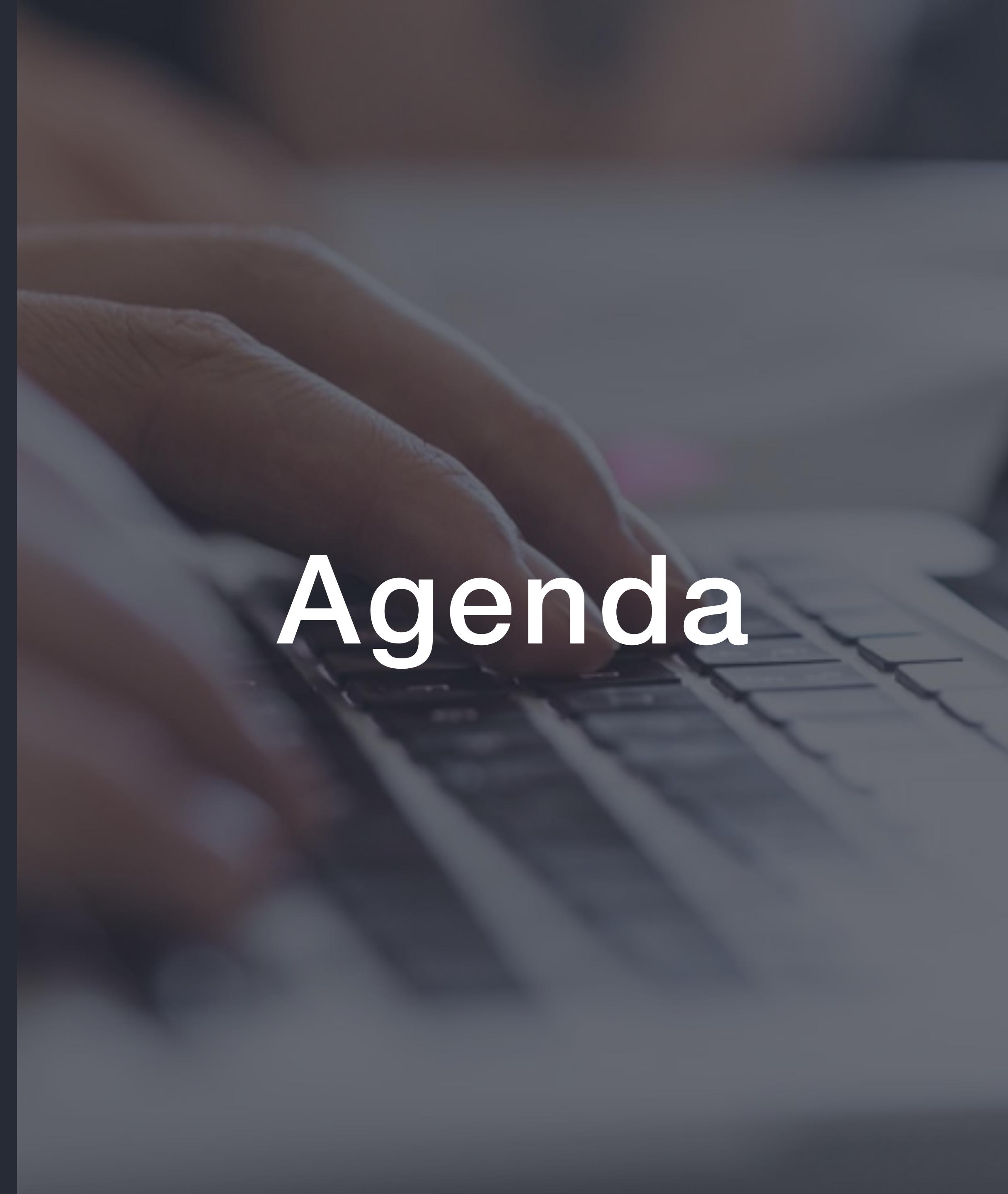


1. Semester - JavaScript

Client-Server, Variabler, Datatyper, Template String, Objects, Arrays, Funktioner, DOM-Manipulation, CSS Grid & Fetch

1. Intro til RACE & JavaScript
2. Hvad er det nu vi skal med dig? Movie App
3. Client-Server
4. Variabler & Datatyper
5. Arrays & Objekter
6. Funktioner
7. DOM-Manipulation



Agenda

Movie App

Seg film...

Allé gener

Sortér: Titel (A-Å)

Barbie (2023)
Adventure, Comedy, Fantasy
★★ 7
Director: Greta Gerwig

Dune (2021)
Adventure, Drama, Sci-Fi
★★ 8
Director: Denis Villeneuve

Dune: Part Two (2024)
Action, Adventure, Drama
★★ 8.7
Director: Denis Villeneuve

Everything Everywhere All at Once (2022)
Action, Adventure, Comedy
★★ 7.8
Director: Daniel Kwan, Daniel Scheinert

Fight Club

Tom Hanks Is

The Dream Is Real

Movie App

Seg film...

Allé gener

Sortér: År (neste)

Barbie (2023)
Adventure, Comedy, Fantasy
★★ 7
Director: Greta Gerwig
Actors: Margot Robbie, Ryan Gosling, America Ferrera

Barbie and Ken embark on a journey of self-discovery after leaving the utopian Barbie Land for the real world.

Inception (2010)
Action, Adventure, Sci-Fi
★★ 8.8
Director: Christopher Nolan

The Matrix (1999)
Action, Sci-Fi
★★ 8.7
Director: Lana Wachowski, Lilly Wachowski

Movie App

Seg film...

Allé gener

Sortér: Titel (A-Å)

Dune

Dune (2021)
Adventure, Drama, Sci-Fi
★★ 8
Director: Denis Villeneuve

Dune: Part Two (2024)
Action, Adventure, Drama
★★ 8.7
Director: Denis Villeneuve

Movie App

Seg film...

Sci-Fi

Sortér: År (neste)

Dune (2021)
Adventure, Drama, Sci-Fi
★★ 8
Director: Denis Villeneuve

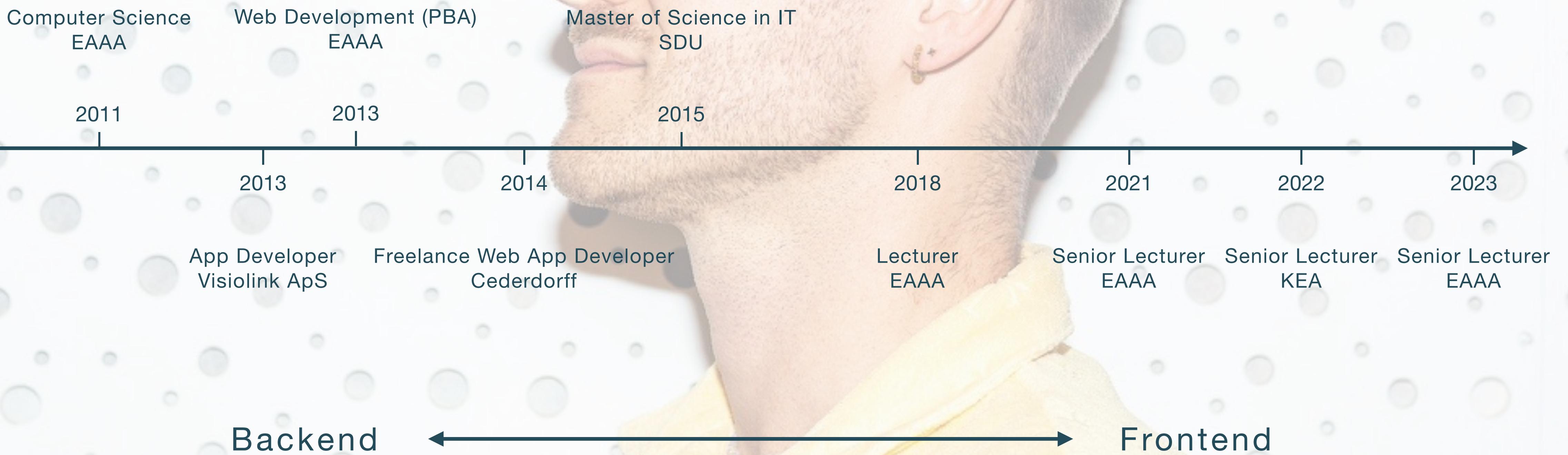
Inception (2010)
Action, Adventure, Sci-Fi
★★ 8.8
Director: Christopher Nolan

The Matrix (1999)
Action, Sci-Fi
★★ 8.7
Director: Lana Wachowski, Lilly Wachowski

I'm Rasmus Cederdorff (RACE)
Senior Lecturer
Freelance Web App Developer

- Programming with an eye for UI and UX.
- Web Development, JavaScript, React, BaaS & Node.js
- “I speak JavaScript”.
- Websites, Webshops, Web Apps, Mobile Apps, Server App and BaaS.







I'm Rasmus Cederdorff
Holstebro
Alicia & Ida

From Holstebro
I'm into sports
Love (apple) gadgets, to take
pictures & interior design
projects

What's with my arm?





**“Learning to code requires a HUGE
investment of time and energy.”**

<https://medium.com/martinssoft/learn-javascript-c1cca9db9015>

“

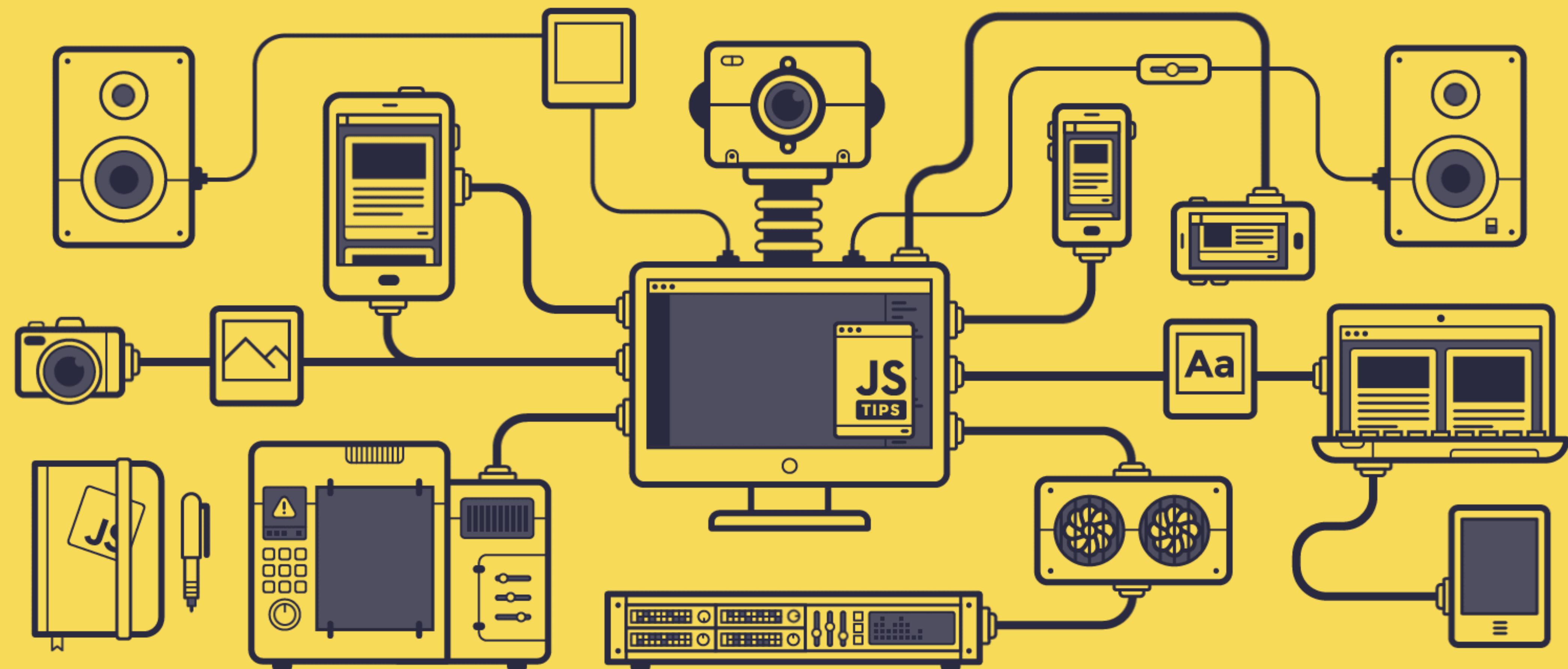
The cool thing about JavaScript is
that you can create stuff that will put
a smile on your face, as a developer.

You can create stuff and it will
visually look like something, and it'll
do stuff, and it makes you feel good,
it makes you fall in love with
programming.

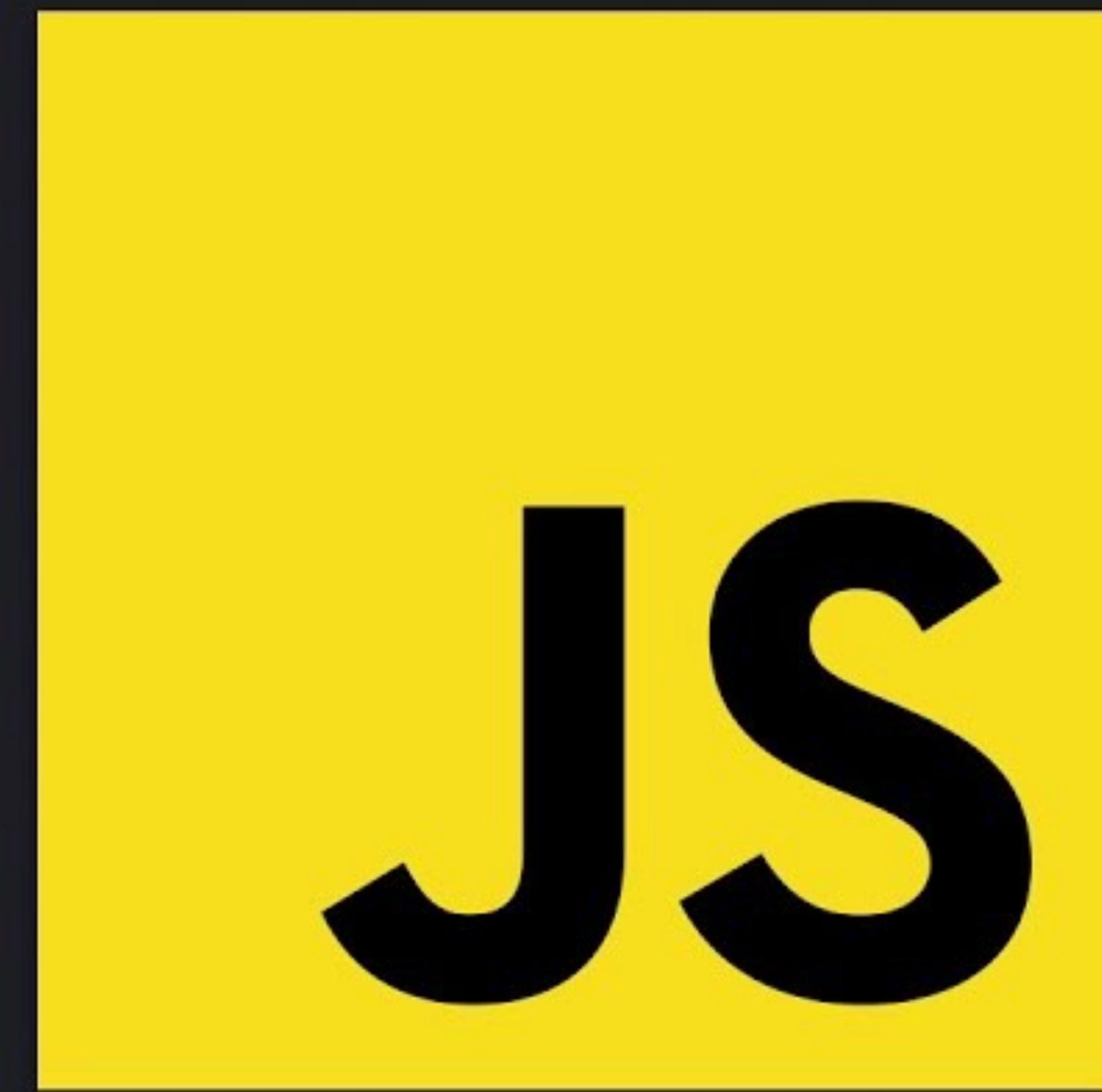
”

Lex Fridman
Computer Scientist

<https://www.youtube.com/watch?v=GLhyjVZp0cw&t=252s>



100 *SECONDS OF*



<https://www.youtube.com/watch?v=DHjqpvDnNGE>

JS

101



<https://www.youtube.com/watch?v=IkIFF4maKMU>

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Page Title</title>
5      <link rel="stylesheet" href="styles.css" />
6    </head>
7    <body>
8      <h1>This is a Heading</h1>
9      <p>This is a paragraph.</p>
10     <button onclick="tryMe()">Try me</button>
11     <script src="app.js"></script>
12   </body>
13 </html>
14
```

What is JavaScript?

.. is the world's most popular programming language.

... is the programming language of the Web.

... is easy to learn.

... can change content of a webpage (HTML content).

... can change styling of HTML.

```
1  function tryMe() {
2    document.body.style.backgroundColor = "red";
3    document.body.style.color = "white";
4  }
5
```

index.html x

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Page Title</title>
5      <link rel="stylesheet" href="styles.css" />
6    </head>
7    <body>
8      <h1>This is a Heading</h1>
9      <p>This is a paragraph.</p>
10     <button onclick="tryMe()">Try me</button>
11     <script src="app.js"></script>
12   </body>
13 </html>
```

With JavaScript we are able to

... build dynamic web pages and web apps.

