Week 02 - CSS

Petr Wehrenberg

Outline

- Intro to CSS and how it works
- Layouting vs. styling
- BEM
- Responsivity
- Alternatives, tips and tricks

What is CSS?

- Stands for **Cascading Style Sheets**
- Basic HTML is readable in a browser, but isn't aesthetically pleasing
- Basic styling (colors, fonts) and advanced styling (animations, 3D transforms)
- Allows to style different viewport widths (mobile vs desktop) within a document

An example of CSS

```css
.btn {
  font-size: 1.15rem;
  color: white;
  padding: 1rem 1.5rem;
  text-align: center;
  background-color: #36393f;
}
```

```css
selector {
  property: value;
}
```

How to include styles

Link

`/style.css`

```css
.cool-navbar {
  display: flex;
  color: #343434;
}
```

`/index.html`

```html
<html>
  <head>
    <link rel="stylesheet" href="style.css" />
    <link rel="stylesheet" media="max-width: 600px" href="mobile.css" />
  </head>
  <body>
    <nav class="cool-navbar"></nav>
  </body>
</html>
```

Style element

`/index.html`

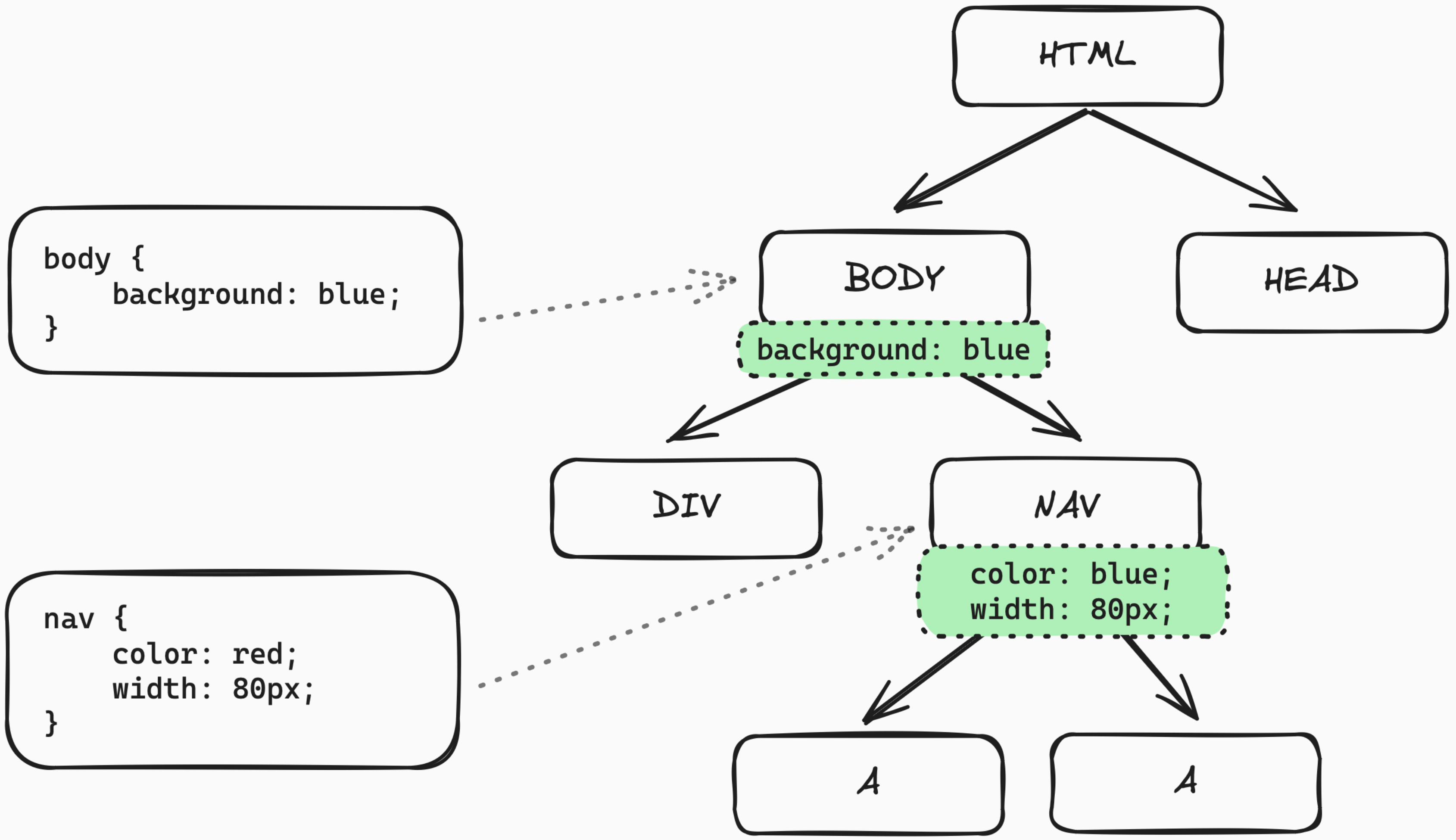
```
<html>
  <head> </head>
  <body>
    <nav class="cool-navbar"></nav>
  </body>
  <style>
    .cool-navbar {
      display: flex;
      color: #343434;
    }
  </style>
</html>
```

