Placing Grid lines tipps Some facts about grid lines:

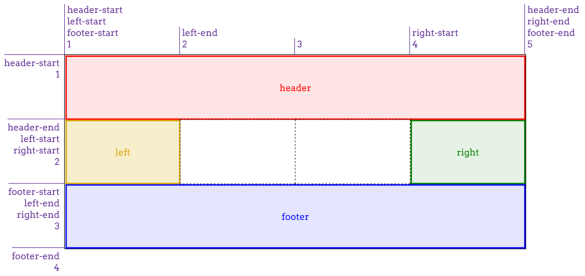
- Grid lines can always be referred to by number (starting from 1)
- They can also be named by the author
- Same grid line can have more than one name
- They can be named using a square bracket notation `grid-template-columns: [start col-a] 200px [col-b] 50% [col-c] 100px [stop end last];` (start/col-a are names, but start and end are always there)



grid-template-areas Used to define template areas

- Repeating the name of a grid area causes the content to span those cells
- A period signifies an empty cell
- The syntax itself provides a visualization of the structure of the grid
- It can be named using a square bracket notation

```
#grid {
  display: grid;
  grid-template-areas:
    "header heade header header"
    "left ... ... right"
    "footer footer footer footer";
  /* you can also adjust the size */
  grid-template-rows: 1fr 1fr 1fr 1fr;
  grid-template-columns: 1fr 1fr 1fr 1fr;
}
```



Grid gaps

- `grid-column-gap`, `grid-row-gap` and `grid-gap` specify the size of the grid lines.

```
grid-row-gap: 1em;
grid-column-gap: 3em;
grid-gap: row-gep-legth column-gat-length
    \rightarrow grid-gap: 1em 3em; =
    shorthand
```

justify-items horizontal alignment of grid items

```
.grid {
  justify-items: start; /* end / center /
      stretch */
}
```

align-items Vertical alignment of grid items

```
.grid {
  align-items: start; /* end / center /
      stretch */
}
```

justify-content Horizontal alignment of grid contents (i.e. all the grid items when the grid container is greater than the sum of grid items)

```
.grid {
  justify-content: start; /* end / center
      / stretch */
}
```

place-items shorthand for `align-items / justify-items`

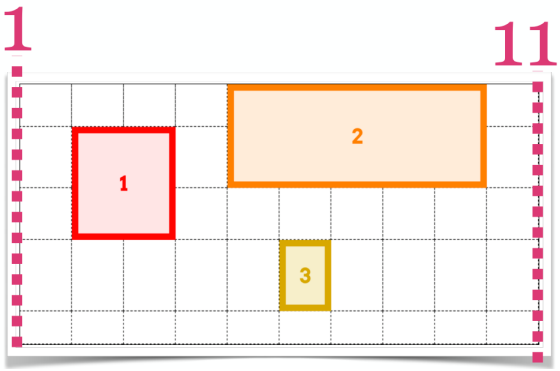
```
.start-end {
  place-items: start end;
}
```

Grid item Properties

Attaching Elements to the grid grid-row-start, grid-row-end, grid-column-start, grid-column-end

- used for attaching elements to a grid-line
- Formal syntax: `<number> | <name> | span <number> | span <name> | auto`
- Using grid-line numbers to say where and how the elements should be placed within the grid
- Column numbers count from left to right
- Row numbers from top to bottom
- numbers start from 1

```
.box01 {
  grid-row-start: 2; /* from top to bottom
      second line */
  grid-row-end: 4; /* from top to bottom
      forth line */
  grid-column-start: 2; /* from left to
      right second line */
  grid-column-end: span 2; /* spans 2 grid
      cells from start */
}
```



- Another way to say that same thing is by replacing the ending values by span
- Supplying span with a number means "span across this many grid tracks"
- Supplying only span is the same as span 1
- Another way to say that same thing is by replacing the values by the named grid lines (named lines example)

```
.header {
  grid-row-start: header-start;
  grid-row-end: header-end;
  grid-column-start: header-start;
  grid-column-end: header-end;
}
```

