JavaScript and the DOM

by @filipbech

Filip

IMPACT™

#weAreHiring #Denmark

LEGO

nemlig·com
DAGLIGVARER NÅR DET PASSER DIG

NOA NOA

Stofa:

quintess

babywalz
welcome to the family

VELUX

Bog&idé

Børsens
FORLAG

Skousen
-den bedste handel

Helsam
Din Helsekost Specialist

TØJ | eksperten

SE
SYD ENERGI

DR

ZEISS

WAGNER

IMERCO

B&W
vølund

RETAIL INSTITUTE
SCANDINAVIA

EGMONT

quint

TDC

HAY

SILVAN

ARNOLDBUSCK

tvilum
living your way

SKAGEN
DENMARK

Nilfisk
setting standards

botex
hjem til dig

dinex

aasted
chocolate & bakery solutions

SVENDBORG BRAKES

TEXMAN

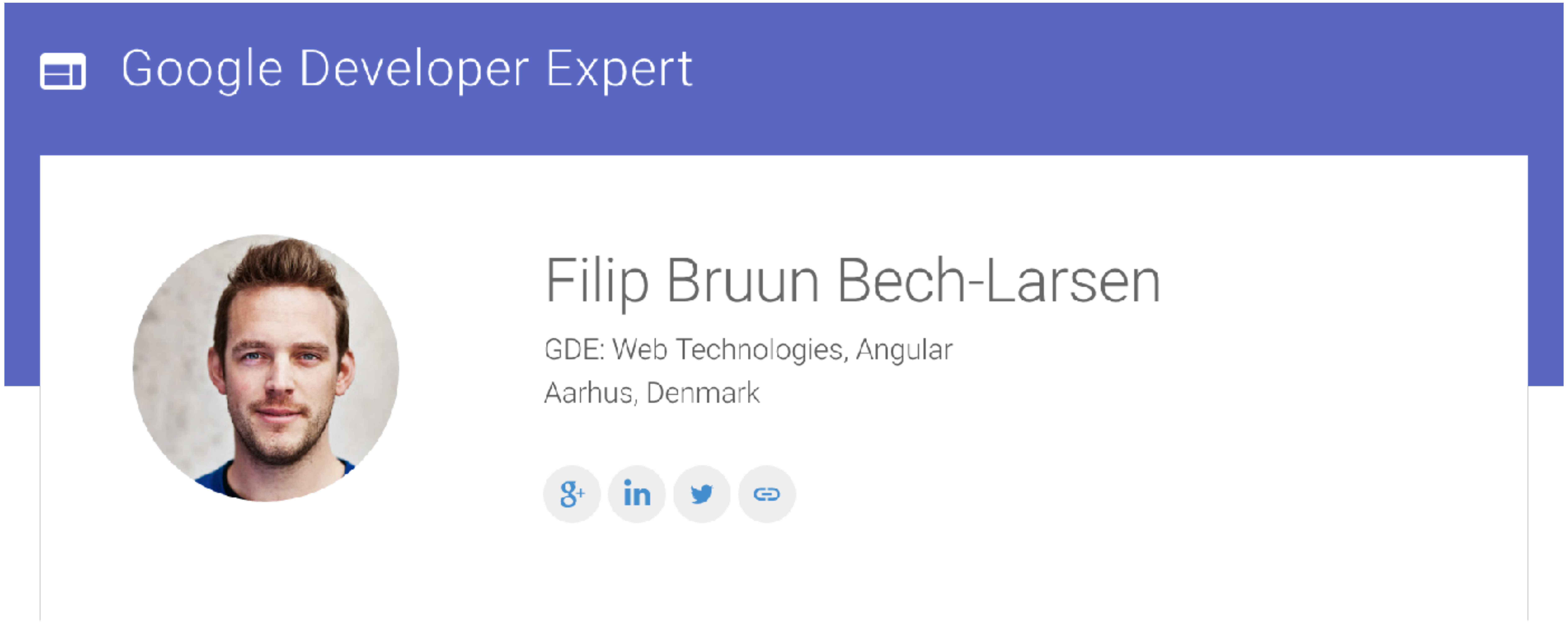
AXEL

KAUFMANN
SINCE 1908

Plantorama

unoliving

ORKLA



Google Developer Expert

Filip Bruun Bech-Larsen

GDE: Web Technologies, Angular

Aarhus, Denmark

https://developers.google.com/experts/people/filip-bruun-bech-larsen https://www.facebook.com/groups/ngAarhus/