Inline

`/index.html`

```html
<html>
  <head> </head>
  <body>
    <nav style="display: flex; color: #343434"></nav>
  </body>
</html>
```

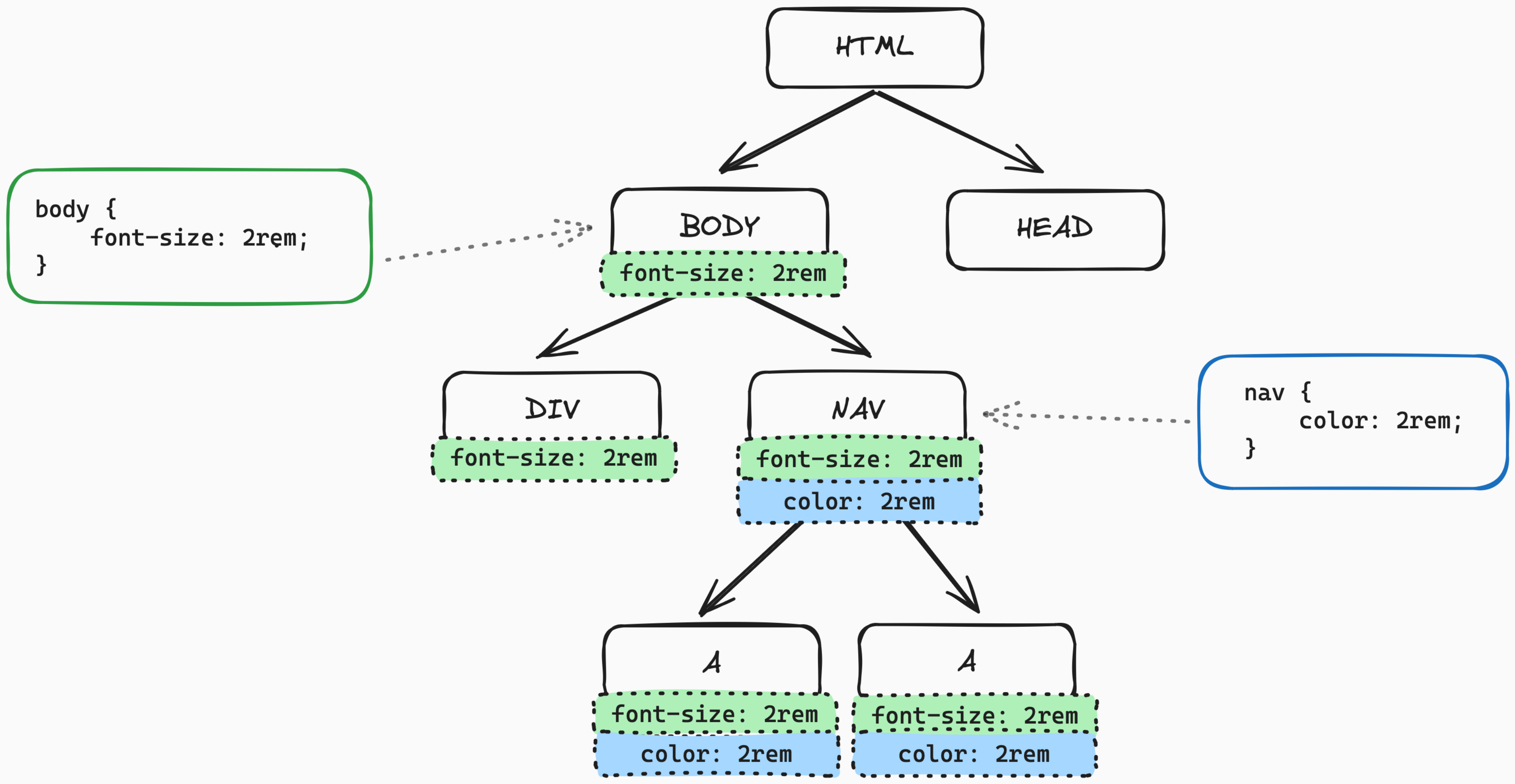
So how it works



```
body {
    background: blue;
}
```

```
nav {
    color: red;
    width: 80px;
}
```