Shorthand:

- grid-row and grid column are a shorthand for:
  - `grid-row-start + grid-row-end` and `grid-column-start + grid-column-end`
  - `grid-area` (shorthand)

```
.leftside {
  grid-column: left-start / left-end; /*
      named lines example*/
  grid-row: left-start / left-end;
  grid-area: 1 / 1 / 3 / 6; /* row start /
      col start / row end / col end */
}
```

justify-items Horizontal alignment of the grid item. Overrides container grid value

```
.item {
  justify-self: start; /* end / center /
      stretch */
}
```

align-self Vertical alignment of the grid item. Overrides container grid value

```
.item {
  align-self: start; /* end / center /
      stretch */
}
```

place-items Shorthand for `<align-tems> / <justify-items>`

```
.start-end {
  place-items: start end;
}
.end-end {
  place-items: end;
}
```

Difference between flex and Grid

!!! warning Difference between Flex and grid Flex: single dimension. Grid: 2 dimension

CSS3 animations

- where previously only possible with Flash or JavaScript
- **CSS Transitions**
  - a way to make style changes fade smoothly from one to another
  - needs a trigger
- **CSS Transforms**
  - a way to rotate, relocate, resize, and skew HTML elements in both two-and three-dimensional way
  - needs a trigger
- **CSS Keyframe Animation**
  - Similar to transitions but mire complex and powerful
  - does not need a tricker

CSS transitions

Apply a style smoothly or gradually

- Which CSS property to change (**transistion-property**)
- How long it should take (**transition-duration**)
- The manner in which the transition accelerates (**transition-timing-function**)
- Whether there should be a pause before it starts (**transition-delay**)
- You also need something to trigger the transition. A state change such as `:hover`, `:focus` or `:active` makes a good trigger

transition-property

Transition-property identifies the CSS property that we want to transition smoothly

transition-duration

Sets the amount of time it will take for the animation to complete in seconds (s) or milliseconds (ms). In our next example, it′ s 2 seconds.

Example

```
#box1{
...
  background-color: red;
  transition-property: background-color,
      color; /* or "all" */
  transition-duration: 2s;
  transition-timing-function: linear;
  transition-delay: 2s;
}
/* This will switch the colors after two
    seconds with a 2 seconds duration*/
#box1:hover {
  background-color: blue;
  color: red;
}
```

transition-timing-function

- Default: `ease` Starts slowly, accelerates quickly, then slows down at the end
- The transition-property and the transition-duration from the foundation of a transition,

but you can refine it further. The transition-timing-function is used to set the way in which a transition can roll out over time
- Other values: `ease / linear / ease-in / ease-out / step-start / step-end / steps`

transition-delay

- the transition-delay property delays the start of the animation by a specified amount of time
- If you just want to add a little bit of smoothness to all your state changes, regardless of which property might change, use the **all*** value for `transition-property: all;`

multiple transitions

- We can change several properties at once
- By listing all of the values for each property separated by commas

CSS Transform

- A way to rotate, relocate, resize, and skew HTML elements in both two-and three-dimensional space
- You can apply a transform to the normal state of an element, and it will appear in its transformed state when the page loads
- It is common to pull out the transforms only when user interact with the element
- Transforms are supported on all major browser versions with vendor prefixes
- Does need a trigger

`trsansform` needs a value (which is a function), for example `rotate(degree)` or `translateX(offset)`

rotate

- Use the rotate function to make the element appear tilted
- `transform: rotate(-20deg)`

```
#box6:hover {
  transform: rotate(-20deg);
  transform-origin: center top; /* 50% 0 /
      75px 0 */
}
```

translate

Use the `translateX(offset)`, `translateY(offset)` or `translate(offsetX, offsetY)` to move the rendered element to a new location. Positive or negative values are accepted

```
#box9:hover {
  transform: translateX(20px);
}
```

scale

Transforms the size. Use the `scaleX(horizontal)`, `scaleY(vertical)` or `scale(horizontal, vertical)` to make an element appear larger or smaller

```
#box12:hover {
  transform: scaleX(1.5);
}
```

skew (warp/strach)