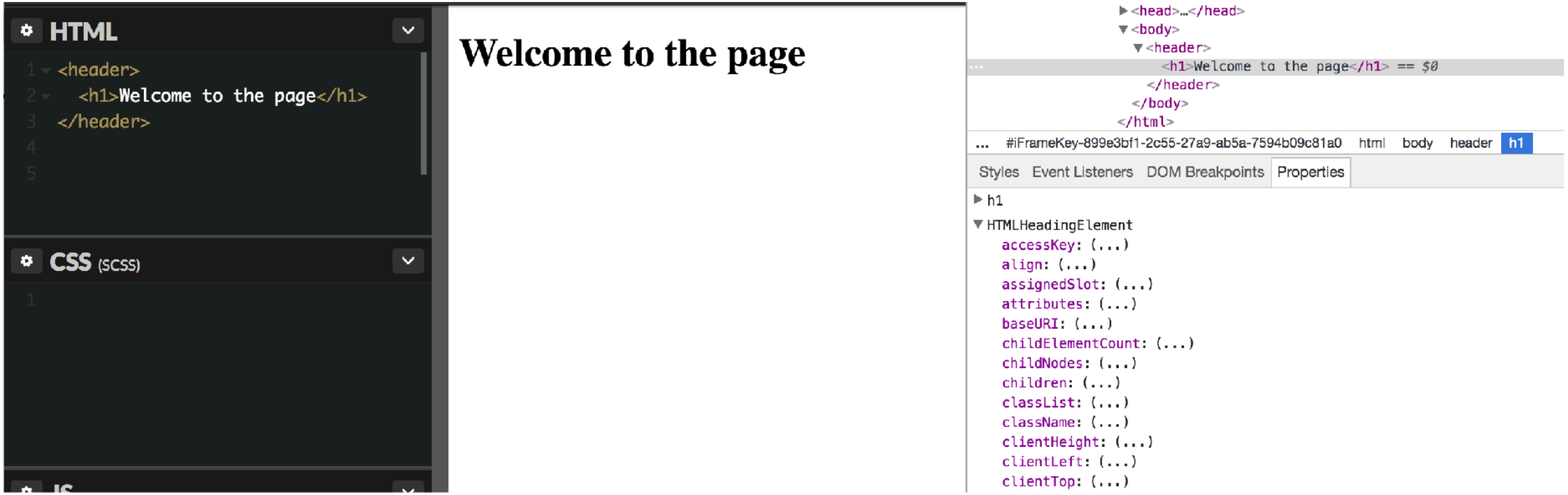
you already know javascript

Beware of some ES2015+ stuff

Document Object Model

- A tree of (DOM) nodes (starting from the documentElement)
- The basis of the view-layer of the (document-based) web
- typically starts out from a (graceful) parsing of an html-document (xml-like, depending on the type)

Lets see it



```
<header>
  <h1>Welcome to the page</h1>
</header>
```

Welcome to the page

```
▶<head>…</head>
▼<body>
  ▼<header>
      <h1>Welcome to the page</h1> == $0
    </header>
  </body>
</html>
```

... #iFrameKey-899e3bf1-2c55-27a9-ab5a-7594b09c81a0 html body header h1
Styles Event Listeners DOM Breakpoints Properties

```
▶ h1
▼ HTMLHeadingElement
    accessKey: (...)
    align: (...)
    assignedSlot: (...)
    attributes: (...)
    baseURI: (...)
    childElementCount: (...)
    childNodes: (...)
    children: (...)
    classList: (...)
    className: (...)
    clientHeight: (...)
    clientLeft: (...)
    clientTop: (...)
```