"Cascading" meaning



```css
body {
    font-size: 2rem;
}
```

```css
nav {
    color: 2rem;
}
```

CSS Selectors

```css
#identifier {
}

.class {
}

div {
}
```

Specificity

element < class < identifier < inline

CSS Pseudo elements

- State: `hover`, `visited`, or `focus`
- Ancestor relationship: `first-child`, `last-child`, `only-child`, `nth-of-type`, `empty`, etc.

```css
.class:hover {
}

input:focus {
}

tr:nth-child(even) {
}
```

CSS Atribute selectors

- the presence of an attribute
- specific value

```
a[title] {
}

a[href="https://example.org"]
{
}
```

CSS Selectors

```css
.comment.btn {
  font-size: 2rem;
}

.comment h2 {
  font-size: 2rem;
}

.comment > .content {
  font-size: 2rem;
}
```

Messy CSS

```css
.header .navigation ul li a.active {
  color: white;
  background-color: blue;
}

body.homepage .content .article > p:first-child {
  font-size: 20px;
}

.footer .widget-area .widget:first h3 {
  font-weight: bold;
}

.sidebar .widget,
.footer .widget-link-list,
.header .dropdown-menu {
  background-color: #f0f0f0;
}
```

CSS Priority

1. `!important`
2. Specificity
3. Source (author, user, broswer)
4. Order in source file

Browser Support

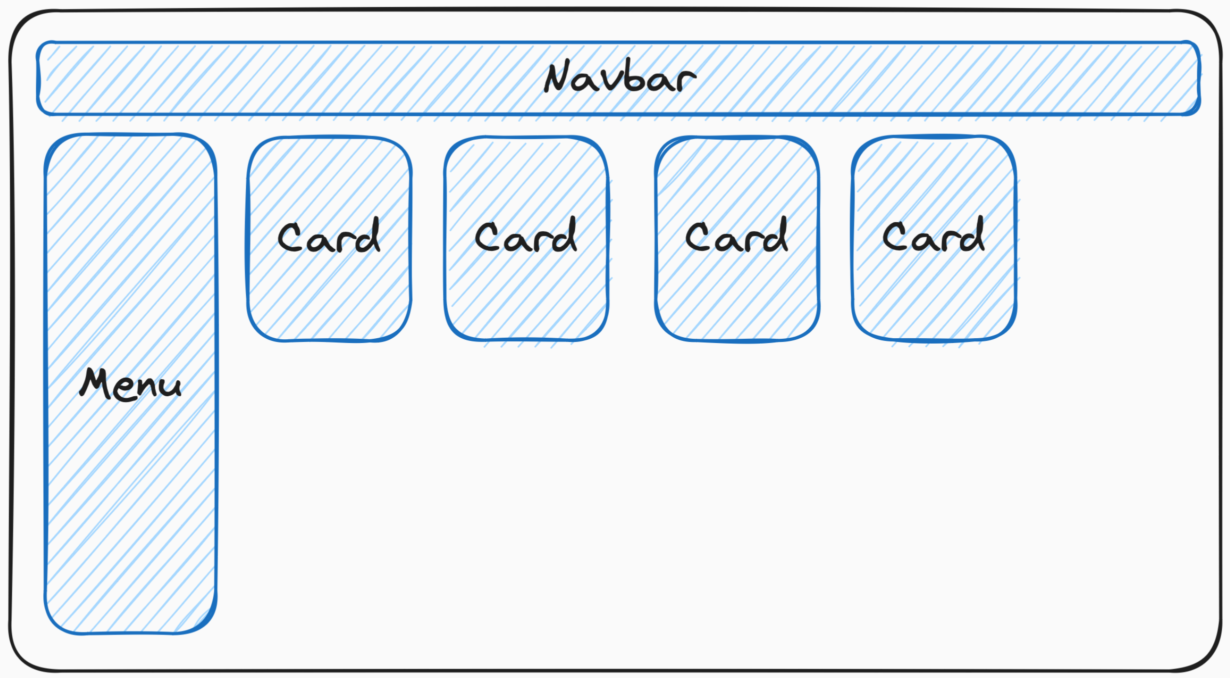
Visit caniuse.com to see what browsers support what features