... fetch content/ data from a backend (web service, data source, etc.) through an API.

app.js x

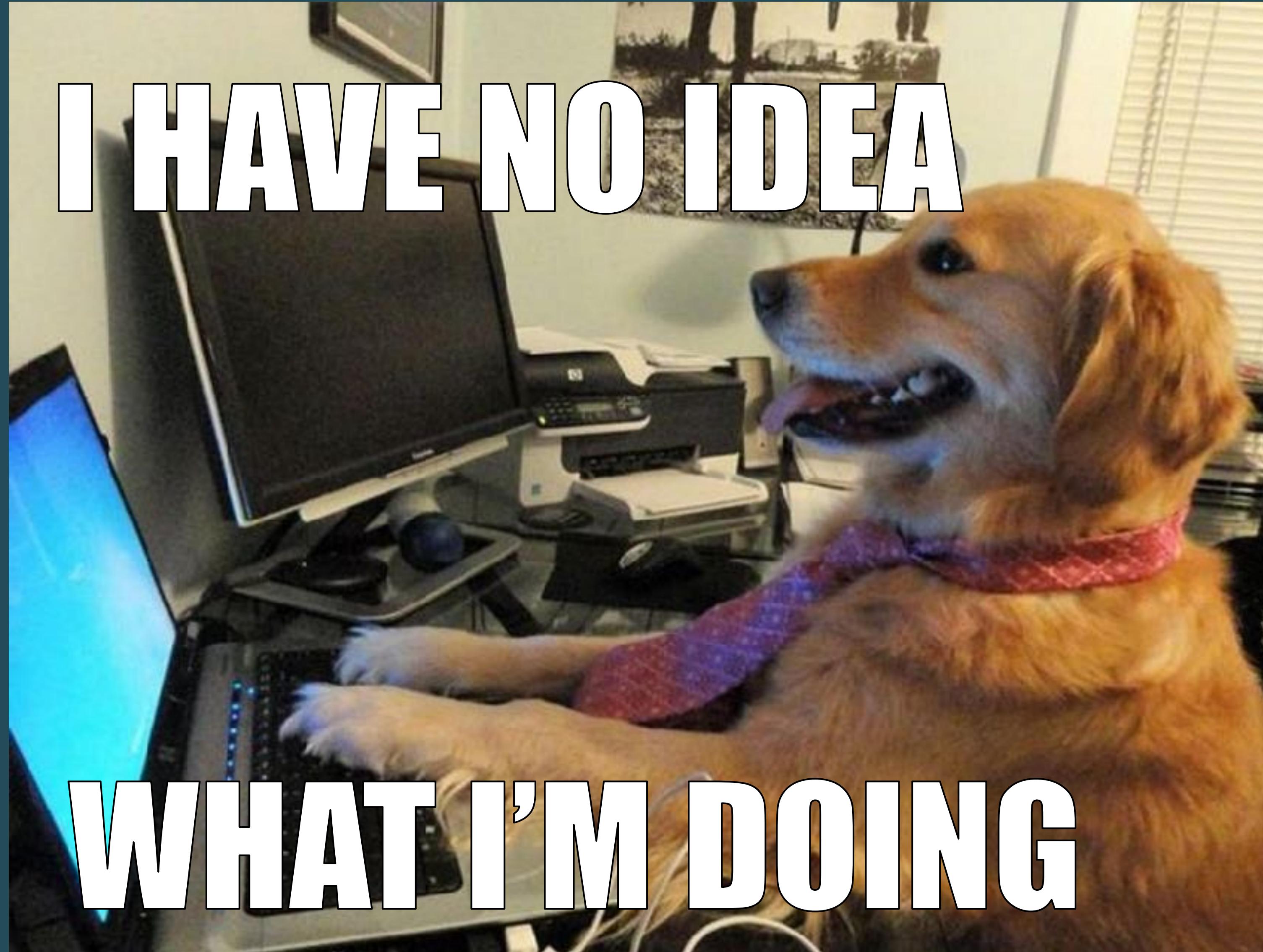
```
1  function tryMe() {
2    document.body.style.backgroundColor = "red";
3    document.body.style.color = "white";
4  }
5
```

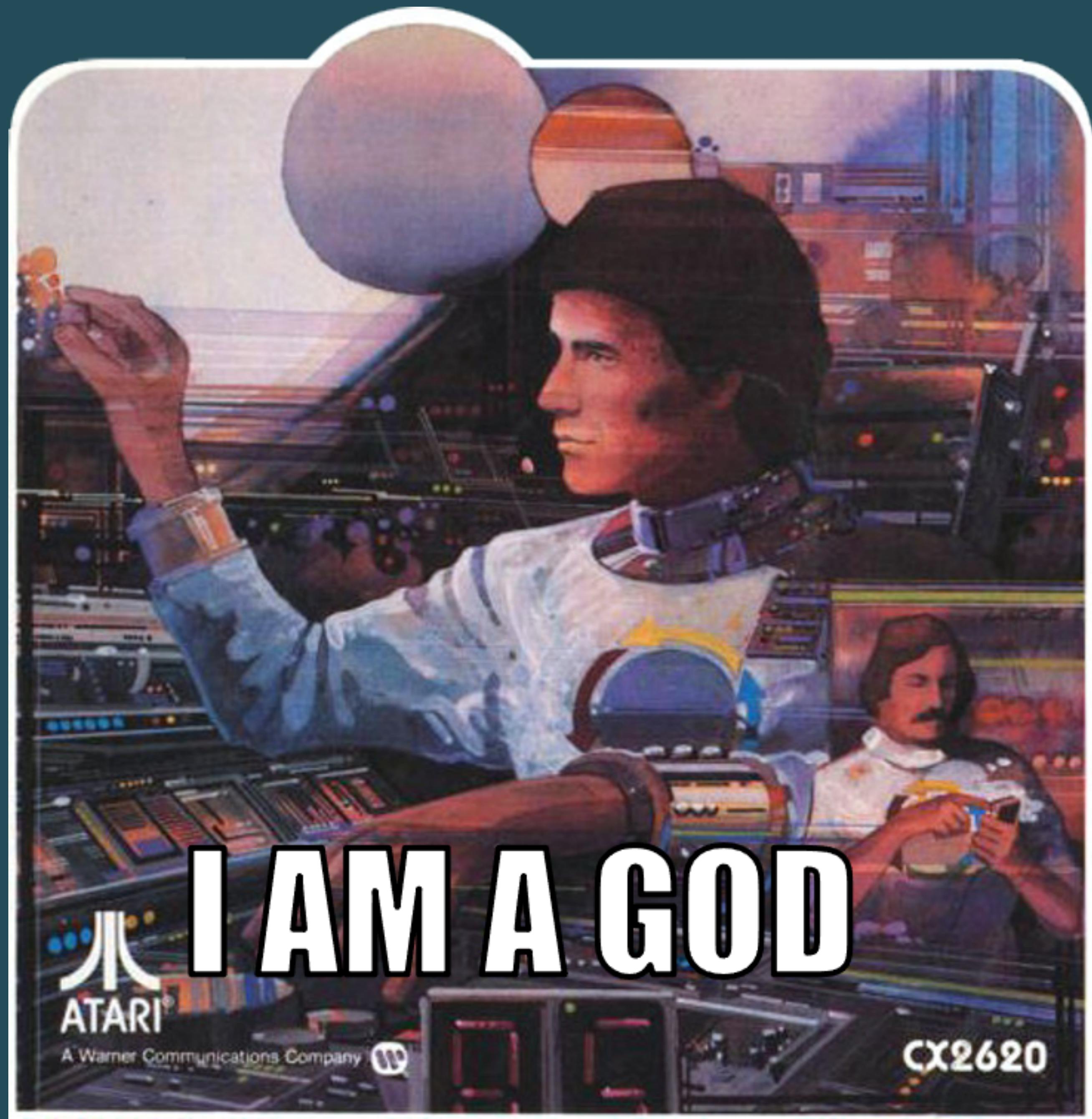
... do DOM-manipulation.

... build and develop anything 

I HAVE NO IDEA

WHAT I'M DOING



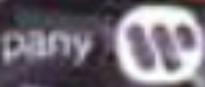


I AM A GOD



ATARI

A Warner Communications Company

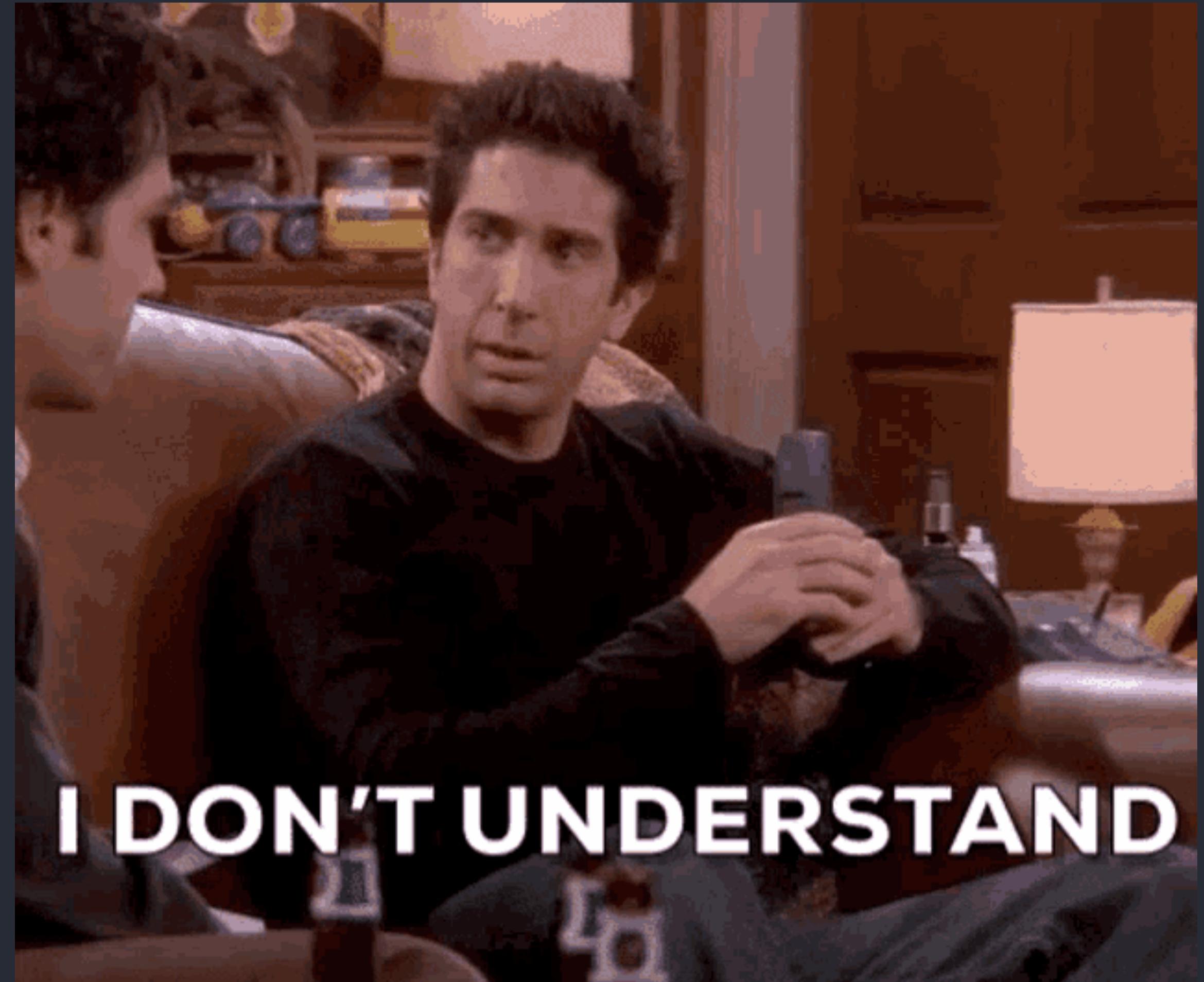


CX2620

JUST DO IT.



Please,
Do not try to
understand
everything!