Use the browser tools for debugging

- All browsers have them, but they are different
- element-inspector
- $0

lets get some definitions straight

- element - (objects in the tree when tags are parsed to DOM)
- attribute (on the element)
- property (on the object)
- **BE AWARE** not all properties are reflected back to their attributes (if they exist)

element vs node

- elements have tagNames (inherits from node)
- nodes can be text

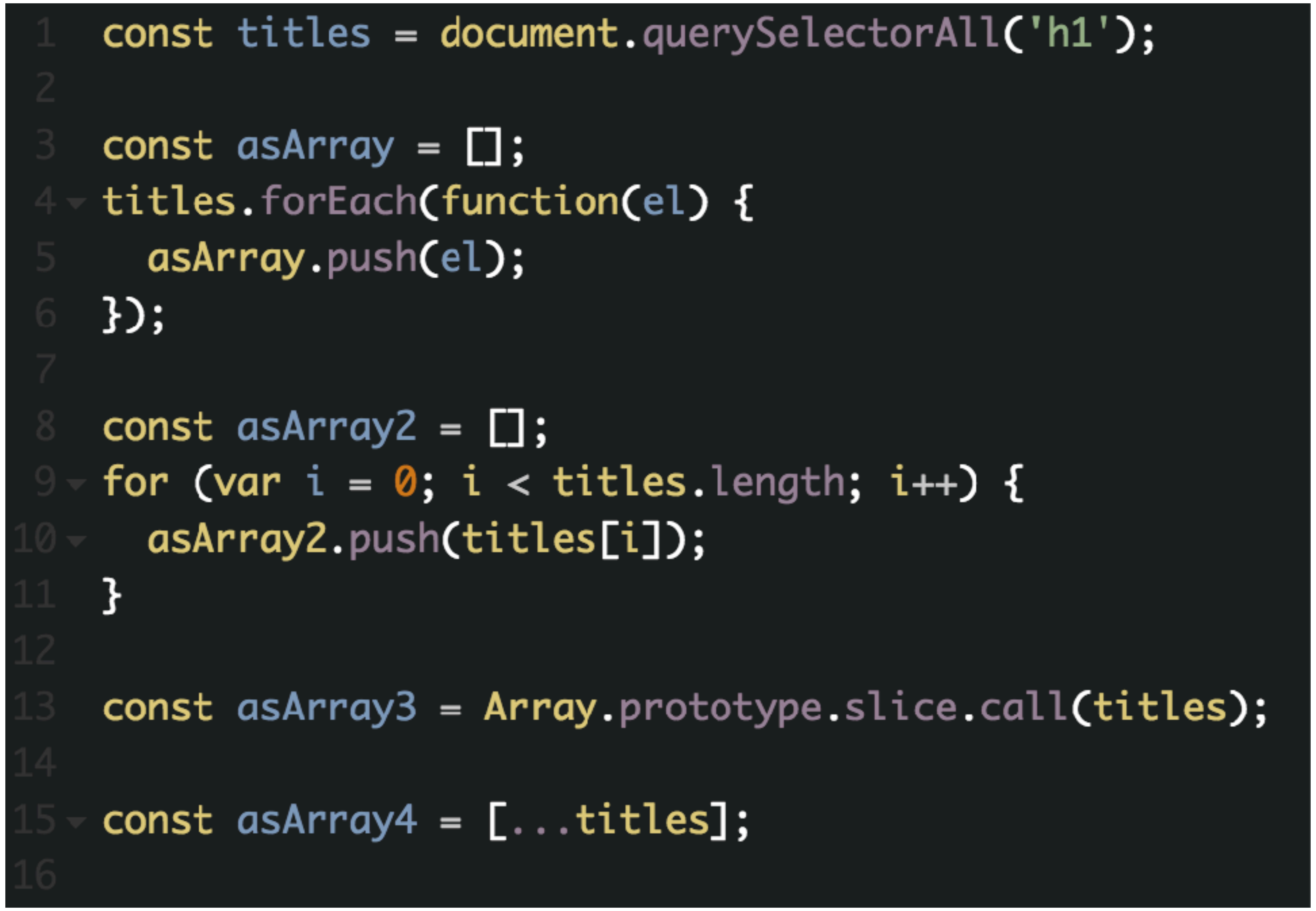
Finding elements in the tree

- getElementById
- getElementsByTagName
- getElementsByClassName
- querySelector (snapshot)
- querySelectorAll (snapshot)
- and the other way around with element.matches(selector):bool