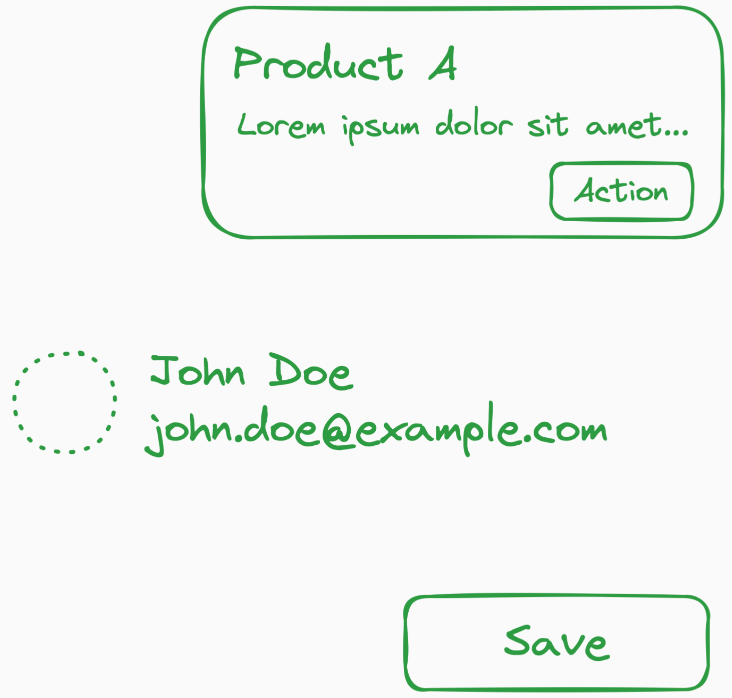
Checkpoint - Intro to CSS

- Anatomy
- Source (css files, css element, inline)
- Selectors (identifier, class, element)
- Pseudo selectors
- Specificity
- Priority

Layouting and styling

Layouting and styling





Styling

- Reusable Components: Buttons, cards, avatars.
- Visual Attributes: Color, font, rounded corners.
- Theming: Dark/light mode, brand colors.
- Interaction Feedback: Hover, active, disabled states.

```css
.btn {
  border-radius: 1rem;
  background-color: #33aa22;
  font-size: 1.2rem;
}
```

CSS Variables

```css
:root {
  --radius-sm: 0.8rem;
  --radius-md: 1rem;
  /* ... */
}

.btn {
  border-radius: var(--radius-md);
  background-color: var(--bg-accent);
  font-size: var(--text-lg);
}
```

CSS Units

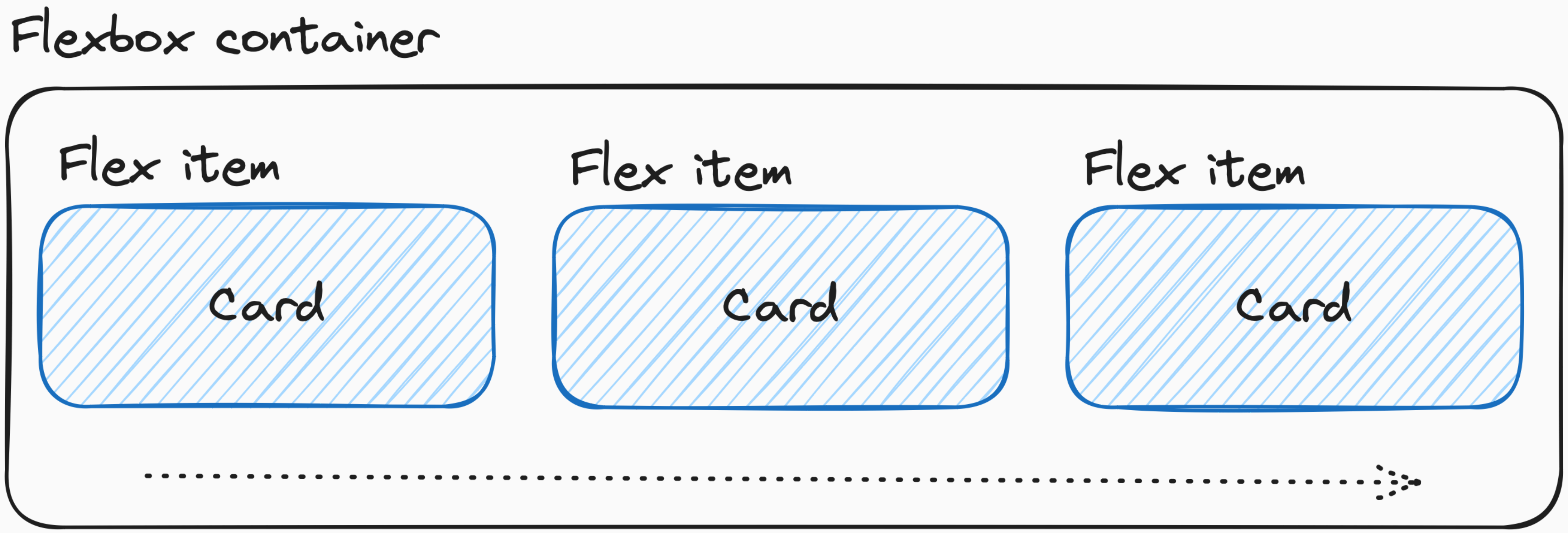
```css
body {
  font-size: 12px; /* NOT Recommended */
  font-size: 2em; /* 2x current font */
  font-size: 2rem; /* 2x root font */

  width: 10rem;
  width: 10vh;
  width: 10vw;
}
```