I DON'T UNDERSTAND

Accept the gaps

– understanding grows with practice

Chef on TV



Real Chef



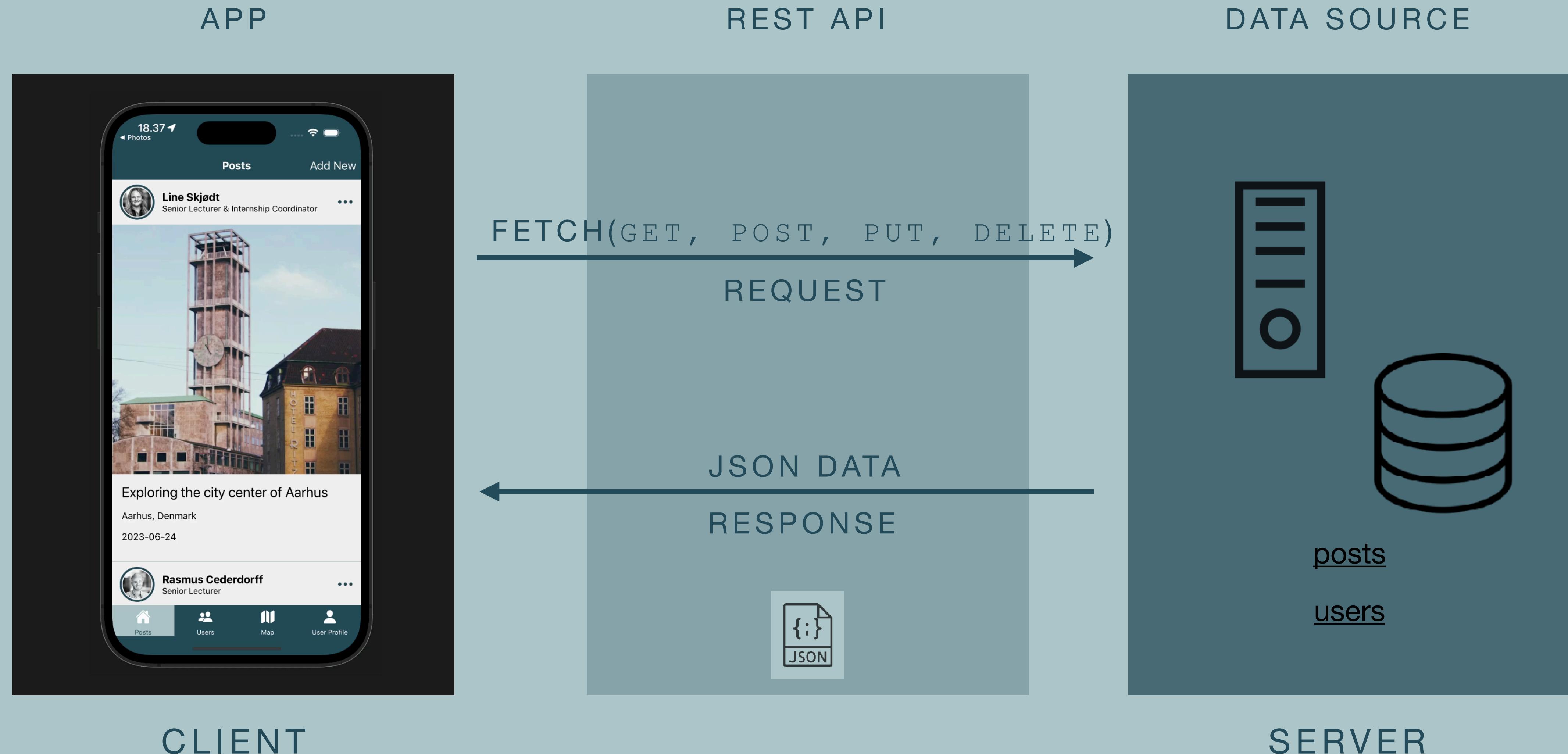
IxD - Interaction Design

Se det som en smagsprøve på IxD

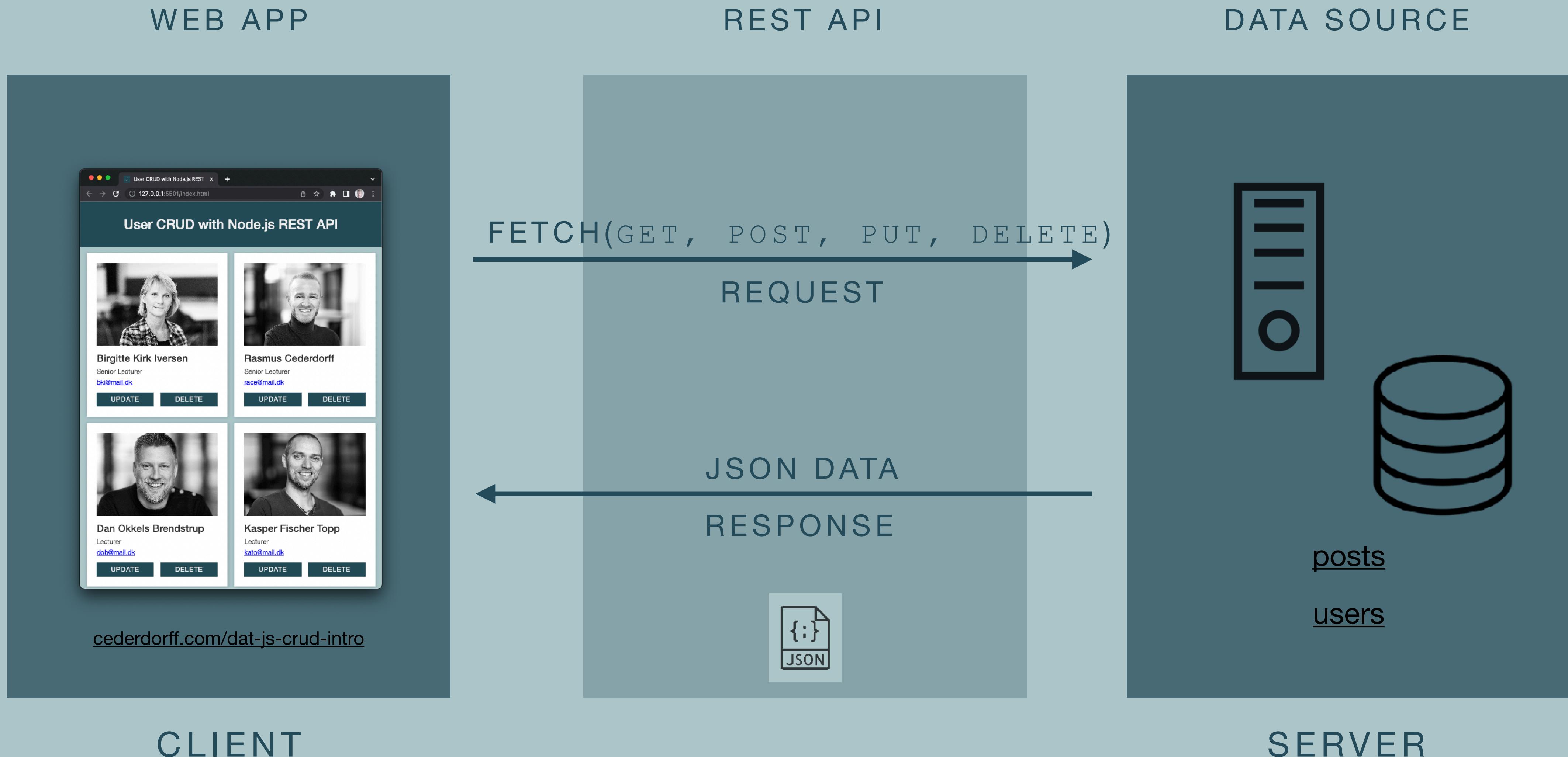
Client Server

The Basic Architecture of the Web

Client Server Architecture



Web Development

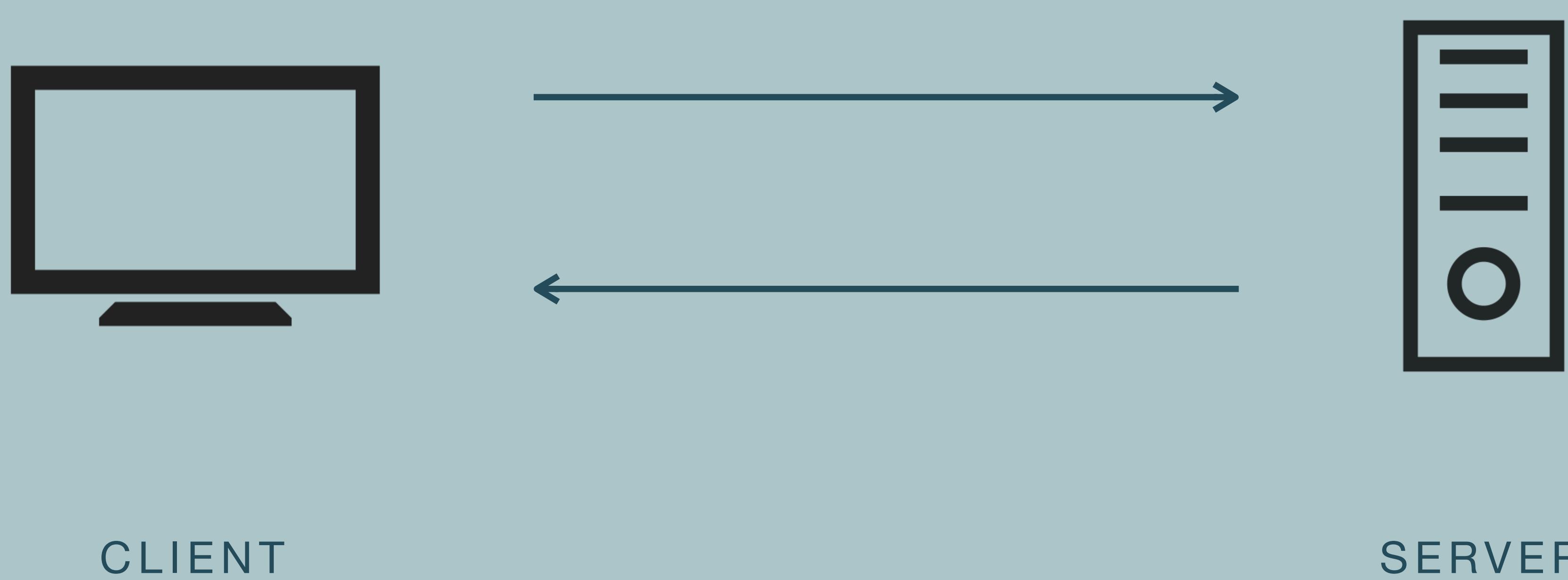


Webudvikling



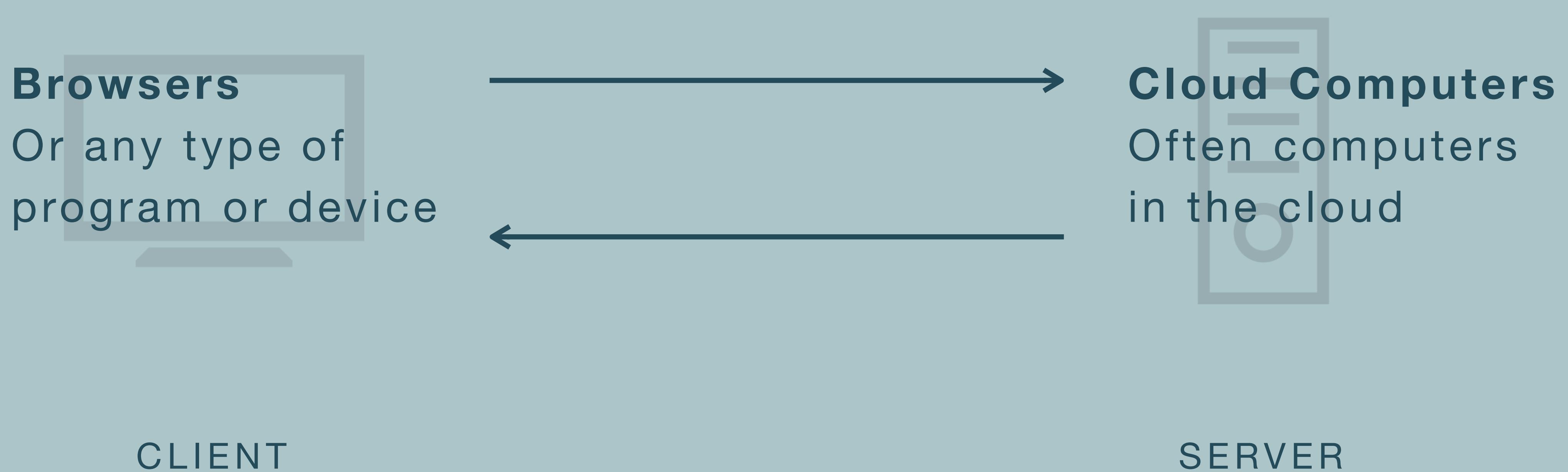
Client-Server Model

Communication between web **clients** and web **servers**.



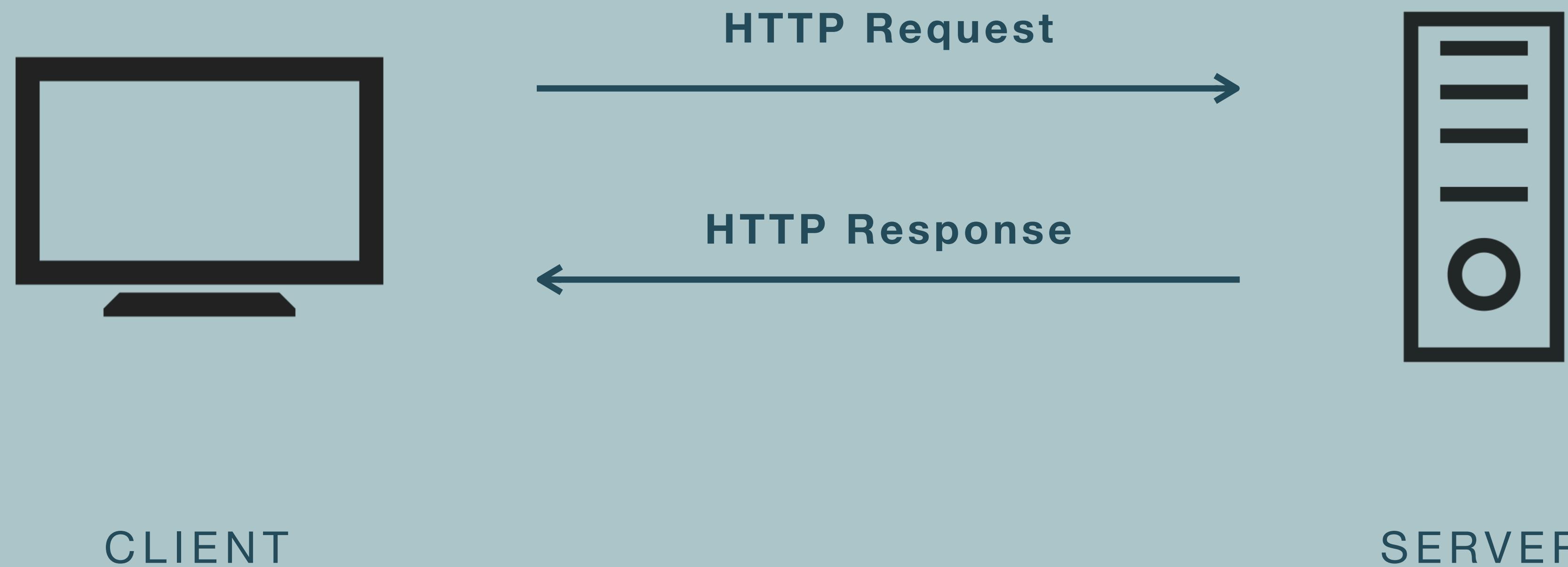
Client-Server Model

Communication between web **clients** and web **servers**.



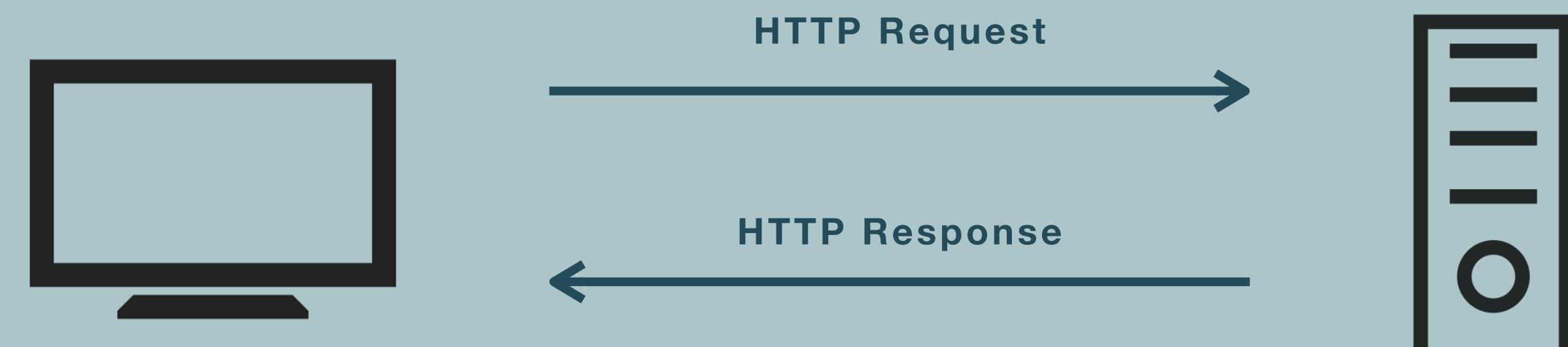
Client-Server Model

Communication between web **clients** and web **servers**.



Hyper Text Transfer Protocol

- A protocol and standard for fetching data, HTML and other resources (text, images, videos, scripts, JSON).
- The foundation of the web.



What is HTTP

Not Secure | w3schools.com/whatis/whatis_http.asp

HTML CSS JAVASCRIPT SQL PYTHON

HTTP Request / Response

Communication between clients and servers is done by **requests** and **responses**:

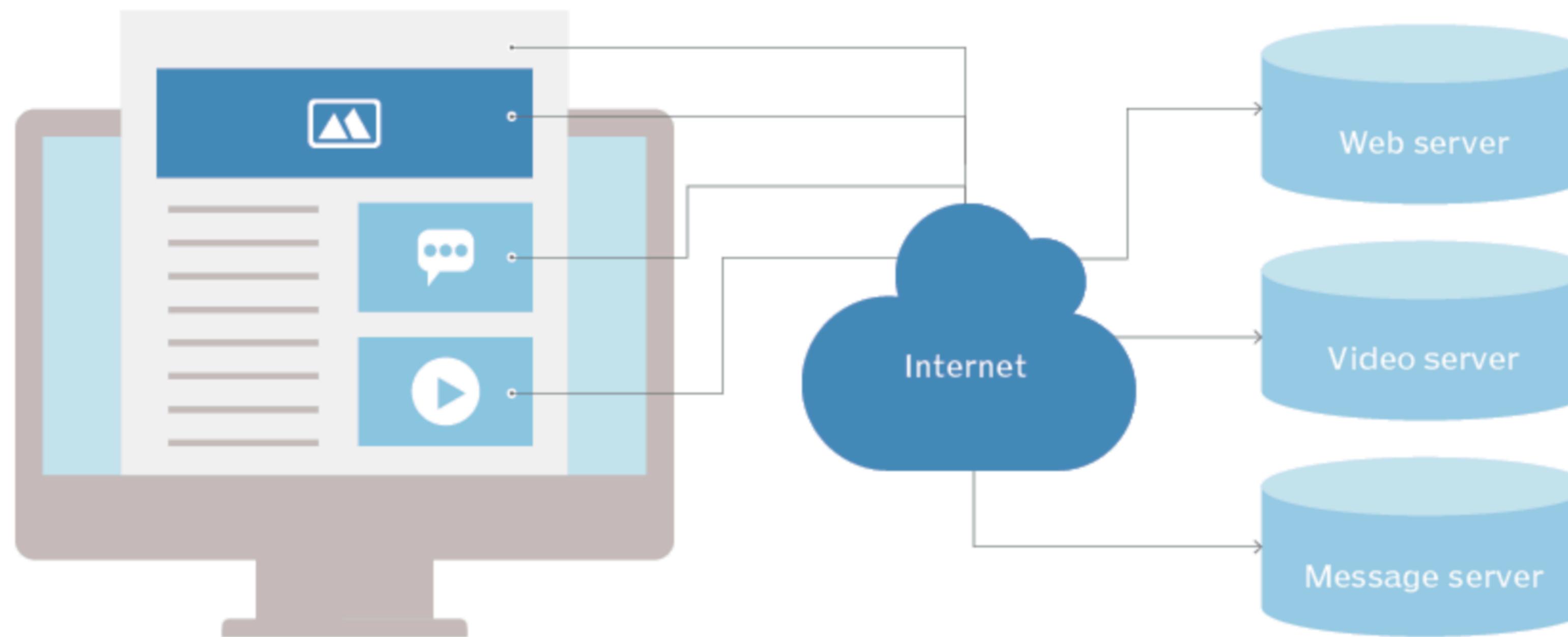
1. A client (a browser) sends an **HTTP request** to the web
2. A web server receives the request
3. The server runs an application to process the request
4. The server returns an **HTTP response** (output) to the browser
5. The client (the browser) receives the response

The HTTP Request Circle

A typical HTTP request / response circle:

1. The browser requests an HTML page. The server returns an HTML file.
2. The browser requests a style sheet. The server returns a CSS file.
3. The browser requests an JPG image. The server returns a JPG file.
4. The browser requests JavaScript code. The server returns a JS file
5. The browser requests data. The server returns data (in XML or JSON).

How HTTP Works



<https://www.techtarget.com/whatis/definition/HTTP-Hypertext-Transfer-Protocol>

Network Tab

The screenshot illustrates the Network tab of a browser's developer tools, overlaid on a live website. The website's content includes a header for 'ERHVERVSAKADEMI AARHUS', a main image of two people working, and three cards at the bottom: 'Uddannelser til erhvervslivet', 'Videregående uddannelser', 'Efteruddannelse og kurser', and 'Samarbejde og s...

Name	Method	Status	Type	Initiator	Size	T..	Waterfall
1.gif?dgi=70fb8fd...	GET	200	gif	uc.js?cbid...	(me... 0...		
heatmaps.js	GET	200	script	monsido-s...	(disk... 2...		
page-correct.js	GET	200	script	monsido-s...	(disk... 2...		
?a=h0r3JOS3pvXaDa...	GET	200	gif	monsido-s...	57 B 1...		
favicon.ico	GET	200	x-icon	Other	(disk... 1...		
h0r3JOS3pvXaDabSjt...	GET	200	xhr	heatmaps....	(disk... 0...		
h0r3JOS3pvXaDabSjt...	GET	200	xhr	page-corr...	(disk... 0...		
cast_sender.js?loadC...	GET	200	script	vendor.mo...	(disk... 0...		
1765804027-75aafef...	GET	200	avif	Other	(disk... 0...		
master.json?base64_i...	GET	200	xhr	vendor.mo...	3.0 kB 1...		
1765804027-75aafef...	GET	200	avif	vendor.mo...	(me... 0...		
cast_framework.js	GET	200	script	cast_send...	14 B 1...		
cast_sender.js	GET	200	script	cast_send...	(disk... 1...		
settings.json	GET	200	xhr	uc.js?cbid...	(disk... 0...		
widgetIcon.min.js	GET	200	script	uc.js?cbid...	(disk... 0...		
2b51a972.mp4?r=dX...	GET	200	xhr	vendor.mo...	5.4 kB 2...		
d509129e.mp4?r=dX...	GET	200	xhr	vendor.mo...	264 kB 3...		
cf6acf66.mp4?r=dXM...	GET	200	xhr	vendor.mo...	4.3 ... 2...		
2b51a972.mp4?r=dX...	GET	200	xhr	vendor.mo...	5.4 kB 1...		
2b51a972.mp4?r=dX...	GET	200	xhr	vendor.mo...	5.1 kB 2...		
cf6acf66.mp4?r=dXM...	GET	200	xhr	vendor.mo...	4.6 ... 2...		
cf6acf66.mp4?r=dXM...	GET	200	xhr	vendor.mo...	3.5 ... 3...		
collect?v=2&tid=G-D...	POST	204	ping	js?id=G-D...	17 B 6...		