HTMLcollection and Nodelist

- list (or snapshot) of elements and/or nodes
- length + index lookup (e.g. list[0])
- they are similar to Arrays, but lack some of the methods from Array.prototype
- You can call these methods directly from the prototype, or you can convert to an Array.

convert to array



```
const titles = document.querySelectorAll('h1');

const asArray = [];
titles.forEach(function(el) {
  asArray.push(el);
});

const asArray2 = [];
for (var i = 0; i < titles.length; i++) {
  asArray2.push(titles[i]);
}

const asArray3 = Array.prototype.slice.call(titles);

const asArray4 = [...titles];
```

forEach is not on a HTMLcollection

or you can do Array.from(list)

a quick rant

- when googling for DOM-stuff, you will stumble over w3schools. IGNORE all those results - always go for the MDN result

Manipulating the DOM

- createElement (or cloneNode)
- appendChild
- removeChild
- createTextNodes
- innerHTML (or innerText)
- There are other ways that might be a bit faster, but these are stable and simple to understand. (e.g. use these until you have a proven perf-problem)

Manipulating attributes

- Use properties when possible (the data-property for custom data)
- getAttribute
- setAttribute
- use primarily for styling and only when necessary

Forms

- form
- input (default to text for fallback)
- select
- textarea
- button

- Validation, accessibility, etc.

Lets talk about layout

- Types of elements (display-types)
- The old and well supported approach (block, inline-block, inline + floats and clearfixes for layout)
- The modern approach (flexbox, grid + specific configuration)
- out of the flow (with position:absolute or fixed)

Styling

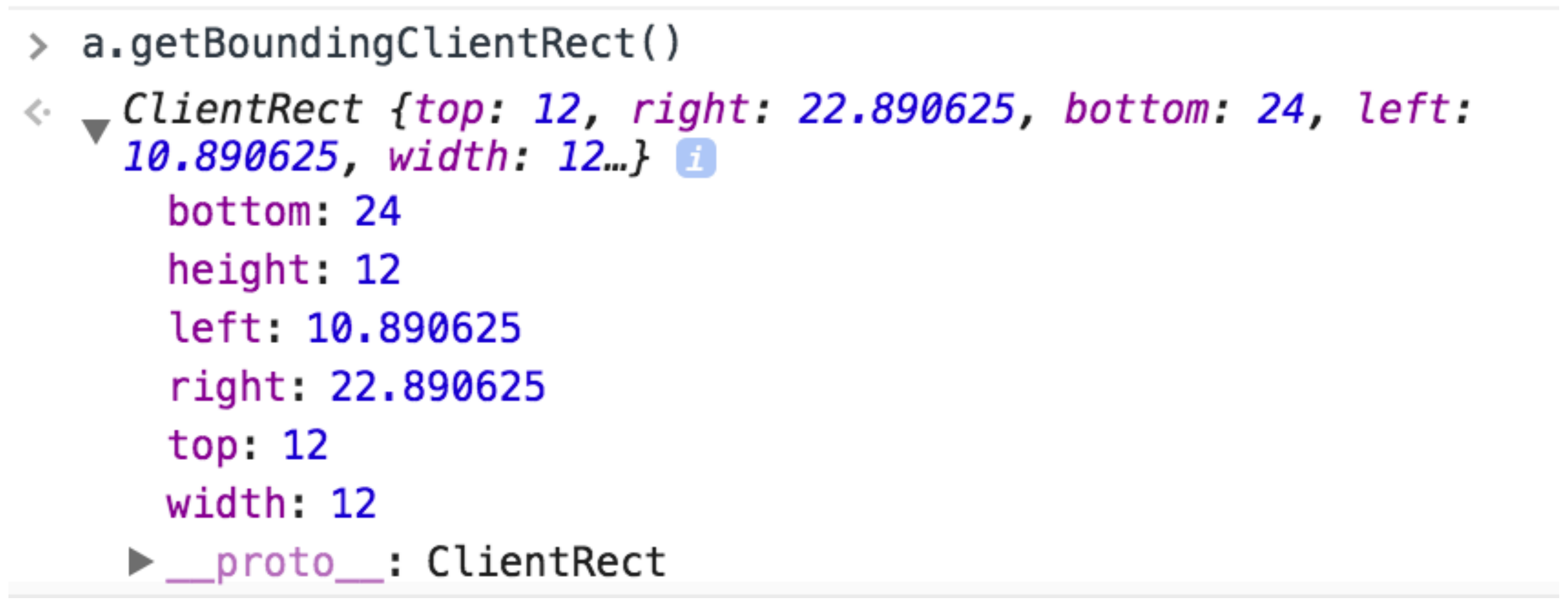
- Selector=>property:value
- Cascading Style Sheets (external file, inline style tag or element-specific styles)
- custom properties
- Normalising styles