Layouting

- Component Positioning: Arrange elements spatially.
- Content Flow: Logical structure, easy navigation.
- Responsivity: Media queries, flexible grids.
- Main tools: Grid, Flexbox

Flexbox


Flexbox container
Flex item
Card
Flex item
Card
Flex item
Card

Flex container

```css
.parent-element {
  display: flex;
  flex-direction: row;
  flex-wrap: no-wrap;
  justify-content: space-between;
  align-items: center;
}
```

Flex item

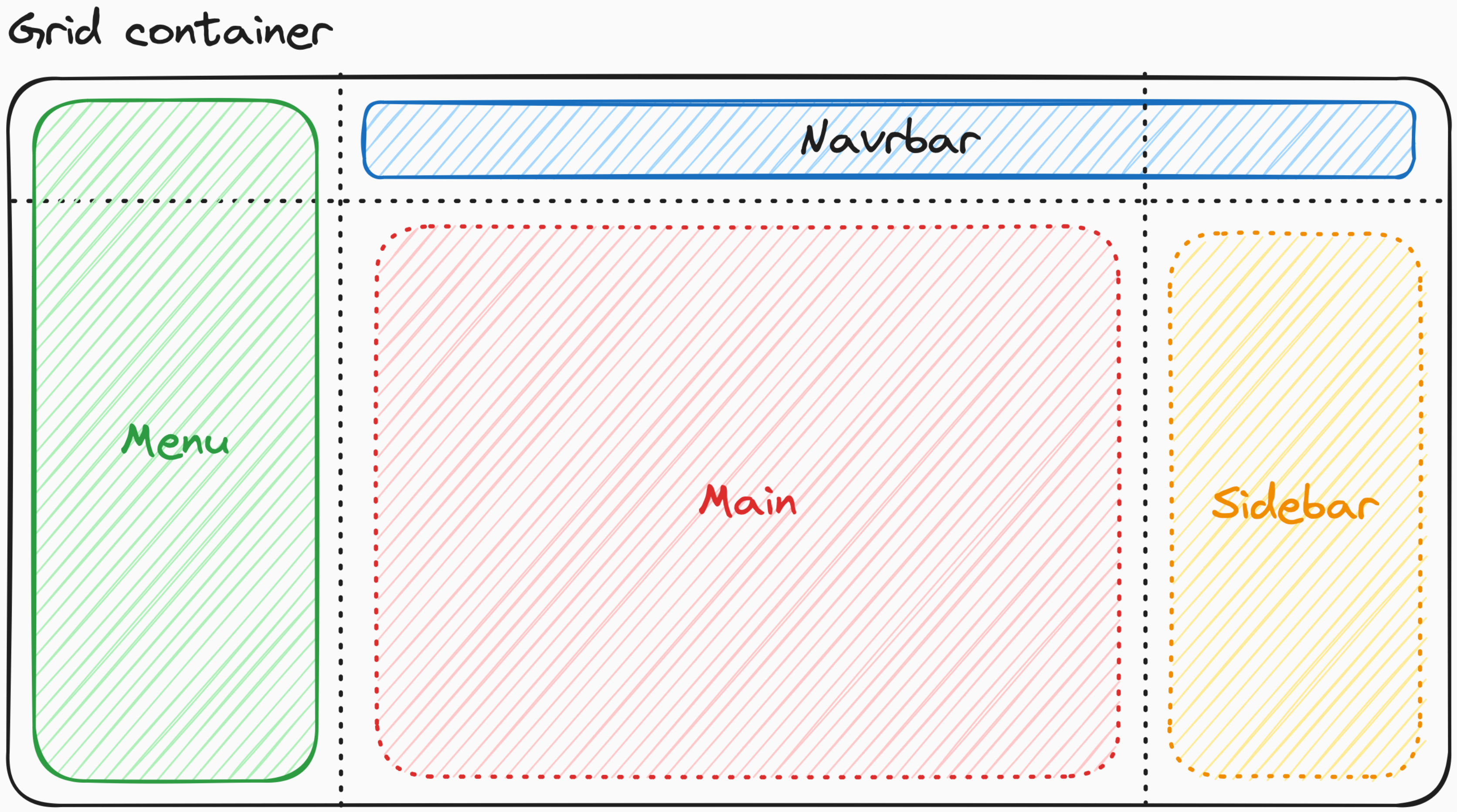
```css
.child-element {
  flex: 1 0 100px; /* grow, shrink, basis */
}
```