91 requests | 12.8 MB transferred | 16.2 MB resources | Finish: 5.47 s | DOMContentLoaded: 290 ms

Network Tab

The screenshot shows a web browser window for 'Erhvervsakademi Aarhus - EA' at 'eaaa.dk'. The main content features a large image of a modern building with solar panels and several flags. Overlaid text reads 'Uddannelser til erhvervslivet' and 'Skal du vælge videregående uddannelse? Eller finde det kursus, der løfter din karriere? Tal med os. Vi er et af landets største erhvervsakademier.' Below this are three smaller images: a woman smiling, two people talking at a desk, and a person's profile.

The browser's developer tools Network tab is open, showing a list of requests. One request is selected, displaying its details:

```
▼ {type: "video", version: "1.0", provider_name: "Vimeo", provider_url: "https://vimeo.com/user209157220", account_type: "starter", author_name: "Erhvervsakademi Aarhus", author_url: "https://vimeo.com/user209157220", description: "", duration: 17, height: 240, html: "<div style='padding:56.25% 0 0 0;position:relative;'><div><img alt='Silent video D4' data-vimeo-player='1' data-vimeo-player-type='video' data-vimeo-player-width='426' data-vimeo-player-height='240' data-vimeo-player-embed-id='1765804027-75aaafef51' data-vimeo-player-embed-type='video' data-vimeo-player-embed-allowscript='1' data-vimeo-player-embed-allowfullscreen='1' data-vimeo-player-embed-allowmuted='1' data-vimeo-player-embed-allowplaybackrate='1' data-vimeo-player-embed-allowcontroll... is_plus: "0", provider_name: "Vimeo", provider_url: "https://vimeo.com/", thumbnail_height: 166, thumbnail_url: "https://i.vimeocdn.com/video/1765804027-75aaafef51", thumbnail_url_with_play_button: "https://i.vimeocdn.com/filter/ovf/1765804027-75aaafef51", thumbnail_width: 295, title: "Silent video D4", type: "video", upload_date: "2023-12-08 06:54:18", uri: "/videos/892590144", version: "1.0", video_id: 892590144, width: 426}
```

At the bottom of the Network tab, it says '15 / 111 requests | 12'.

**ERHVERVSAKADEMI
AARHUS**

Kom til u-days 20.-22. februar

Besøg vores 29 videregående uddannelser – og bliv klogere på dit studievalg.

Læs om u-days her

Uddannelser til erhvervslivet

Tal du vælge videregående uddannelse? Eller finde det kursus, der løfter din karriere? Tal med os. Vi er et af landets største erhvervsakademier.

102 requests | 12.9 MB transferred | 17.4 MB resources | Finish: 2.59 s | DOMCo

Network Tab

Investigating Network Activity in the Browser

Variabler

const & let

Variables

... are used to store data (values, objects, collections) in the memory

Variables

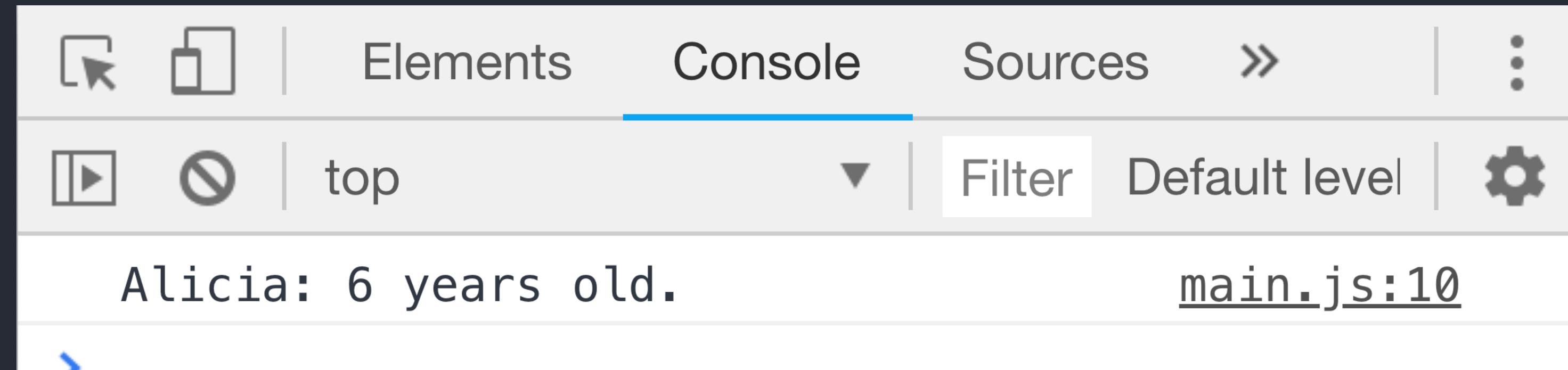
... a simple way to store, get and set data in
your code.

Variables

Store data in the memory

```
let name = "Alicia";
let age = 6;

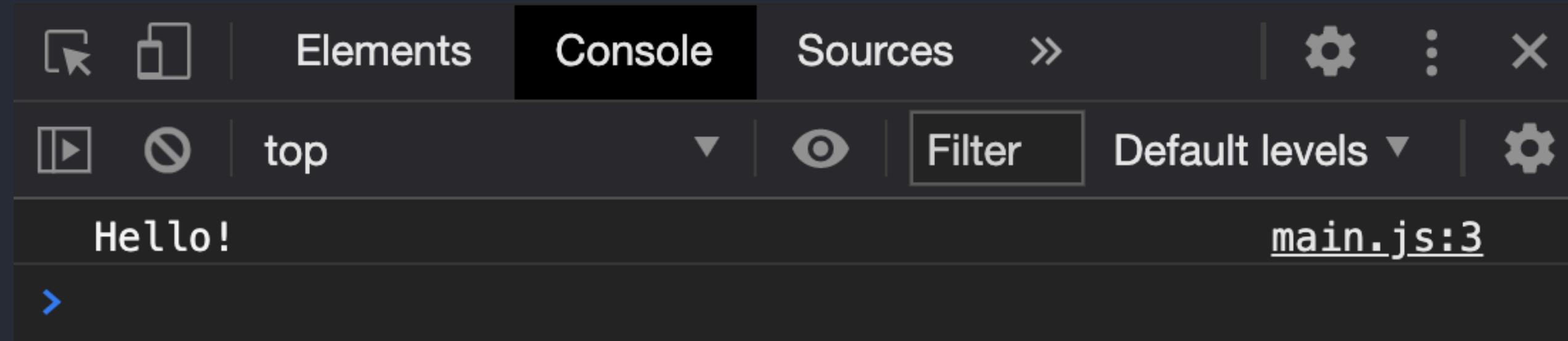
console.log(name + ": " + age + " years old.");
```



Variables

Store data in the memory

```
let message = "Hello!";
console.log(message);
```

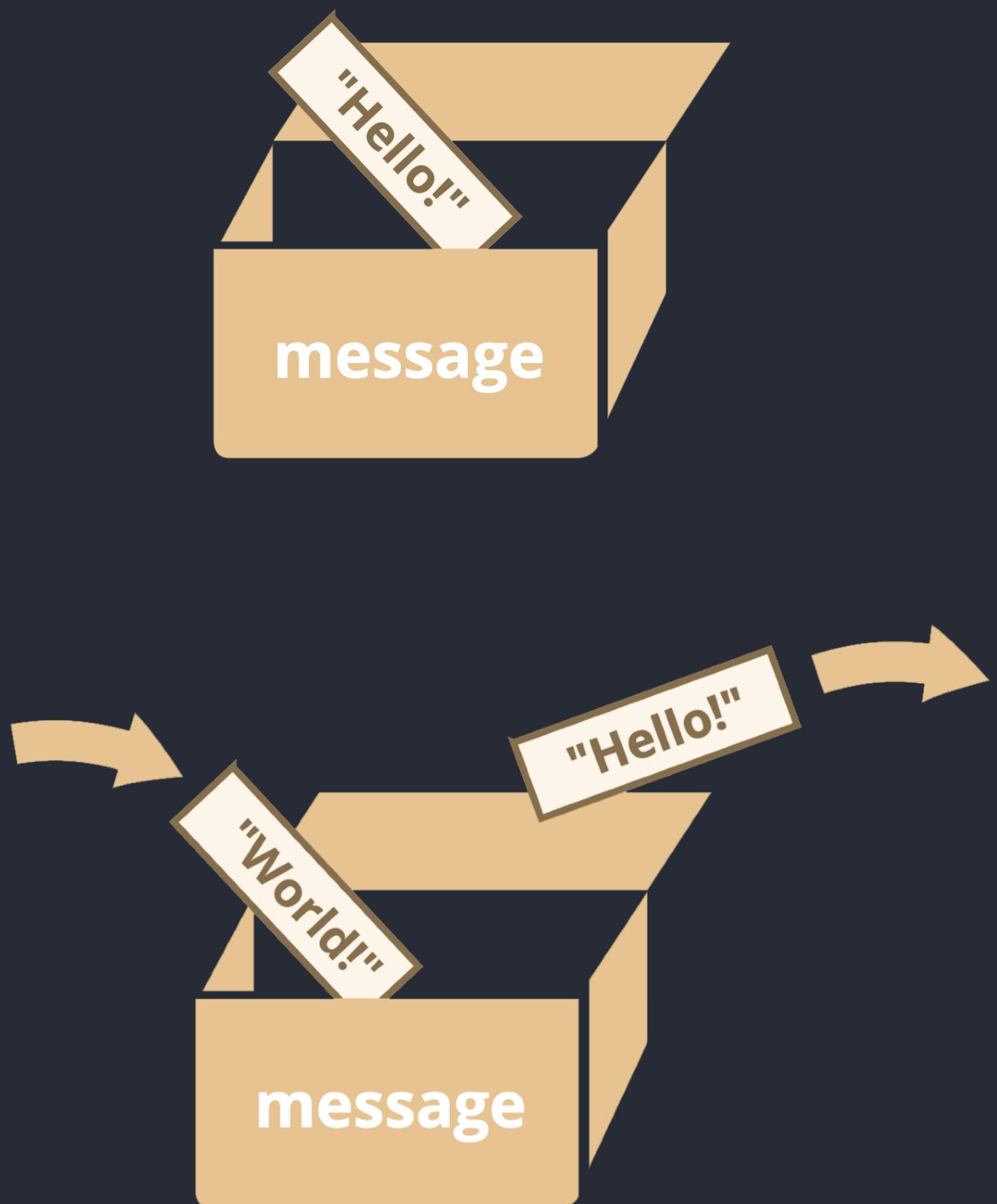


Variables

Store data in the memory

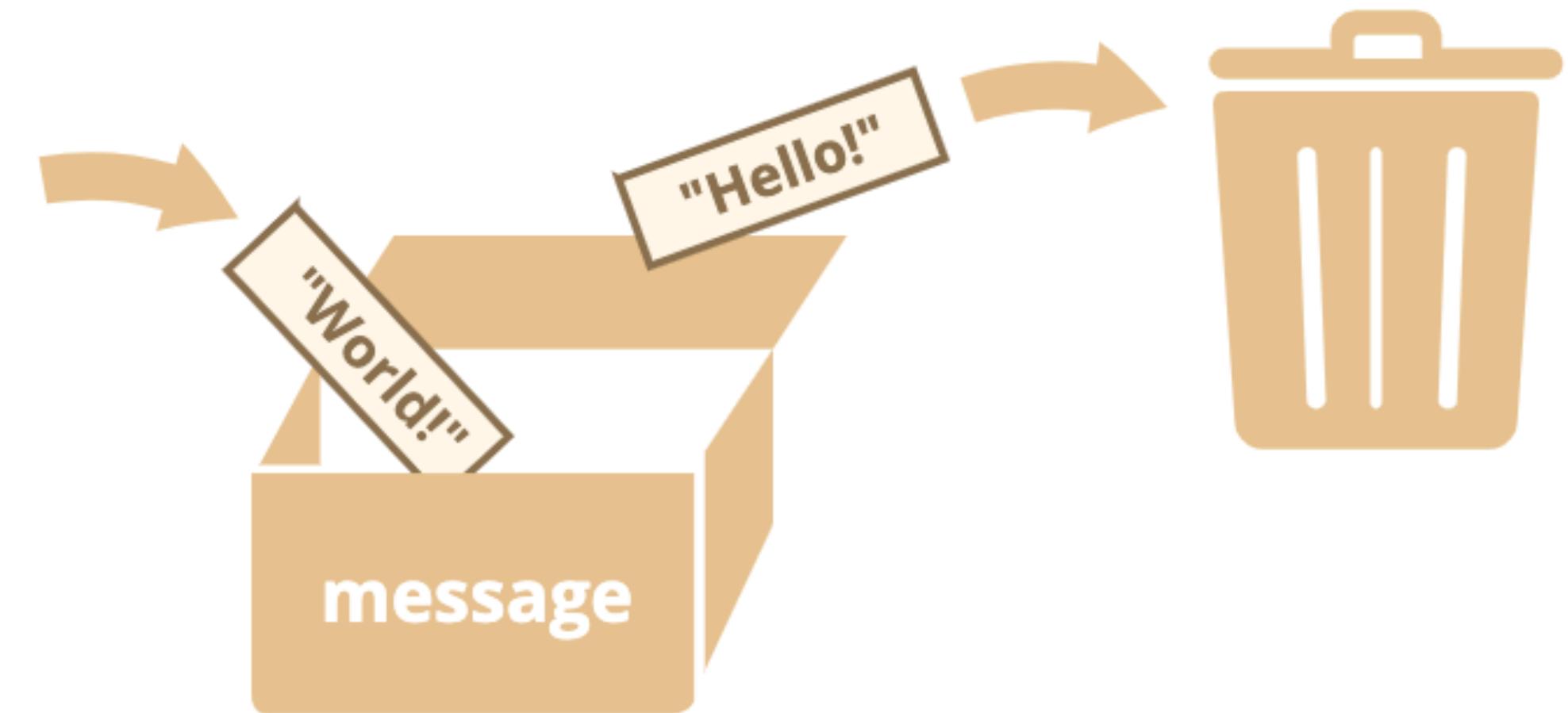
```
let message = "Hello!";
console.log(message);
```

```
message = "World!";
console.log(message);
```