Specificity

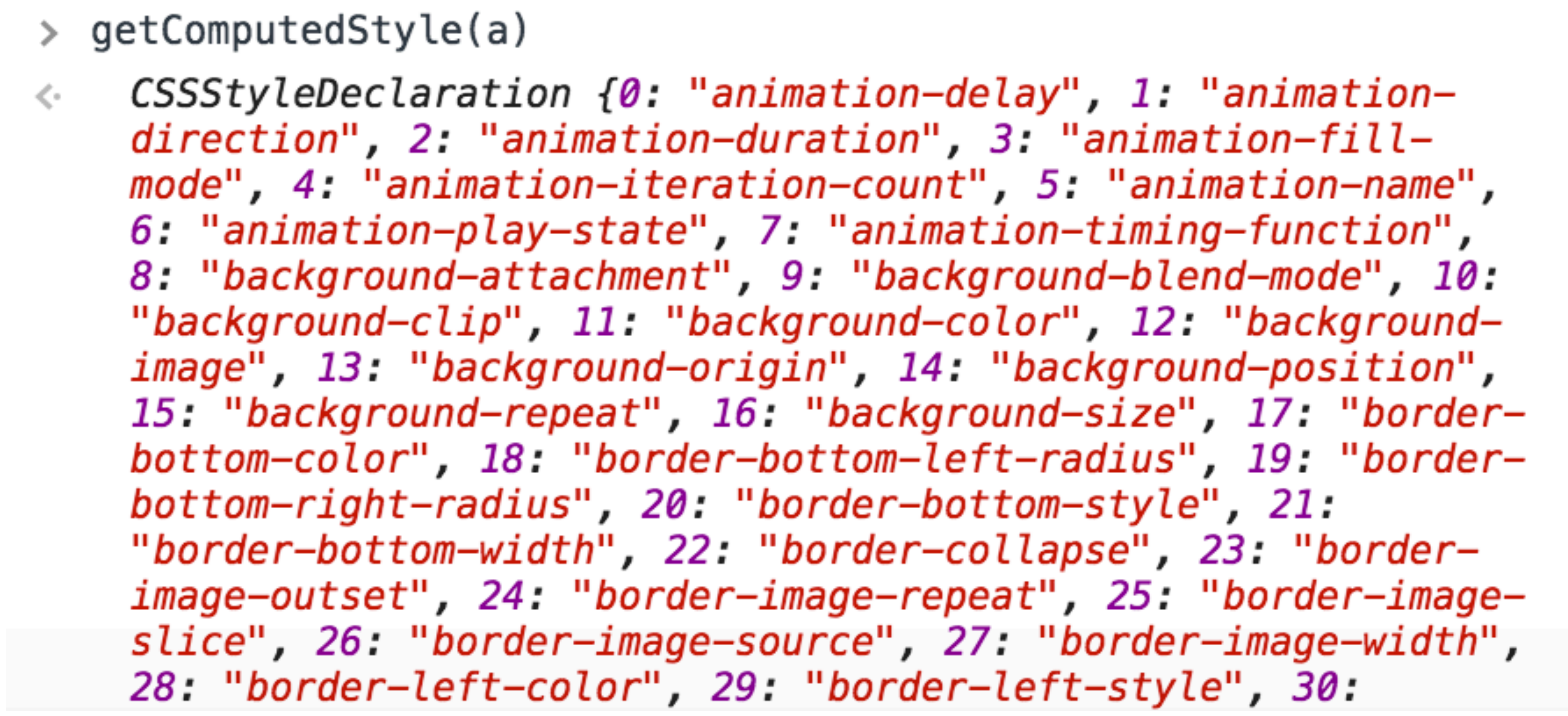
- source order
- selector specificity (element, class/attributes, nesting, id’s)
- element.styles
- !important (avoid like the plague)