Use the `skewX(horizontal)`, `skewY(vertical)` or `skew(horizontal, vertical)` to change the angle of either the horizontal or vertical axis (or both axes) by a specified number of degrees

```
#box15:hover {
  transform: skewX(15deg);
}
```

transform origin

- to indicate from which point the element will be rotated
- first value is the horizontal offset and the second is the vertical offset (if only one is provided, it will be used for both)

```
.gallery-img {
  max-width: 100%;
  height: auto;
  object-fit: cover;
  opacity: 0.5;
  /* Define transition*/
  transition: opacity 0.5s ease-in-out,
      transform 2s ease-in-out;
}
```

```
.gallery-grid > a {
  overflow: hidden;
}

/*Trigger transition when hover*/
.gallery-img:hover {
  opacity: 1;
  transform: scale(1.5);
}
```

Keyframe animation

- similar to transitions
- more granular control: several states
- more complex
- **Trigger not needed**
- Keyframe animation is known as explicit animation because you program it′ s behaviour

Basics

Two parts:

1. Establish the keyframe with a `@keyframe` rule and name

```
@keyframes animate-background {
  0% { background-color: red; }
  25% { background-color: orange; }
  50% { background-color: yellow; }
  75% { background-color: green; }
  100% { background-color: purple; }
}
```

2. Add animation properties to the elemets that will be animated

```
#box18 {
  animation-name: animate-background;
}
```

Properties

- Which animation to use (`animation-name`)
- How long it should take (`animation-duration`)
- The manner in which it should accelerate (`animation-timing-function`)
- Whether to pause before it starts (`animation-delay`)
- How many times it should repeat (`animation-iteration-count`)
- Whether it plays forward, in reverse, or alternates back and forth (`animation-direction`)
- Whether it should be running or paused. The play-state can be toggles on and off with JavaScript or on hover (`animation-play-state`)
- Whether to override defaults that prevent properties from applying out-side runtime (`animation-fill-mode`)

```
#box18 {
  animation-name: super-animation;
  animation-duration: 5s;
  animation-iteration-count: infinite;
      /*required*/
  animation-direction: alternate;
}
```

Recap

What is the difference between FlexBox and Grid Layout? -> Flexbox is only one dimension and Grid Layout has 2

Zje way to make style changes fade smoothly from one to another is known as...-> Transition

How is it know the way to rotate, relocate, resize, and skew HTML elements in both two- and three-dimensional

Forms

Why forms

- For example search bars
- Login/register
- Shipping address
- INPUTS

Form controls

- 4 Types:
  - Adding Text (single-line, password, text area/multi-line)
  - Making Choices: Radio buttons, Check-boxes, Drop-down boxes
  - Submit Forms: Submit buttons, Image buttons

- Upload file form

How forms work

1. User fills in form and presses button to submit info to server
2. Name of each form control sent with value user entered
3. Server processes information using programming language
4. Server creates new page (or confirmation response) to send back to the browser based on info received

Name and Value Pairs

- Name: `username`, value: `Ivy` -> `username = Ivy`

Form Structure

```
<form action="http://example.com/join.php"
    method="get">
  This is where the form controls will
      appear.
</form>
```

- YOU HAVE TO HAVE SOMETHING IN THE ACTION
- Method is either `get` or `post`
-

Text input

```
<form action="http://example.com">
  <label for="username">Your Username:
      </label>
  <input type="text" name="username"
      size="15" maxlength="30" />
  <!-- or type="password" -->
</form>
```

use `<label>` and not `<p>` !!!