Variable

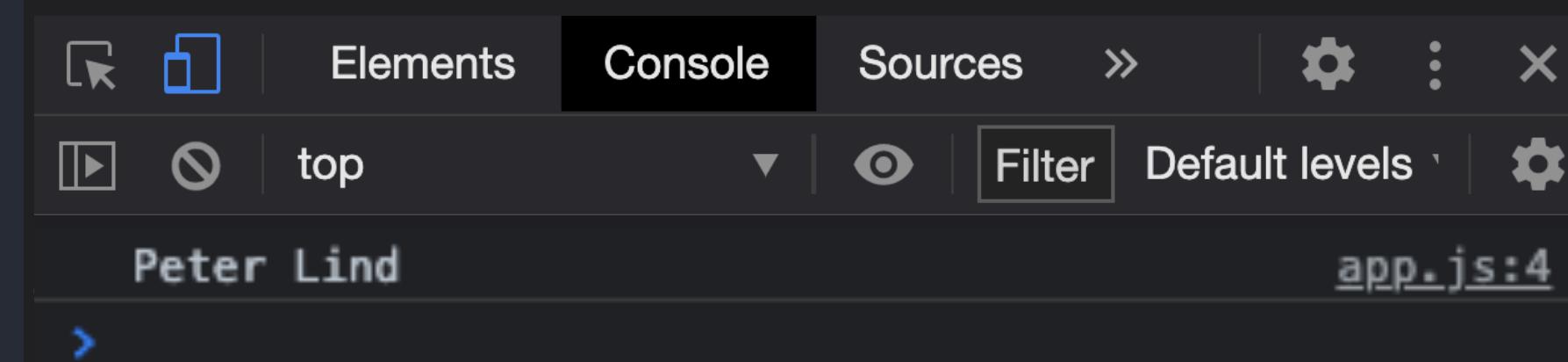
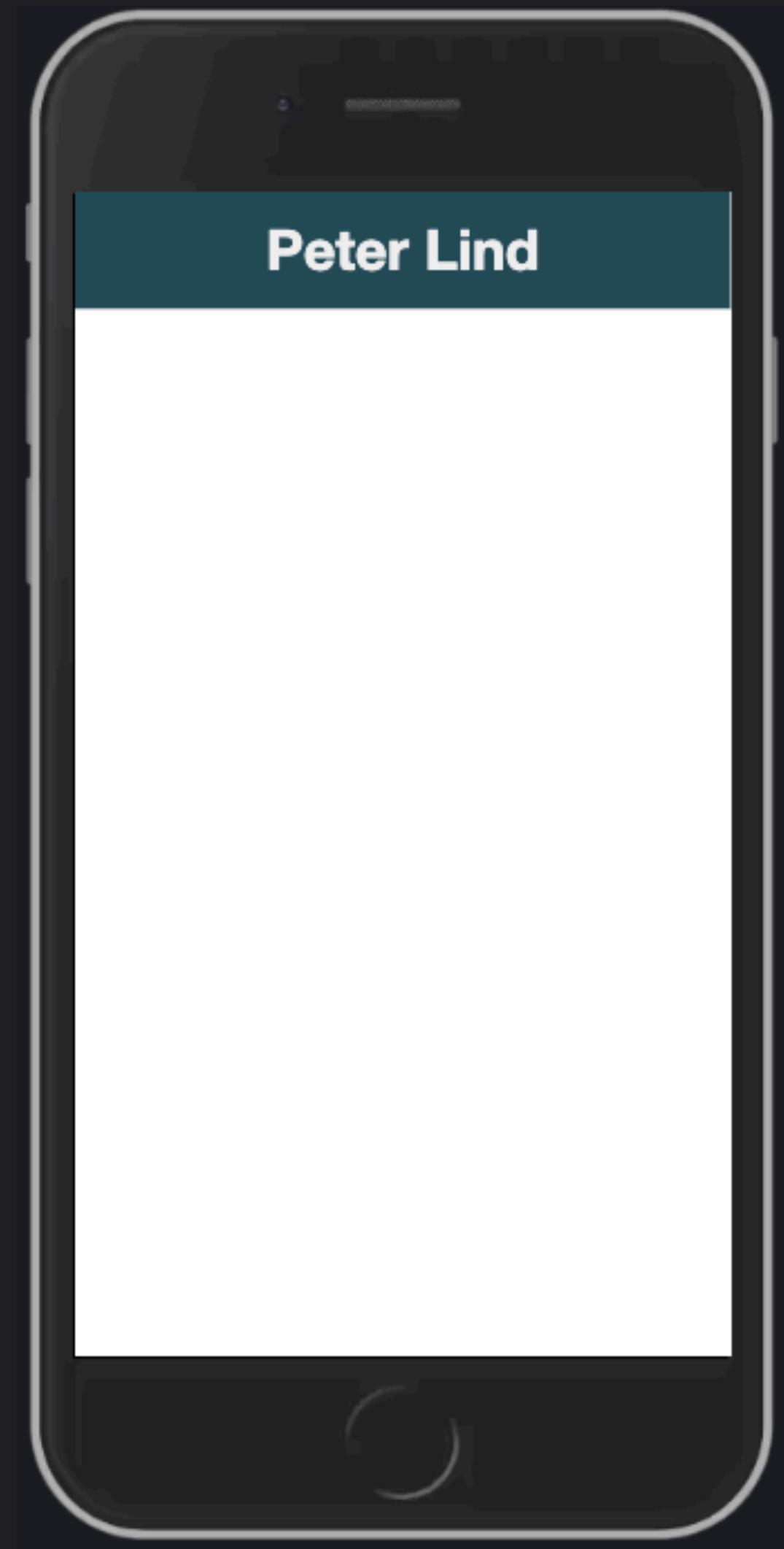
A variable is a “named storage” and stored in the memory of the browser.



We can change the value of the variables as many times as we want.

Variables & UI

```
let fullName = "Peter Lind";
console.log(fullName);
document.querySelector("#fullName_container").textContent = fullName;
```



Variables & UI

```
let fullName = "Peter Lind";
console.log(fullName);
document.querySelector("#fullName_container").textContent = fullName;
```

```
<body>
  <header>
    <h1 id="fullName_container"></h1>
  </header>
  <script src="app.js"></script>
</body>
```



Variables & UI

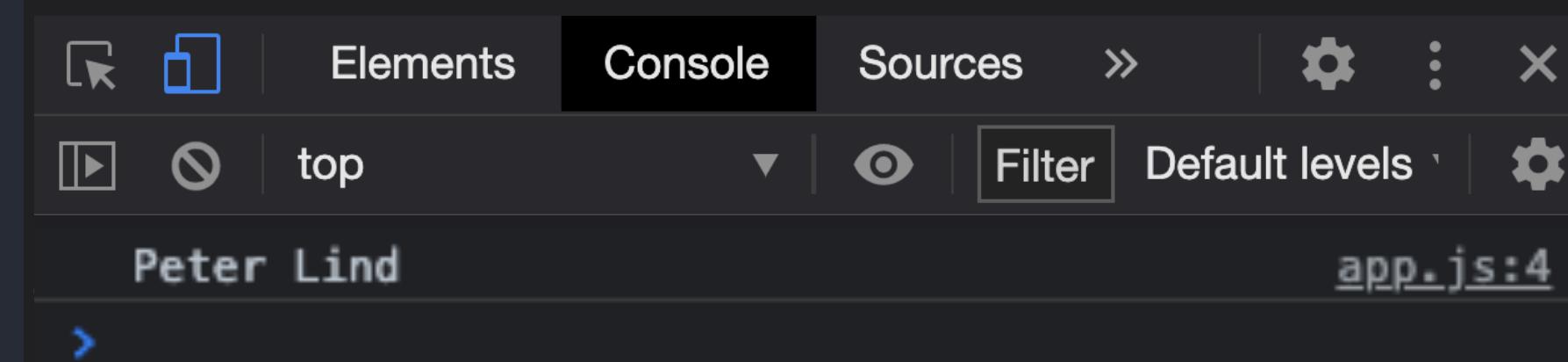
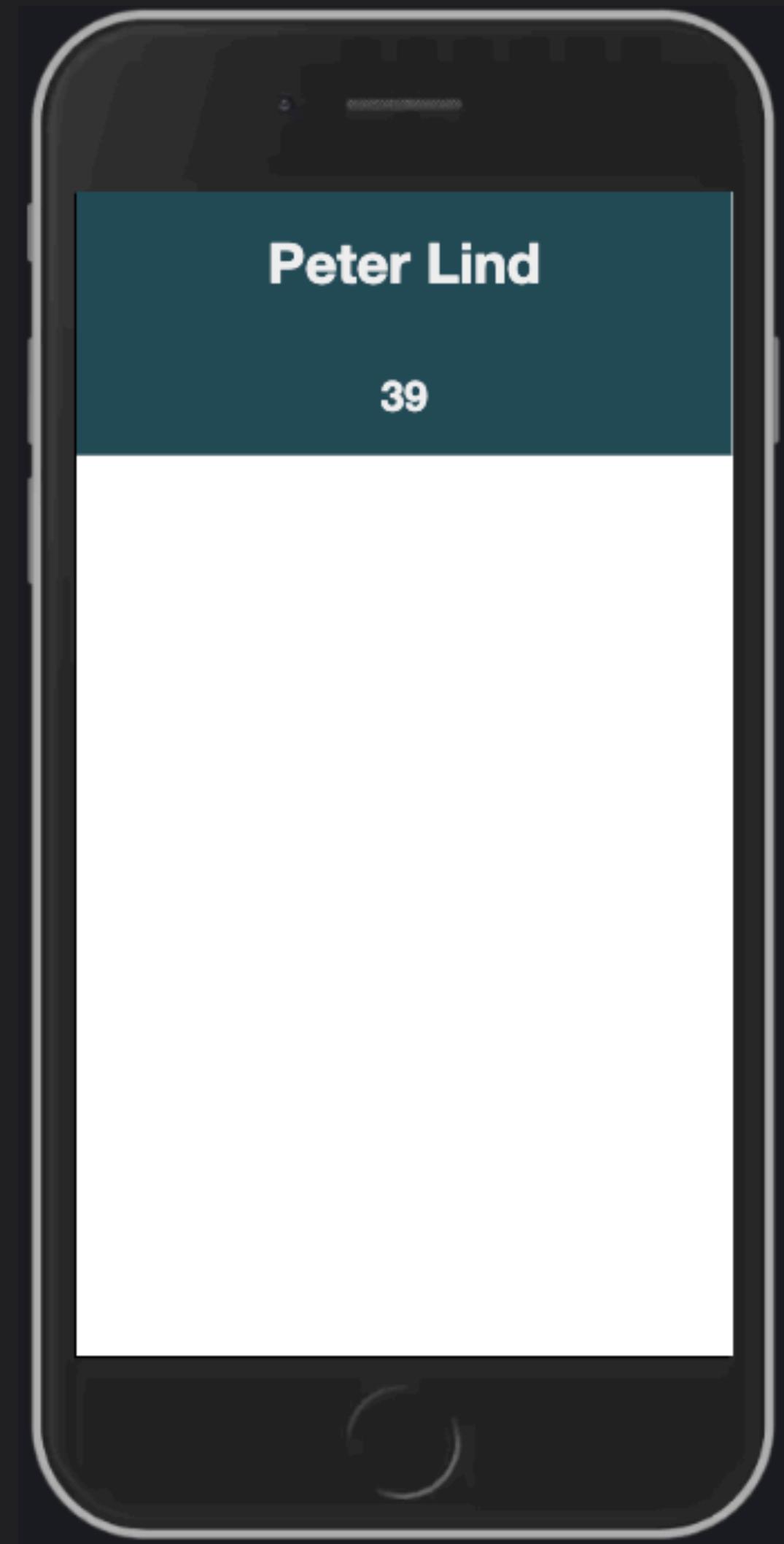
```
let fullName = "Peter Lind";
console.log(fullName);
document.querySelector("#fullName_container").textContent = fullName;
```

.textContent

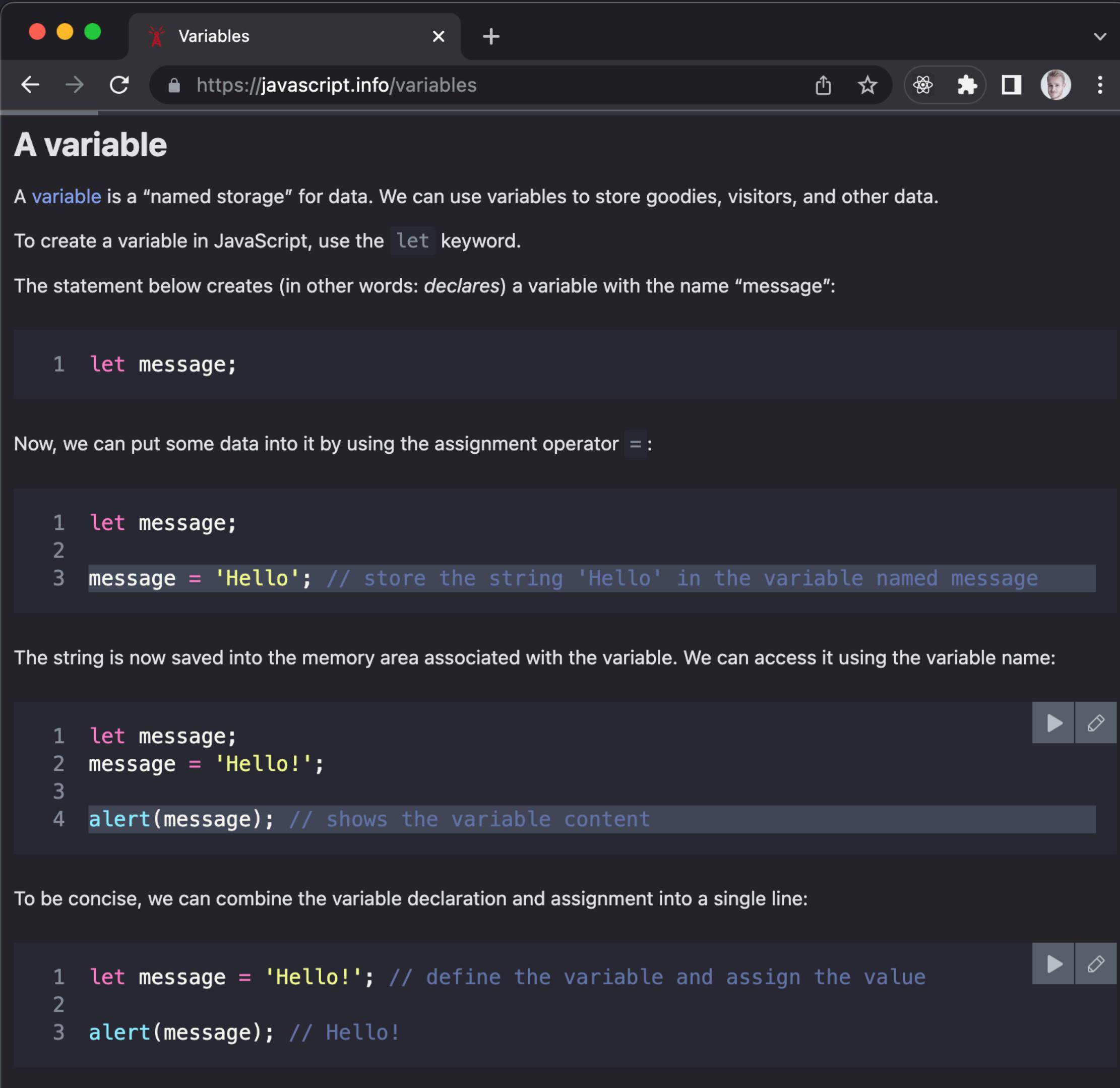
Get or set the text content of a given (HTML) element

Variables & UI

```
let fullName = "Peter Lind";
let age = 39;
console.log(fullName, age);
document.querySelector("#fullName_container").textContent = fullName;
document.querySelector("#age_container").textContent = age;
```



JavaScript.info/Variables



A screenshot of a web browser window titled "Variables". The URL in the address bar is <https://javascript.info/variables>. The page content is as follows:

A variable

A [variable](#) is a “named storage” for data. We can use variables to store goodies, visitors, and other data.

To create a variable in JavaScript, use the `let` keyword.

The statement below creates (in other words: *declares*) a variable with the name “message”:

```
1 let message;
```

Now, we can put some data into it by using the assignment operator `=`:

```
1 let message;
2
3 message = 'Hello'; // store the string 'Hello' in the variable named message
```

The string is now saved into the memory area associated with the variable. We can access it using the variable name:

```
1 let message;
2 message = 'Hello!';
3
4 alert(message); // shows the variable content
```

To be concise, we can combine the variable declaration and assignment into a single line:

```
1 let message = 'Hello!'; // define the variable and assign the value
2
3 alert(message); // Hello!
```

Read, read,
read,
The docs



var vs let

THE DIFFERENCE IS THE SCOPING

VAR IS FUNCTION-WIDE OR GLOBAL SCOPE

LET IS BLOCK SCOPED

VAR TOLERATES REDECLARATION

<https://javascript.info/variables>

<https://javascript.info/var>

```
// Example 1
// "var" has no block scope
if (true) {
| var test1 = true; // use "var" instead of "let"
}
console.log(test1); // true, the variable lives after if

// Example 2
if (true) {
| let test2 = true; // use "let"
}
console.log(test2); // Error: test is not defined

// Example 3
for (var i = 0; i < 10; i++) {
| // ...
}
console.log(i); // 10, "i" is visible after loop, it's a global variable
```

```
// "var" tolerates redeclarations
var user1 = "Pete";
var user1 = "John"; // this "var" does nothing (already declared)
// ...it doesn't trigger an error
console.log(user1); // John

let user2;
let user2; // SyntaxError: 'user' has already been declared
```

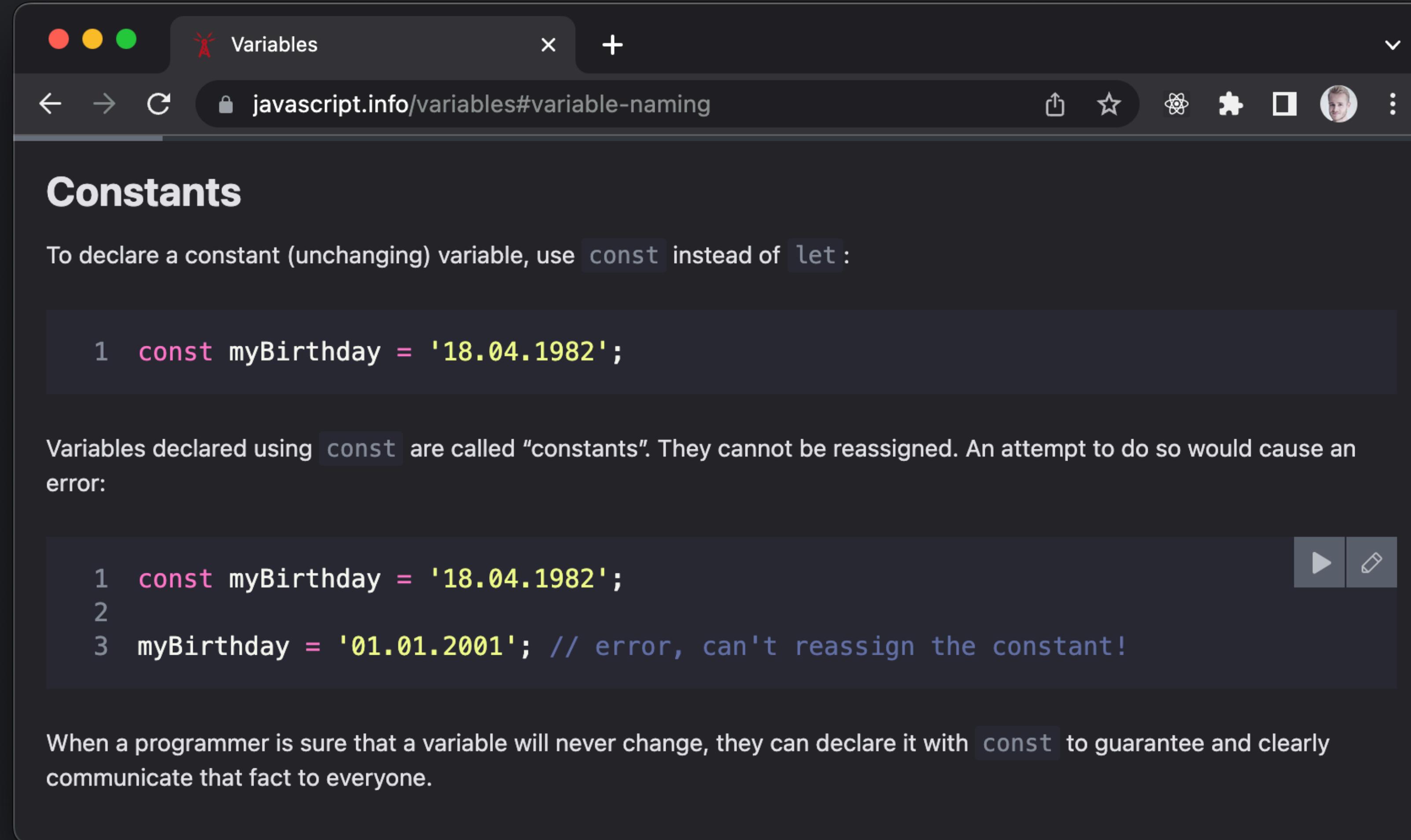
Const

Const is an unchanging variable.

```
const myBirthday = "12-03-1990";
myBirthday = "12-03-1989";
// Uncaught TypeError: can't reassign the constant!
```

const cannot be reassigned.
If you try to, an error will be thrown.