- css queries (like mediaqueries) do not add specificity

Reading styles



```
> a.getBoundingClientRect()
<· ▼ClientRect {top: 12, right: 22.890625, bottom: 24, left:
   10.890625, width: 12…} i
      bottom: 24
      height: 12
      left: 10.890625
      right: 22.890625
      top: 12
      width: 12
    ▶ __proto__: ClientRect
```



```
> getComputedStyle(a)
<·  CSSStyleDeclaration {0: "animation-delay", 1: "animation-
    direction", 2: "animation-duration", 3: "animation-fill-
    mode", 4: "animation-iteration-count", 5: "animation-name",
    6: "animation-play-state", 7: "animation-timing-function",
    8: "background-attachment", 9: "background-blend-mode", 10:
    "background-clip", 11: "background-color", 12: "background-
    image", 13: "background-origin", 14: "background-position",
    15: "background-repeat", 16: "background-size", 17: "border-
    bottom-color", 18: "border-bottom-left-radius", 19: "border-
    bottom-right-radius", 20: "border-bottom-style", 21:
    "border-bottom-width", 22: "border-collapse", 23: "border-
    image-outset", 24: "border-image-repeat", 25: "border-image-
    slice", 26: "border-image-source", 27: "border-image-width",
    28: "border-left-color", 29: "border-left-style", 30:
```

Beware that these are perf-heavy - cache the readings if possible

yes, it is a little messy (layout is a black box - about to open up)

Adding style

- on the element by element.style[prop]=value; (be aware of camelCase vs dash-case)

ClassNames are your friend

- easy to manage with element.classList
- just one level of specificity
- Many philosophies like BEM, SMACSS

Animations

- CSS transition
- CSS animation (keyframes)
- Javascript animations (w. requestAnimationFrame)