Flex Guide

Pictures are worth more than 1000 words of the lecturer:

css-tricks.com/snippets/css/a-guide-to-flexbox/

Grid



Grid container

```css
.parent-element {
  display: grid;
  grid-template-columns: 1fr 1fr 50px;
  grid-template-rows: 30px 30px 30px;
}
```

Grid item

```css
.child-element {
  grid-column-start: 2;
  grid-column-end: 3;
  grid-row: 1 / span 2;
}
```

Grid Guide

Pictures are worth more than 1000 words of the lecturer:

css-tricks.com/snippets/css/complete-guide-grid/

Positioning

Static

```css
.parent-element {
  position: static;
}
```

Relative

```css
.parent-element {
  position: relative;

  left: 30px;
}
```

Absolute

```css
.parent-element {
  position: relative;
}

.child-element {
  position: absolute;

  top: 10px;
  left: 30%;
}
```

Fixed

```css
.child-element {
  position: fixed;
  top: 10px;
  left: 30%;
}
```

Sticky

```css
.parent-element {
  position: relative;
}

.child-element {
  position: sticky;
  top: 20px;
}
```

Positioning: DEMO

Checkpoint - Styling and layouting

- Difference between styling and layouting
- Styling (CSS variables, units)
- Layouting (Flex, Grid, Positioning)

BEM

Problem

```css
.header .navigation ul li a.active {
  color: white;
  background-color: blue;
}

body.homepage .content .article > p:first-child {
  font-size: 20px;
}

.footer .widget-area .widget:first-of-type h3 {
  font-weight: bold;
}

.sidebar .widget,
.footer .widget-link-list,
.header .dropdown-menu {
  background-color: #f0f0f0;
}
```

Solution

```css
.nav {
  background-color: #ee42fd;
}

.nav__item {
  margin: 5px 0;
}

.nav__link {
  color: black;
  text-decoration: none;
}

.nav__link--active {
  color: white;
  background-color: blue;
}
```

What is BEM?

- The BEM methodology helps to think with components:

- **Block** - a standalone entity that is meaningful on its own
- **Element** - a part of a block with no meaning on its own, semantically tied to block
- **Modifier** - a flag on block/element used to modify appearance or behavior

BEM - Block

- A BEM Block consists of a singular component, which can potentially be reused throughout the page
- It encapsulates an entity on the page (search bar, navbar, button, avatar, ...)

CSS:

```css
.avatar {
  display: flex;
  clip-path: circle(50% at center);
  overflow: hidden;
}
```

HTML:

```html
<div class="avatar">
  <!-- Some markup here -->
</div>
```

BEM - Element

- BEM elements are tied to the block
- They are parts of the block that make up the whole component, meaningless* on their own
- They are defined **after** the definition of the block in CSS

CSS:

```css
/* This is an element of the "avatar" block. */
.avatar__image {
  object-fit: cover;
}
```

HTML:

```html
<div class="avatar">
  <img class="avatar__image" src="..." alt="..." />
</div>
```

BEM - Modifier

- BEM Modifiers modify the block / element
- Modifier can change a few properties, they usually change color, styling, visibility, etc.
- They are defined **after** the definition of the block / element in CSS

```css
/* Main styling of a block */
.message {
  display: flex;
  flex-direction: row;
  color: #fff;
  background-color: #232323;
}

/* Modifier of a block - inherits most of the properties
   and changes some - for example colors, sizes... */
.message--highlighted {
  background-color: #776a23;
}
```