And the doc says...



The screenshot shows a dark-themed web browser window with the title bar "Variables". The address bar contains the URL "javascript.info/variables#variable-naming". The main content area displays the following text and code examples:

Constants

To declare a constant (unchanging) variable, use `const` instead of `let`:

```
1 const myBirthday = '18.04.1982';
```

Variables declared using `const` are called "constants". They cannot be reassigned. An attempt to do so would cause an error:

```
1 const myBirthday = '18.04.1982';
2
3 myBirthday = '01.01.2001'; // error, can't reassign the constant!
```

When a programmer is sure that a variable will never change, they can declare it with `const` to guarantee and clearly communicate that fact to everyone.

Const can't be reassigned

```
const myBirthday = "12-03-1990";
myBirthday = "12-03-1989"; // Uncaught TypeError: can't reassign the constant!

const person = {
    name: "Kasper",
    mail: "kato@eaaa.dk",
    age: 32
};

person.age = 33; // no error

person = {
    name: "Rasmus",
    mail: "race@eaaa.dk",
    age: 31
}; // Uncaught TypeError: can't reassign the constant!
```

Use let & const
instead of var

<https://javascript.info/variables>
<https://javascript.info/var>

But which one?

<https://javascript.info/variables>
<https://javascript.info/var>

Start with const

If needed, change to let

Never use var!

<https://javascript.info/variables>

<https://javascript.info/var>

The screenshot shows a web browser window with a dark theme. The title bar says "Variables". The address bar shows the URL "javascript.info/variables#name-things-right". The main content area has a sidebar on the left with a list of navigation items: Chapter, JavaScript Fundamentals, Lesson navigation, A variable, A real-life analogy, Variable naming, Constants, Name things right (which is the current page), Summary, Tasks (3), Comments, Share, and Edit on GitHub. The main content area has a header "Name things right" and a sub-header "Talking about variables, there's one more extremely important thing." It discusses the importance of variable naming for readability and maintainability. It also provides some good-to-follow rules for naming variables.

Name things right

Talking about variables, there's one more extremely important thing.

A variable name should have a clean, obvious meaning, describing the data that it stores.

Variable naming is one of the most important and complex skills in programming. A quick glance at variable names can reveal which code was written by a beginner versus an experienced developer.

In a real project, most of the time is spent modifying and extending an existing code base rather than writing something completely separate from scratch. When we return to some code after doing something else for a while, it's much easier to find information that is well-labeled. Or, in other words, when the variables have good names.

Please spend time thinking about the right name for a variable before declaring it. Doing so will repay you handsomely.

Some good-to-follow rules are:

- Use human-readable names like `userName` or `shoppingCart`.
- Stay away from abbreviations or short names like `a`, `b`, `c`, unless you really know what you're doing.
- Make names maximally descriptive and concise. Examples of bad names are `data` and `value`. Such names say nothing. It's only okay to use them if the context of the code makes it exceptionally obvious which data or value the variable is referencing.
- Agree on terms within your team and in your own mind. If a site visitor is called a "user" then we should name related variables `currentUser` or `newUser` instead of `currentVisitor` or `newManInTown`.

Sounds simple? Indeed it is, but creating descriptive and concise variable names in practice is not. Go for it.

<https://javascript.info/variables#name-things-right>

The screenshot shows a dark-themed web browser window with the title bar 'Variables'. The address bar displays the URL 'javascript.info/variables'. The main content area is titled 'Variable naming'.

There are two limitations on variable names in JavaScript:

1. The name must contain only letters, digits, or the symbols `$` and `_`.
2. The first character must not be a digit.

Examples of valid names:

```
1 let userName;
2 let test123;
```

When the name contains multiple words, **camelCase** is commonly used. That is: words go one after another, each word except first starting with a capital letter: `myVeryLongName`.

What's interesting – the dollar sign `'$'` and the underscore `'_'` can also be used in names. They are regular symbols, just like letters, without any special meaning.

These names are valid:

```
1 let $ = 1; // declared a variable with the name "$"
2 let _ = 2; // and now a variable with the name "_"
3
4 alert($ + _); // 3
```

Examples of incorrect variable names:

```
1 let 1a; // cannot start with a digit
2
3 let my-name; // hyphens '-' aren't allowed in the name
```

<https://javascript.info/variables#variable-naming>

Name things right (or wrong?)

1. Create a variable with the name of our planet. How would you name such a variable?
2. Create a variable to store the name of a current visitor to a website. How would you name that variable?

<https://javascript.info/variables>

Global Variables

Variables outside a function (and scopes) are global variables

JavaScript.info/function-basics#local-variables

Local variables

A variable declared inside a function is only visible inside that function.

For example:

```
1 function showMessage() {  
2   let message = "Hello, I'm JavaScript!"; // local variable  
3  
4   alert( message );  
5 }  
6  
7 showMessage(); // Hello, I'm JavaScript!  
8  
9 alert( message ); // <-- Error! The variable is local to the function
```

Scopes:

- Local Variable
- Global Variable

Outer variables

A function can access an outer variable as well, for example:

```
1 let userName = 'John';  
2  
3 function showMessage() {  
4   let message = 'Hello, ' + userName;  
5   alert(message);  
6 }  
7  
8 showMessage(); // Hello, John
```

Global variables

Variables declared outside of any function, such as the outer `userName` in the code above, are called *global*.

Global variables are visible from any function (unless shadowed by locals).

It's a good practice to minimize the use of global variables. Modern code has few or no globals. Most variables reside in their functions. Sometimes though, they can be useful to store project-level data.

Datatyper

Application Programming Interface

Data Types

In JavaScript there are two main data types:

- **Primitive values** like strings, numbers and booleans.
- **Objects** with properties.

```
1 let str = "Hello";
2 let str2 = 'Single quotes are ok too';
3 let phrase = `can embed another ${str}`;
```

```
1 let n = 123;
2 n = 12.345;
```

```
1 let user = new Object(); // "object constructor" syntax
2 let user = {}; // "object literal" syntax
```

In JavaScript, a value always has a certain type like a string, number, boolean, object, array, etc.

Data Types

Variables can hold 7 different kinds of values (data types)

- only one at a time ...

Boolean

Number

String

Object

Null

Undefined

Symbol

Datatypes - when to use which one?

Boolean	Values that are true or false – conditions
Number	Numbers! Points, counting, math-operations , measuring ...
String	Text! Input-fields – HTML and CSS-attributes ... URLs (Phone-number)
Object	Everything (Arrays, Functions) Combined/collected data (of the other types)
Null	No datatype yet – no value!
Undefined	Something that hasn't defined before – <u>don't set things to undefined!</u>
Symbol	Something complicated

Datatyper

1. Prøv disse og test med
console.log
2. Hvordan ser vi værdien?
3. Hvordan ser vi typen?
4. Hvordan definerer du,
hvilken type, du vil
gemme i en variabel?

```
const bool = true;  
const num = 41;  
const str = "Peter";  
const obj = {  
    cats: 2,  
    cars: 1  
};  
const nothing = null;  
let undf;  
const symbol = Symbol("symbol");
```

Strings

In JavaScript textual data is stored as strings.

Strings are defined with either ‘single quotes’, “double quotes” or `backticks`.

```
let single = 'single-quoted';
```

```
let double = "double-quoted";
```

```
let backticks = `backticks`;
```

```
let fullName = "Peter Lind";
let age = 39;

let message = fullName + " is " + age + " years old.";
console.log(message); // Peter Lind is 39 years old.
```

Concatenation

We can combine two or more variables in one string (variable).

```
let fullName = "Peter Lind";
let age = 39;

// single
console.log(fullName + ' is ' + age + ' years old.');
// double
console.log(fullName + " is " + age + " years old.");
// backticks
console.log(` ${fullName} is ${age} years old.`);
```

Concatenation

Strings are defined with either ‘single quotes’, “double quotes” or `backticks`.

String quotes

1. Test the code to the right.
2. What is the output of the script to the right?

```
"use strict";  
  
let fullName = "Alicia Keys";  
  
console.log(`hello ${1}`); // ?  
  
console.log(`hello ${"name"} `); // ?  
  
console.log(`hello ${fullName}`); // ?
```

Data Type Conversion

In JavaScript, we never specify the type of a variable (String, Number, Boolean, Object, etc.).

We set the value and use the value as the type we expect it to be.

```
const firstName = "Rasmus";
const lastName = "Cederdorff";

let age = 32;

let isSeniorLecture = true;

const obj = {
  cats: 0,
  cars: 1
};

const kids = ["Alicia", "Ida"];
```

It happens when we do:

- Concatenation (+)
- Comparisons (==, !=, <, >, <=, >=)
- Calculations (-, *, /, %)

Automatic conversion

So often JavaScript has to do automatic conversion.

Everything in HTML is strings, so when writing or reading numbers, they have to be converted.

Most of the time this happens automatically.

“Semi-automatic” conversion

`String(value)` converts a value to a string, usually by calling `value.toString()`

`Number(value)` converts a value to a number, usually by calling `value.valueOf()`

`Boolean(value)` converts a value to a boolean, by unknown magic means!

Note:

`Boolean(0)` is false, but
`Boolean("0")` is true

It is always possible to convert something into a string –
but it might not be possible to convert everything into a number!

Numeric conversion

1. How do you convert to Number?

```
let string = "1234";  
console.log(string);  
// number??
```

String conversion

1. How do you convert to String?

```
let number = 1234;  
console.log(number);  
// string??
```

Numeric conversion 2

1. Try yourself!

2. What is the output in
the console?

```
console.log(Number(" 123  ")); // ?  
console.log(Number("123z")); // ?  
console.log(Number(true)); // ?  
console.log(Number(false)); // ?
```

Type Conversions

← → C 🔒 javascript.info/type-conversions

EN  Buy EPUB/PDF  ☼ ⚡

Chapter

JavaScript Fundamentals

Lesson navigation

String Conversion

Numeric Conversion

Boolean Conversion

Summary

Comments

Share  

Edit on GitHub

Type Conversions

January 24, 2023

Most of the time, operators and functions automatically convert the values given to them to the right type.

For example, `alert` automatically converts any value to a string to show it. Mathematical operations convert values to numbers.

There are also cases when we need to explicitly convert a value to the expected type.

i Not talking about objects yet

In this chapter, we won't cover objects. For now, we'll just be talking about primitives.

Later, after we learn about objects, in the chapter [Object to primitive conversion](#) we'll see how objects fit in.

String Conversion

String conversion happens when we need the string form of a value.

For example, `alert(value)` does it to show the value.

We can also call the `String(value)` function to convert a value to a string:

```
1 let value = true;
2 alert(typeof value); // boolean
3
4 value = String(value); // now value is a string "true"
5 alert(typeof value); // string
```

String conversion is mostly obvious. A `false` becomes "false", `null` becomes "null", etc.

Numeric Conversion

Numeric conversion in mathematical functions and expressions happens automatically.

For example, when division `/` is applied to non-numbers,

Read, read, read, The docs



Manual conversion

You can omit
the prefix
`Number.` if you
want

`Number.parseInt(string)`

Converts a string **to a number**, using the specified numerical system ([MDN](#))

`Number.prototype.toString()`

Converts a number **to a string**, in the specified numerical system ([MDN](#))

There's also: `parseFloat`, `toExponential`, `toFixed`, `toPrecision` for other kinds of conversion

```
const n1 = 5;  
const n2 = 6;
```

```
const s1 = "4";  
const s2 = "9";
```

```
console.log(n1 + n2);  
console.log(s1 + s2);
```

```
console.log(n1 - n2);  
console.log(s1 - s2);
```

```
console.log(n1 + s2);  
console.log(s1 + n2);
```

```
console.log(n1 - s2);  
console.log(s1 - n2);
```

Concatenation looks like addition

1. What will be the result of these calculations/concatenations?
2. Test them in the console!

Automatic data type conversion with concatenation

String + string is a **concatenation**

Number + number is a **calculation**

But if the types are different, one gets converted to the other!

The challenge is to figure out which one gets converted to what ...

```
"" + 1 + 0;  
"" - 1 + 0;  
true + false;  
6 / "3";  
"2" * "3";  
4 + 5 + "px";  
"$" + 4 + 5;  
"4" - 2;  
"4px" - 2;  
" -9 " + 5;  
" -9 " - 5;  
null + 1;  
undefined + 1;  
" \t \n" - 2;
```

Type Conversion

1. Test these expressions.
2. First: Try to guess the result, then check your guess in the console!
3. Explain to each other why your guess was correct/wrong.

Template String

A template string (template literal) in JavaScript is a string enclosed in backticks (`) that allows embedding variables and expressions using \${ ... } and supports multi-line text.

`template string`

“Template literals are literals delimited with backtick (`) characters, allowing for multi-line strings, for string interpolation with embedded expressions, and for special constructs called tagged templates.”

```
let name = "Alicia";
let age = 6;
```

```
console.log(name + " is " + age + " years old.");
```

```
console.log(` ${name} is ${age} years old. `);
```

Alicia is 6 years old.

[main.js:10](#)

Alicia is 6 years old.

[main.js:12](#)

`template string`

A *template string (template literal)* in JavaScript is a string enclosed in backticks () that allows embedding variables and expressions using \${...} and supports multi-line text.

```
const navn = "Rasmus";
const alder = 34;

const tekst = `Hej, jeg hedder ${navn} og jeg er ${alder} år gammel.`;

console.log(tekst);
// Output: Hej, jeg hedder Rasmus og jeg er 34 år gammel.
```

```
const navn = "Rasmus";
const alder = 34;

const tekst = `Hej, jeg hedder ${navn} og jeg er ${alder} år gammel.`;
const tekstUdenTemplate = "Hej, jeg hedder " + navn + " og jeg er " + alder + " år gammel.";

console.log(tekst);
// Output: Hej, jeg hedder Rasmus og jeg er 34 år gammel.
```

`template string`

Backtick String / Template Literals

- Extended functionality
- Simplifies concatenating strings
- Embed values and expression into a string with \${ ... }
- Simplifies the syntax and the reading
- Let us create more readable HTML templates

```
let name = "Alicia";
console.log(`Hello, ${name}`);
```

Hello, Alicia

main.js:8

`template string`

```
let name = "Alicia";
let age = 6;

console.log(name + " is " + age + " years old.");

console.log(`${name} is ${age} years old.`);
```

Alicia is 6 years old.

[main.js:10](#)

Alicia is 6 years old.