Response

Animation

Idle

Load

Only animate transform and opacity

95% of the times, you can rethink you animation to be just that

Scrolling is animation

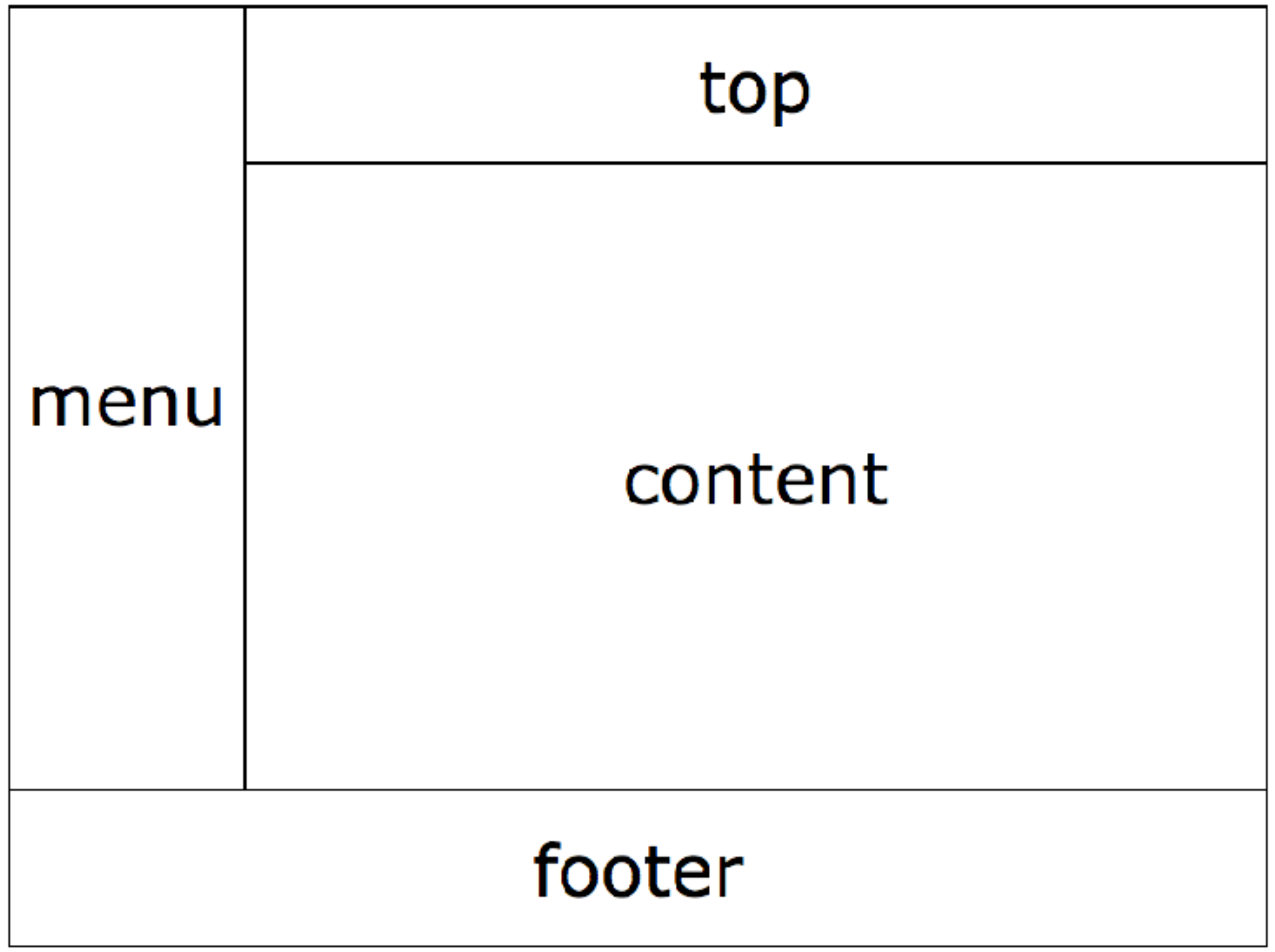
- Think about what you really NEED to do while you are scrolling
- (always use passive event-listeners)

a note about html-comments

- don’t use them (unless you have very specific reasons to)
- keep you notes on the server

Coding time

- Build a form from a configuration object
  - http://www.json-generator.com/api/json/get/bSWyCNGYeq
- Build this layout in as many ways as possible
- make a cat fly in circles with css-animation and with js-animation



lets talk about events

this is what makes you app possible

Events

- bubbles up
- can be caught and stopped by any ancestor in the tree
- default actions
- the browser fires events - you can fire events - the user can initiate events

Listening for events

- by attribute <button onclick=“doSomething()”>
- by property (buttonElement.onClick=doSomething)
- by adding an eventListener (buttonElement.addEventListener(‘click’, doSomething);

best practices for events

- use addEventListener
- remember to remove them again (due to memory - GC)
- debounce (some events fire quite often)
- use passive event-listeners as a performance enhancer (disables the option to preventDefault)

the event-object

- target
- which
- stopPropagation()
- preventDefault()
- (log to console to inspect)

the event loop

- javascript is syncronous and one-threaded (same as UI-thread)
- async callbacks (events, promises, timeouts, worker-messages, xhr, etc) are added to the bottom of the queue
- (almost) All new APIs are async for perf reasons