BEM - Modifiers

- BEM Modifiers **cannot be used standalone**!
- They do not inherit properties, only overwrite some of them.

```html
<!-- A correct use of a BEM Modifier (modifies a "message" BEM Block) -->
<div class="message message--highlighted">Here is some content</div>

<div class="avatar">
  <!-- Also a correct usage of a BEM Modifier (modifies an "avatar__image" BEM Element) -->
  <img
    class="avatar__image avatar__image--squared-corners"
    src="..."
    alt="..."
  />
</div>

<!-- ! An incorrect use of a BEM Modifier ! -->
<div class="message--highlighted">Here is some other content (but wrong)</div>
```

BEM - Common mistakes

```css
.button--icon__large {
} /* naming structure */

.card__header__button {
} /* nested elements */

.input__field--error--large {
} /* nesting and combining modifiers */

.menu_item_isActive {
} /* naming convention */

.form__submit--text {
} /* misusing modifiers*/
```

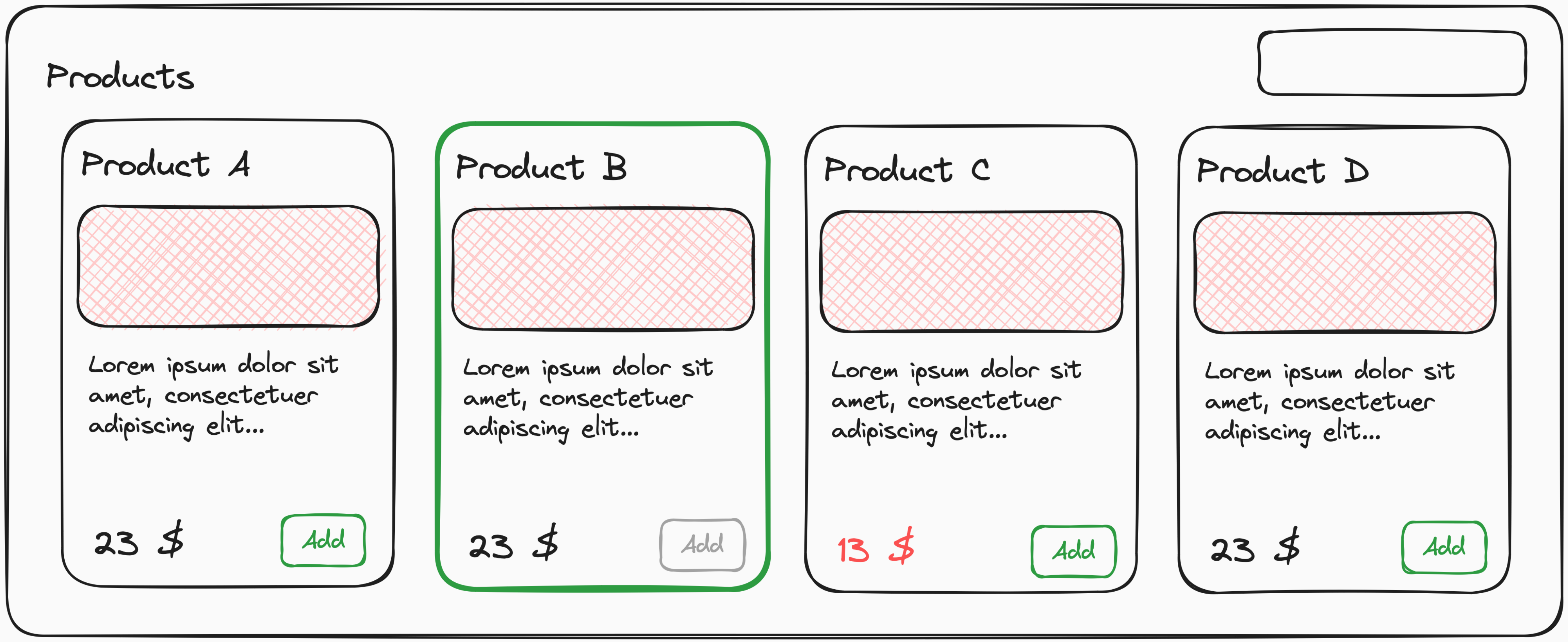
BEM - Common mistakes

```html
<div class="message message--unread">
  <span class="message__sender"> Obi-Wan Kenobi </span>
  <div class="message__avatar">
    <!-- Incorrect use of BEM, DO NOT do this: -->
    <img class="message__avatar__image" src="..." alt="..." />
  </div>
  <p class="message__content">Hello there!</p>
  <!-- Incorrect use of BEM, DO NOT do this: -->
  <button class="button__edit">Edit</button>
</div>
```