[main.js:12](#)

`template string`

REGULAR STRING EXPRESSION

```
function appendTeachers(teachers) {  
  for (let teacher of teachers) {  
    console.log(teacher);  
    document.querySelector("#grid-teachers").innerHTML +=  
      "<article>" +  
      "<img src='" + teacher.img + "'>" +  
      "<h3>" + teacher.name + "</h3>" +  
      teacher.position + "<br>" +  
      "<a href='mailto:" + teacher.mail + "'>" + teacher.mail + "</a>" +  
      "</article>";  
  }  
}
```

TEACHERS



Birgitte Kirk Iversen

Senior Lecturer
bki@baaa.dk



Michael Hvidtfeldt

Senior Lecturer
mhv@baaa.dk



Rasmus Cederdorff

Lecturer
race@baaa.dk

`template string`

... EMBED VARIABLES AND EXPRESSIONS IN A STRING

```
function appendTeachers(teachers) {  
  for (let teacher of teachers) {  
    console.log(teacher);  
    document.querySelector("#grid-teachers").innerHTML +=  
      "<article>" +  
      "<img src=''" + teacher.img + "'>" +  
      "<h3>" + teacher.name + "</h3>" +  
      teacher.position + "<br>" +  
      "<a href='mailto:" + teacher.mail + "'>" + teacher.mail + "</a>" +  
      "</article>";  
  }  
}
```



```
function appendTeachers(teachers) {  
  for (let teacher of teachers) {  
    console.log(teacher);  
    document.querySelector("#grid-teachers").innerHTML += `  
      <article>  
        <img src='${teacher.img}'>  
        <h3>${teacher.name}</h3>  
        ${teacher.position}<br>  
        <a href='mailto:${teacher.mail}'>${teacher.mail}</a>  
      </article>`;  
  }  
}
```

`VS Code ES6 String HTML`

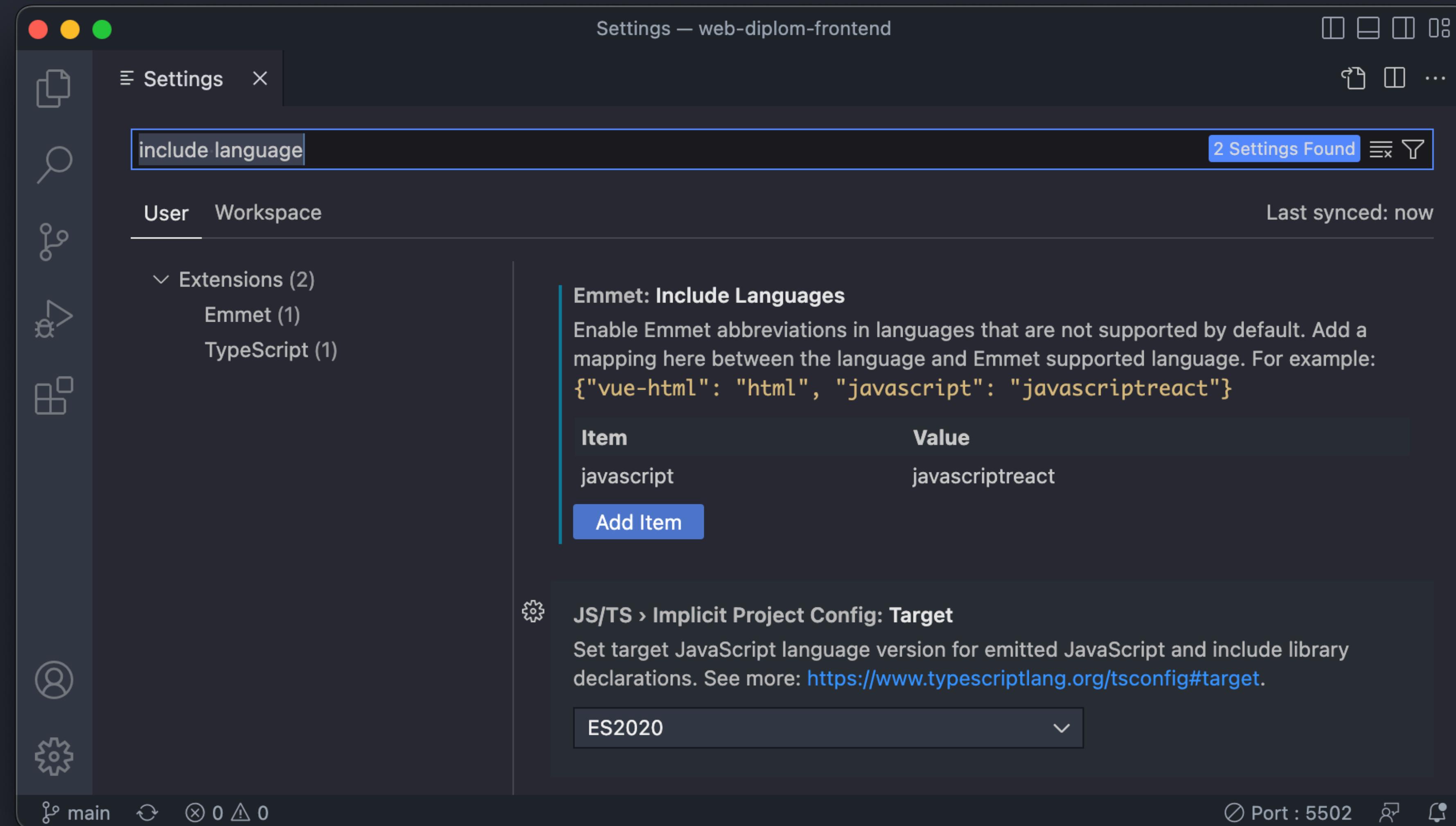
<https://marketplace.visualstudio.com/items?itemName=hjb2012.vscode-es6-string-html>

```
function appendTeachers(teachers) {
  for (let teacher of teachers) {
    console.log(teacher);
    document.querySelector("#grid-teachers").innerHTML += `
      <article>
        <img src='${teacher.img}'>
        <h3>${teacher.name}</h3>
        ${teacher.position}<br>
        <a href='mailto:${teacher.mail}'>${teacher.mail}</a>
      </article>`;
  }
}
```



```
function appendTeachers(teachers) {
  for (let teacher of teachers) {
    console.log(teacher);
    document.querySelector("#grid-teachers").innerHTML +=/*html*/
      <article>
        <img src='${teacher.img}'>
        <h3>${teacher.name}</h3>
        ${teacher.position}<br>
        <a href='mailto:${teacher.mail}'>${teacher.mail}</a>
      </article>;
  }
}
```

Add language support in template string



Objects

A set of named values: `key:value`

Objects

A set of named values

Objects are used to store keyed
collections of various data



Containers for named values
called properties. A property
is a “key: value” pair

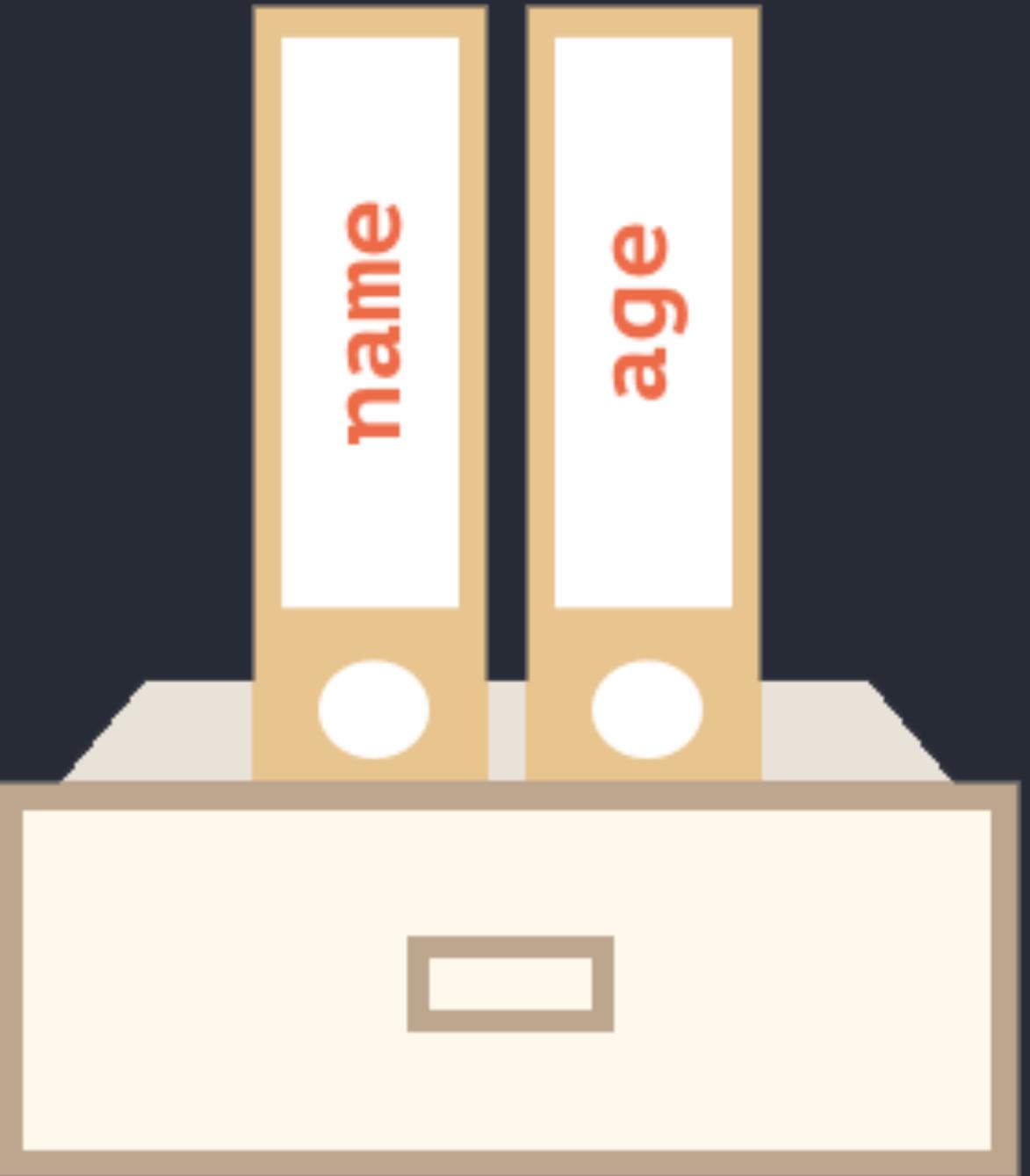
Objects

A set of named values

```
let user = {  
    name: 'Alicia',  
    age: 6  
};
```

```
console.log(user.name +  
    " is " + user.age +  
    " years old.");
```

user

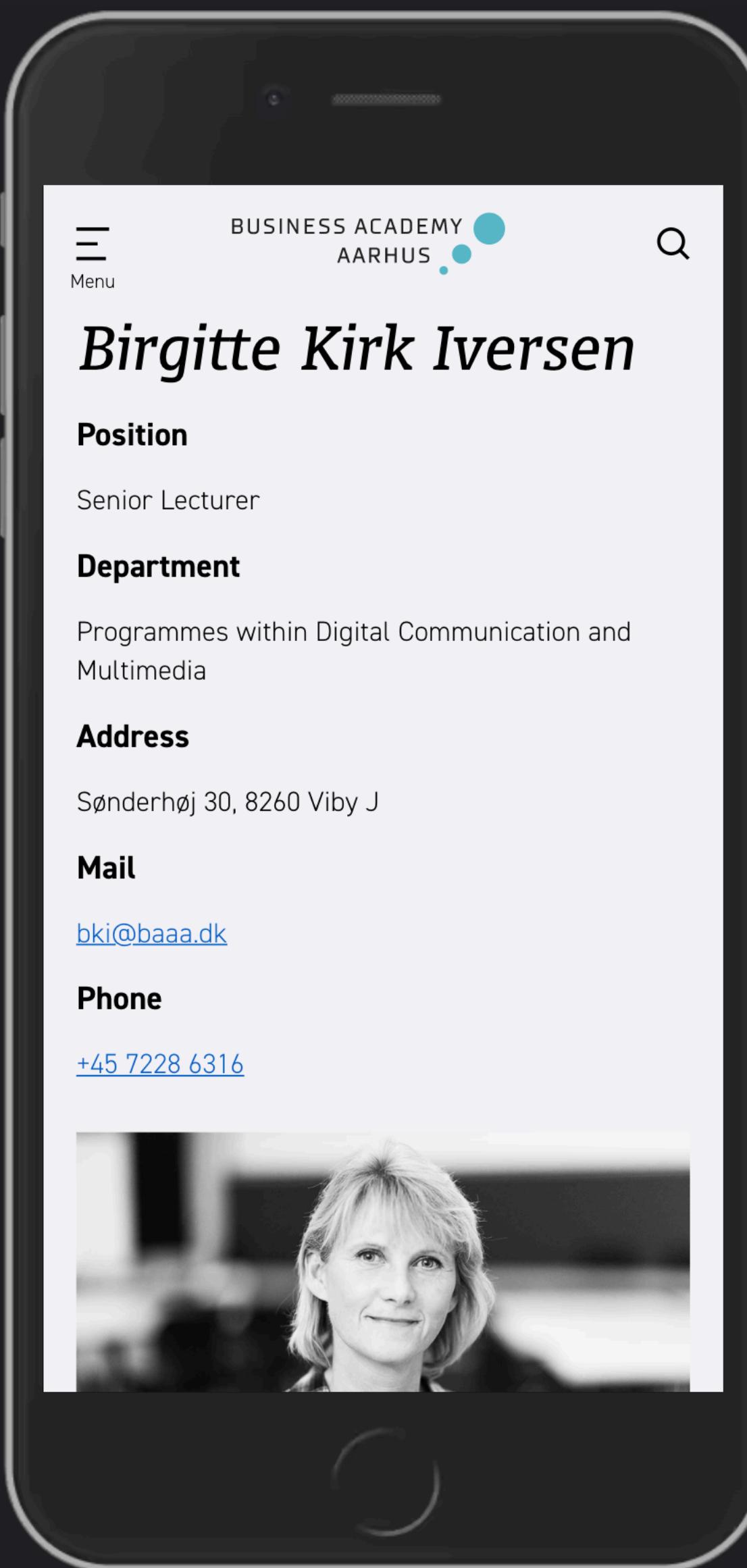


Alicia is 6 years old.

main.js:11

Objects

A set of named values



Objects

The screenshot shows a web browser window with a dark theme. The title bar reads "Birgitte Kirk Iversen" and the address bar shows the URL "https://www.baaa.dk/contact/find-employee/employee/birgitte-kirk-iversen". The page content is as follows:

BUSINESS ACADEMY AARHUS

[Home](#) [Contact](#) [Find employee](#) **Employee**

Birgitte Kirk Iversen

Position
Senior Lecturer

Department
Programmes within Digital
Communication and Multimedia

Mail
bki@baaa.dk

Phone
[+45 7228 6316](tel:+4572286316)

Address
Sønderhøj 7G, 8260 Viby J

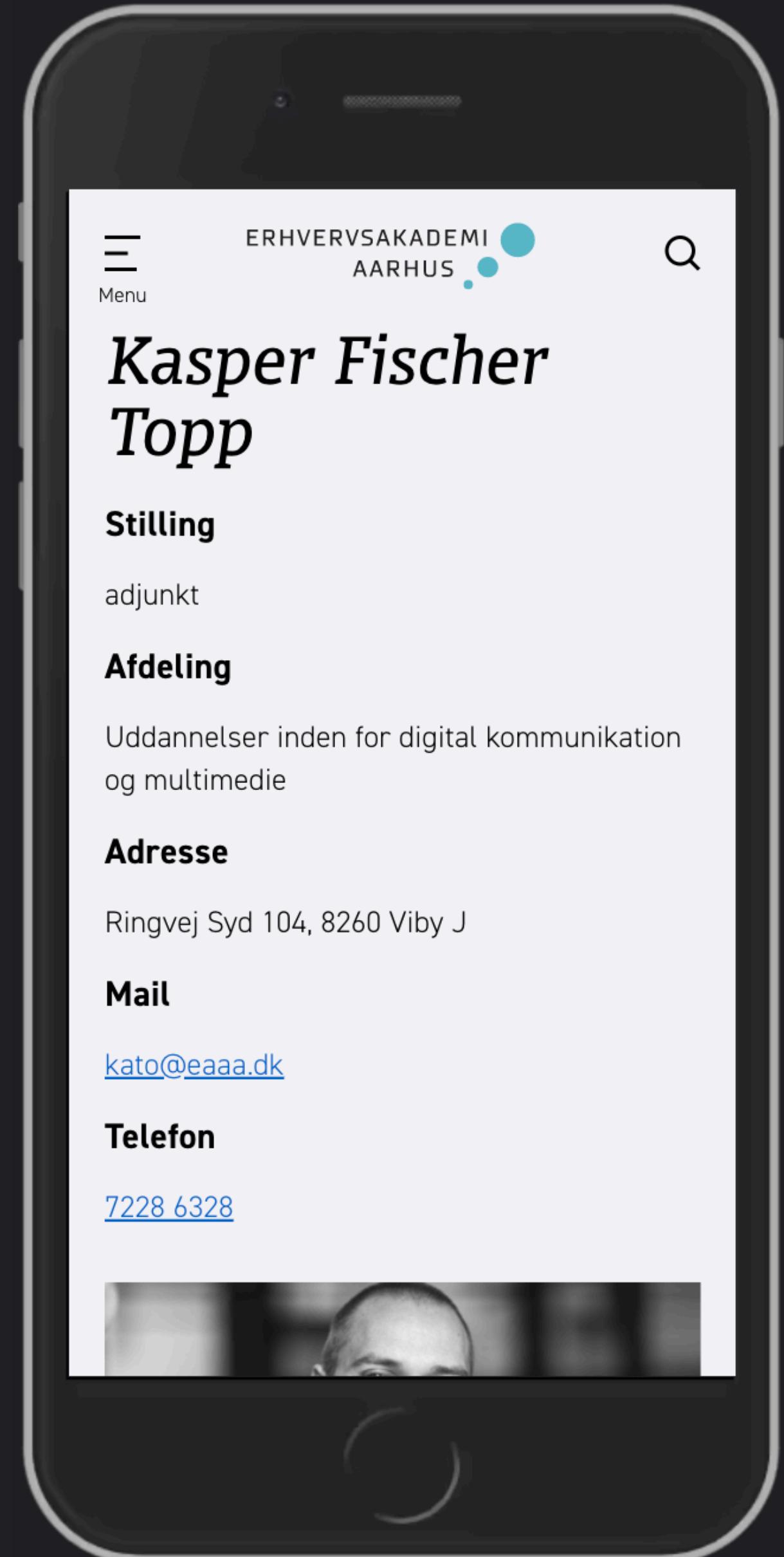
A large circular portrait of Birgitte Kirk Iversen is displayed on the right side of the page. She is a woman with short, light-colored hair and glasses, wearing a dark jacket over a collared shirt.