timing

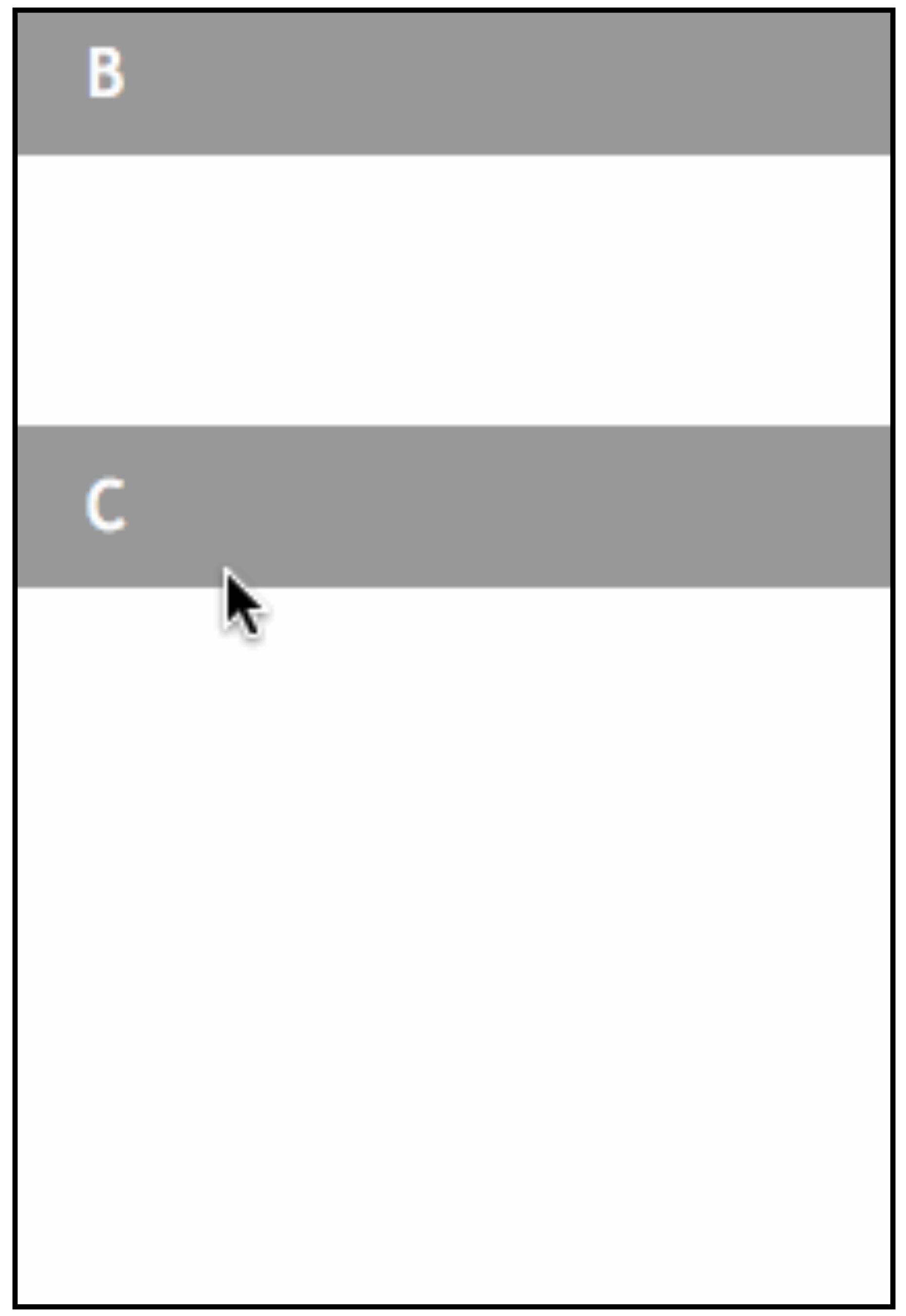
- setTimeout and clearTimeout

- setInterval and clearInterval

put something at the end of the event-loop by using setTimeout with 0 ms

actual coding

- drag-drop a div around with your mouse
- overlapping fixed headers (like contacts on iOS)
- Build a function to load an external script that returns a promise that resolves when the script has loaded



when the DOM is weird, different, hard

- browser inconsistencies
- iframes
- shadow-DOM

thats it!

thanks for having me - im @filipbech