BEM - Example

```html
<div class="message message--unread">
  <span class="message__sender"> Obi-Wan Kenobi </span>
  <div class="message__avatar avatar">
    <img class="avatar__image" src="..." alt="..." />
  </div>
  <p class="message__content">Hello there!</p>
</div>
```

Excercise


Products
Product A
Lorem ipsum dolor sit amet, consectetuer adipiscing elit...
23 $
Add
Product B
Lorem ipsum dolor sit amet, consectetuer adipiscing elit...
23 $
Add
Product C
Lorem ipsum dolor sit amet, consectetuer adipiscing elit...
13 $
Add
Product D
Lorem ipsum dolor sit amet, consectetuer adipiscing elit...
23 $
Add

Checkpoint - Styling and layouting

- Block, Element, Modifier
- Better CSS structure
- No nesting!!

Responsivity

Responsive design

- Plain HTML is responsive
- Today's websites should be developed **mobile-first**
- Using browser dev tools to change viewport to simulate a phone screen
- Tablet and desktop styles come later
- But how do we distinguish which ones to use when?
- **Media queries**

Media queries

```css
/* Base styles for mobile */
body {
  padding: 20px;
}

/* Media query for tablets */
@media (min-width: 768px) {
  body {
    padding: 40px;
  }
}

/* Media query for desktops */
@media (min-width: 1024px) {
  body {
    padding: 60px;
  }
}
```

Media queries - advanced

```css
@media (min-width: 30em) and (orientation: landscape) {
  /* … */
}

@media not print {
  /* … */
}
```

Container queries

```html
<div class="post">
  <div class="card">
    <h2>Card title</h2>
  </div>
</div>
```

```css
.post {
  container-type: inline-size;
}

.card h2 {
  font-size: 1em;
}

@container (min-width: 700px) {
  .card h2 {
    font-size: 2em;
  }
}
```

Load styles based on media width

```html
<html>
  <head>
    <link rel="stylesheet" href="base.css" />
    <link rel="stylesheet" href="components.css" />
    <link
      rel="stylesheet"
      media="min-width: 764px"
      href="components-desktop.css"
    />
  </head>
  <body>
    <nav class="cool-navbar"></nav>
  </body>
</html>
```

Responsivity: DEMO

Checkpoint - Responsivity

- Mobile first approach
- Media queries
- Container queries

Alternatives

CSS in JS

```
// JSS example
const useStyles = createUseStyles({
  myButton: {
    color: "green",
    margin: {
      top: 5, // jss-default-unit makes this 5px
    },
  },
});

const Button = ({ children }) => {
  const classes = useStyles();
  return (
    <button className={classes.myButton}>
      <span className={classes.myLabel}>{children}</span>
    </button>
  );
};
```

```
// Emotion example
const Button = styled.button`
  padding: 32px;
  background-color: hotpink;
  font-size: 24px;
  color: black;
  font-weight: bold;
  &:hover {
    color: white;
  }
`;

() => <Button>This my button component.</Button>;
```

CSS in JS

Good

- JS variables
- Locally scoped

Bad

- Runtime overhead
- Bundle size

We do not recommend this approach!

Tailwindcss

```
<div class="grid grid-cols-3 gap-4 md:grid-cols-5 p-2">
  <button class="focus:outline-black text-white text-sm">Click me</button>
  <p class="overflow-scroll max-w-full">Click me</button>
</div>
```

Tailwindcss

Good

- Simple
- Faster styling
- Bundle size

Bad

- Might lead to messy code

Preprocessors and Frameworks

- Sass, Less, PostCSS (and others) augment regular CSS
- Add variables, mixins, computed values
- Need to compile files to regular CSS before using in production
- Over time, variables and other features were introduced to CSS itself
- The need to use preprocessors has decreased

Preprocessors and Frameworks

```
/* Nesting example */
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li {
    display: inline-block;
  }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

Preprocessors and Frameworks

Good

- Faster styling
- Better dev experience

Bad

- Lots of features are now supported in CSS
- Build time

UI frameworks and libraries

Because life is too short to write all the styles yourself

UI frameworks and libraries

- Bootstrap
- DaisyUI
- MaterialUI
- RadixUI
- Shadcn
- Flowbite
- AntDesign

Checkpoint: Alternatives

- CSS in JS
- Tailwindcss
- Preprocessors and Frameworks
- UI frameworks and libraries

Tips and tricks

- Be consistent
- Design First
- Mobile-first approach
- Keep it simple stupid (KISS)
- Use `clsx` lib

Questions?

Week 02 - CSS

Petr Wehrenberg