Text area

```
<p>What do you think of this gig?</p>
<textarea name="comments" cols="20"
    rows="4">
  <!-- cols and rows represent letters -->
  Enter your comments
</textarea>
```

Radio Buttons

- needs to have the same name

```
<p>Your favorite genre:
  <input type="radio" name="genre"
      value="rock" checked="checked"/> Rock
  <input type="radio"
      name="genre"value="pop"/> Pop
</p>
```

Checkbox

- needs to have the same name
- Returns a list of items
- `checked="checked"` makes it the default
- Same syntax as Radio Buttons, but: `type="checkbox"`

Drop down list box `<select>`

```
<select name="devices" multiple="multiple">
  <option value="ipod"
      selected="selected">ipod</option>
  <option ...>iPhone</iption>
</select>
```

- `selected="selected"` makes it the default
- Can also act as checkboxes `<select multiple="mutiple"...>` but is not required
  - Can set the max. amount of selectable options: `size="4"` →you are allowed to select up to 4

File Input Box

- Method must be `post`

```
<form action="localhost" method="post">
  <input type="file" name="user-song" />
  <input type="submit" value="Subscribe" />
</form>
```

Submit Button

- `<input type="submit" value="Subscribe"/>`
- Could also be image: `type="image" src="image.jpg" with=... height...` but don′ t use this!

Hidden Form control

- For example csrf tokens
- mostly interesting for JS
- `<input type="hidden" name="bookmark" value="lyrics" />`

Labels

- `<label>Age: <input type="text" name="Age"/></label>`
- instead of `<p>`
- Or `<input id="female" .../><label for="female"...>Female</label>`
- the `for` is referencing the `id female`

```
<form action="localhost">
  <label>Age: <input type="text"
      name="Age" /></label>
  <!-- or -->
  <input id="female" type="radio"
      name="gender" value="f" />
  <label for="female">Female</label>
</form>
```

Grouping Forms

- `<fieldset>`

```
<fieldset>
  <legend>Contact details</legend>
  <label>Email: <input type="text"
      name="email"></label>
  ...
</fieldset>
```

HTML5: Form Validation

- `required="required"` make the field required to be filled before submitting (inside `<input required="required" .../>`)

HTML5: Date Input

- `<input type="date" .../>`
- `type="email"`
- `type="url"`
- `type="search"` (the same as input but maybe different style)
  - `placeholder="Enter keyword"` creates a placeholder

Web Accessibility

> Universal design is design that′ s usable by all people, to the greatest extent possible, without the need for adaptation or specialized design″

Usable by all people:

- Challenge: define all people (how to make sure it′ s functional for people with any type of impairment?)
- **UD** (universl design) is for everyone!

To the greatest extend possible:

- Must work for as many people as possible regardless of struggles with: (1) upper body movement, strength, and/or sensation, (2) lower body movement strength, and/or sensation, (3) balance, (4) vision, (5) hearing, (6) cognition and memory, (7) activity tolerances, (8) speech and/or communication, (9) chemical sensitivities, (10) sensory tolerance, (11) needs for caregiver assistance, and (12) extremes in height and weight.

Without need for adaptation or specialisation

- Individuals do not have to change the way their typically interacts with something
- Some people may still have significant functional needs that require specialized design. In those cases, UD is the foundation but further development is needed

Universal design

Originally created for architecture and industrial design but expanded to include digital products and services

Based on seven universal design principles:

1. equitable use;
2. flexibility in use;
3. simple and intuitive use;
4. perceptible information;
5. tolerance for error;
6. low physical effort;
7. size and space for approach and use.

- There is no typical, average, normal user
- User contect will vary widely based on circumstances

Inclusive design

Design methodology that enables and draws on the full range of human diversity

- Inclusive design means making your product available to as many users as possible (like UD, right?)
- Focuses on the process of diversity. This is:
  - Is about engaging authentically with the diverse people and, therefore,
  - it may involve different solutions for different groups of people (edge-cases) - One size fits one

Accessibility

- Goal: ensure there are no barriers (technical, physical or cognitive) to serving products or services to someone
- It also includes testing usability (usually involving users - user testing)
- It could be seen as one piece of inclusive design

Web accessibility

- The inclusive practice of removing barriers that prevent interaction with, or access to, websites by people with disabilities
- The degree to which a web system is usable by as many people as possible
- A quality, - how easily and effectively a system or service can be accessed and used