<https://www.baaa.dk/contact/find-employee/employee/birgitte-kirk-iversen>

Objects

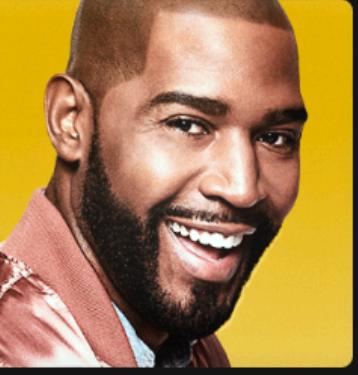
A set of named values

```
key           value
const mrBackend = {
  name: "Kasper Fischer Topp",
  mail: "kato@eaaa.dk",
  phone: "72286328",
  position: "Lecturer",
  favTechnologies: ["PHP", "SQL"]
};
```



NETFLIX

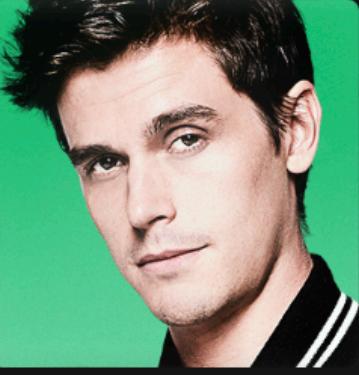
Hvem ser?



Personen der
rent faktisk
betaler for
profilen



Nasser 1



Nasser 2



Nasser 3

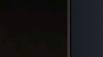


Nasser 4 Khader

Administrer profiler

www.netflix.com/browse

NETFLIX Start Serier Film Spil Nyt og populært Min liste Gennemse efter sprog

Q  

 NGAME
Too Hot To Handle 3

Mobilspil • Interaktiv historie
Inkluderet i dit medlemskab

Du er en sexbombe. Crash retræten, og skab drama blandt parrene i denne sæson af det populære dating-spil. Der er saftige overraskelser i vente.

 Mere info

 Sean

17+

Actionfilm baseret på bøger

Flere titler til dig

Frost II (2019) - IMDb

imdb.com/title/tt4520988/

IMDb Menu All Search IMDb

Frost II

Original title: Frozen II
2019 · 7 · 1h 43m

IMDb RATING YOUR RATING POPULARITY

★ 6.8/10 160K ★ Rate 896 ▲ 102

Cast & crew · User reviews · Trivia · IMDbPro 🔍 All topics | [Share](#)

+ Play trailer 0:16

55 VIDEOS

99+ PHOTOS

Animation Adventure Comedy

+ Add to Watchlist

Anna, Elsa, Kristoff, Olaf and Sven leave Arendelle to travel to an ancient, autumn-bound forest of an enchanted land. They set out to find the origin of Elsa's powers in order to save their kingdom.

1.4K User reviews 289 Critic reviews 64 Metascore

Directors Chris Buck · Jennifer Lee

Writers

```
let movie = {  
  title: "Frozen 2",  
  description: "Elsa the Snow Queen has a",  
  trailer: "https://www.youtube.com/embed",  
  length: "1h 43m",  
  year: "2019"  
}
```

Define yourself as an object

with the following properties

name, age, mail, phone, city, address

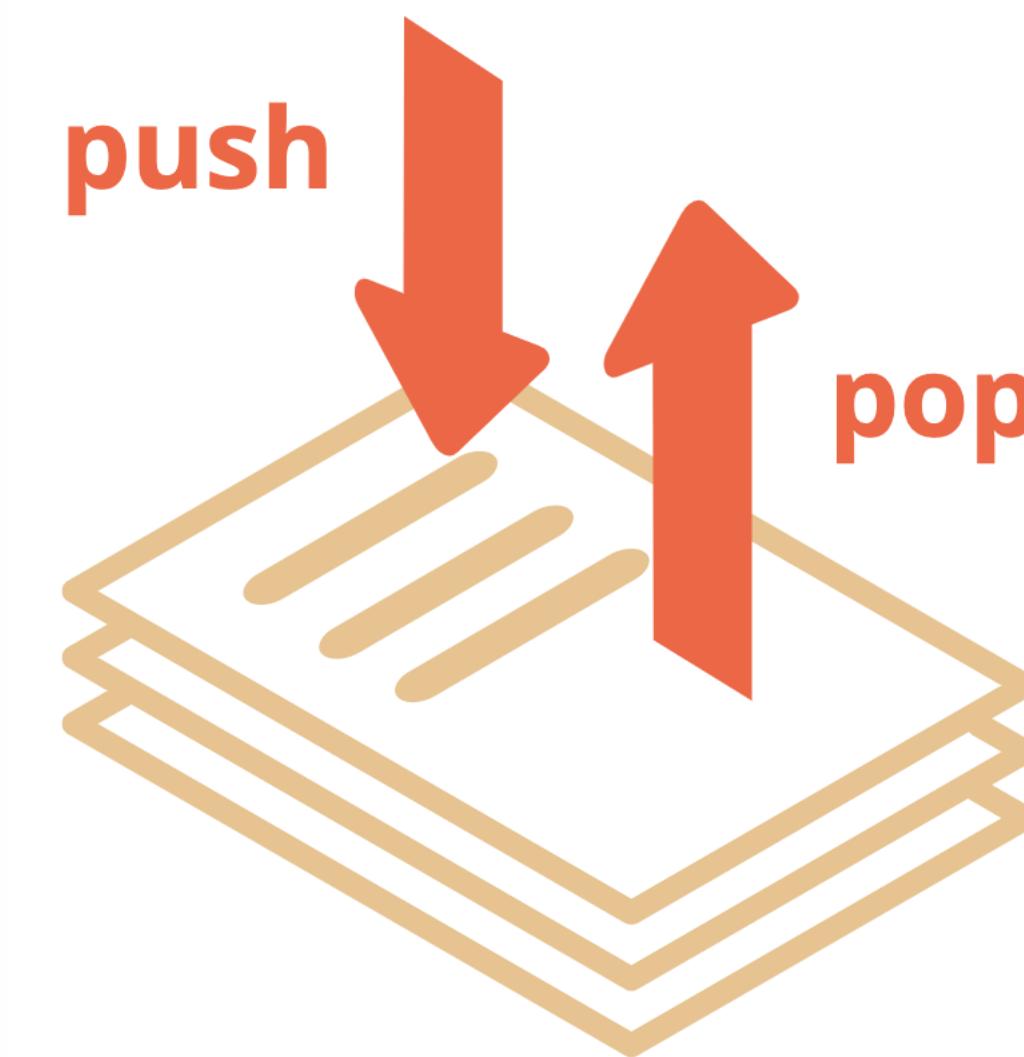
Arrays

Lists - collection of data

Arrays

Collections

Ordered collection of values or
objects



An array is a way to hold more than one value at a time we have a 1st, a 2nd, a 3rd, a 4th element and so on.

Arrays

Collections

Ordered collection of values or objects

```
// Only strings (text)
const movieTitles = ["The Matrix", "Inception"];

// Only numbers
const movieYears = [1999, 2010, 2014];

// Only booleans
const erGode = [true, true, false];

// Blandet indhold (fungerer også!)
const blandetListe = ["The Matrix", 1999, true];

console.log("Film navne:", filmNavne);
console.log("Film år:", filmÅr);
console.log("Er gode:", erGode);
console.log("Blandet:", blandetListe);
```

```
const allMovies = [
  {
    id: 1,
    title: "The Matrix",
    year: 1999, First element
    rating: 8.7,
    genre: ["Action", "Sci-Fi"]
  },
  {
    id: 2,
    title: "Inception",
    year: 2010, Second element
    rating: 8.8,
    genre: ["Action", "Thriller"]
  }
];

console.log("Complete movie database:", allMovies);
console.log("Number of movies:", allMovies.length);
```

```
let todaysLecturers = [
  {
    name: "Kasper Fischer Topp",
    mail: "kato@eaaa.dk",
    phone: "72286328",
    position: "Lecturer",
    favTechnologies: ["PHP", "SQL"],
    nickname: "Mr. Backend"
  },
  {
    name: "Rasmus Cederdorff",
    mail: "race@eaaa.dk",
    phone: "72286318",
    position: "Lecturer",
    favTechnologies: ["JavaScript"],
    nickname: "Mr. Frontend"
  }
];
```

First element

Second element

Arrays

Rasmus Cederdorff	
Position: Lecturer	<i>Michael Hvidtfeldt</i>
Department/ Multimedia De Digital Conce	
Address: Ringvej Syd 10	Position: Lecturer <i>Birgitte Kirk Iversen</i>
Mail: race@baaa.dk	Department/ Multimedia Di
Phone: 7228 6318	Address: Senior Lecturer Ringvej Syd 10
	Department/programme: Multimedia Design
Mail: mhv@baaa.dk	Address: Sønderhøj 30, 8260 Viby J
Phone: 7228 6328	Mail: bki@baaa.dk
	Phone: 7228 6316



```
main.js:21
▼ (3) [...], [...], [...] ⓘ
▶ 0: {name: "Birgitte Kirk Iversen", mail: "bki@baaa..."}
▶ 1: {name: "Michael Hvidtfeldt", mail: "mhv@baaa.dk..."}
▶ 2: {name: "Rasmus Cederdorff", mail: "race@baaa.dk..."}
  length: 3
▶ __proto__: Array(0)
main.js:22
▶ {name: "Michael Hvidtfeldt", mail: "mhv@baaa.dk"} ⓘ
main.js:23
```

```
let teachers = [
  name: "Birgitte Kirk Iversen",
  mail: "bki@baaa.dk"
},
{
  name: "Michael Hvidtfeldt",
  mail: "mhv@baaa.dk"
},
{
  name: "Rasmus Cederdorff",
  mail: "race@baaa.dk"
}
];
```

```
console.log(teachers);
console.log(teachers[1]);
console.log(teachers.length);
```

Teachers

http://127.0.0.1:5501/array-teachers/index.html

Console

main.js:46

```
▶ (4) [{} , {} , {} , {} ] ⓘ
  ▶ 0:
    address: "Sønderhøj 30, 8260 Viby J"
    department: "Multimedia Design"
    img: "https://www.eaaa.dk/media/u4gorzs"
    initials: "bki"
    mail: "bki@baaa.dk"
    name: "Birgitte Kirk Iversen"
    phone: "72286316"
    position: "Senior Lecturer"
    ► [[Prototype]]: Object
  ▶ 1: {name: 'Maria Louise Bendixen', initials: 'mlbe'}
  ▶ 2: {name: 'Kim Elkjær Marcher-Jepsen', initials: 'kje'}
  ▶ 3: {name: 'Rasmus Cederdorff', initials: 'race'}
  ▶ length: 4
  ▶ [[Prototype]]: Array(0)
```



Birgitte Kirk Iversen

Senior Lecturer
bki@baaa.dk



Maria Louise Bendixen

Senior Lecturer
mlbe@baaa.dk



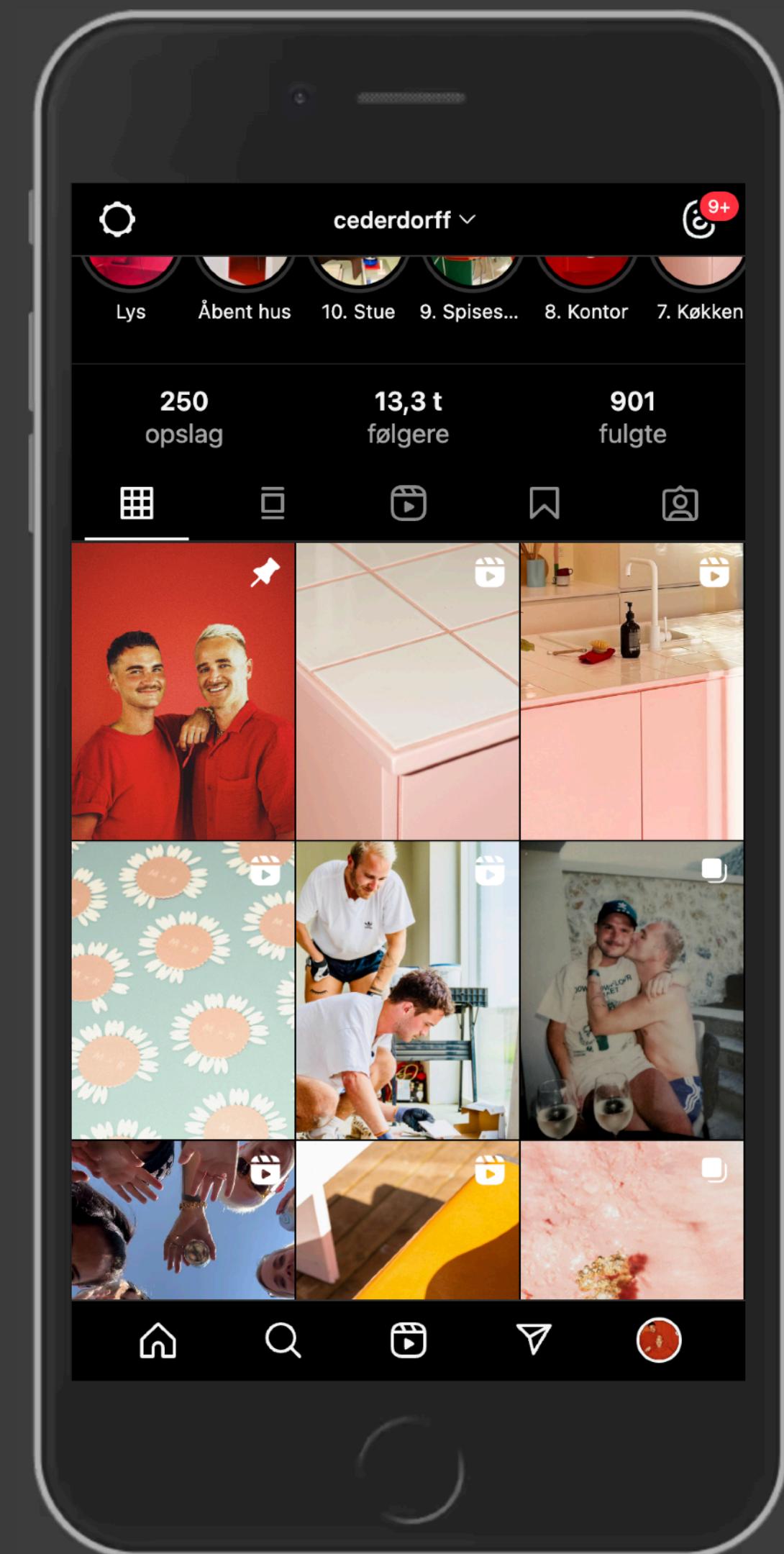
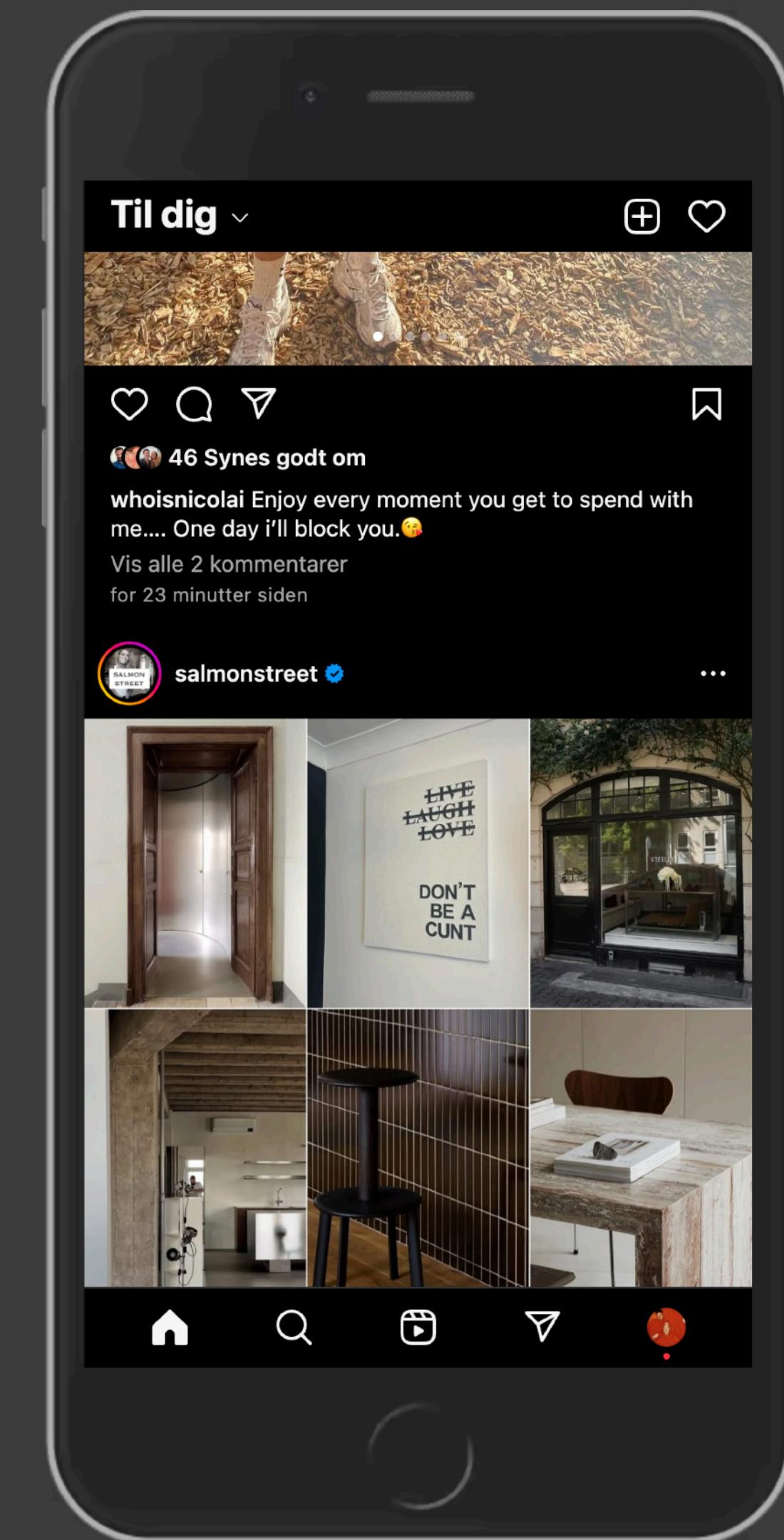
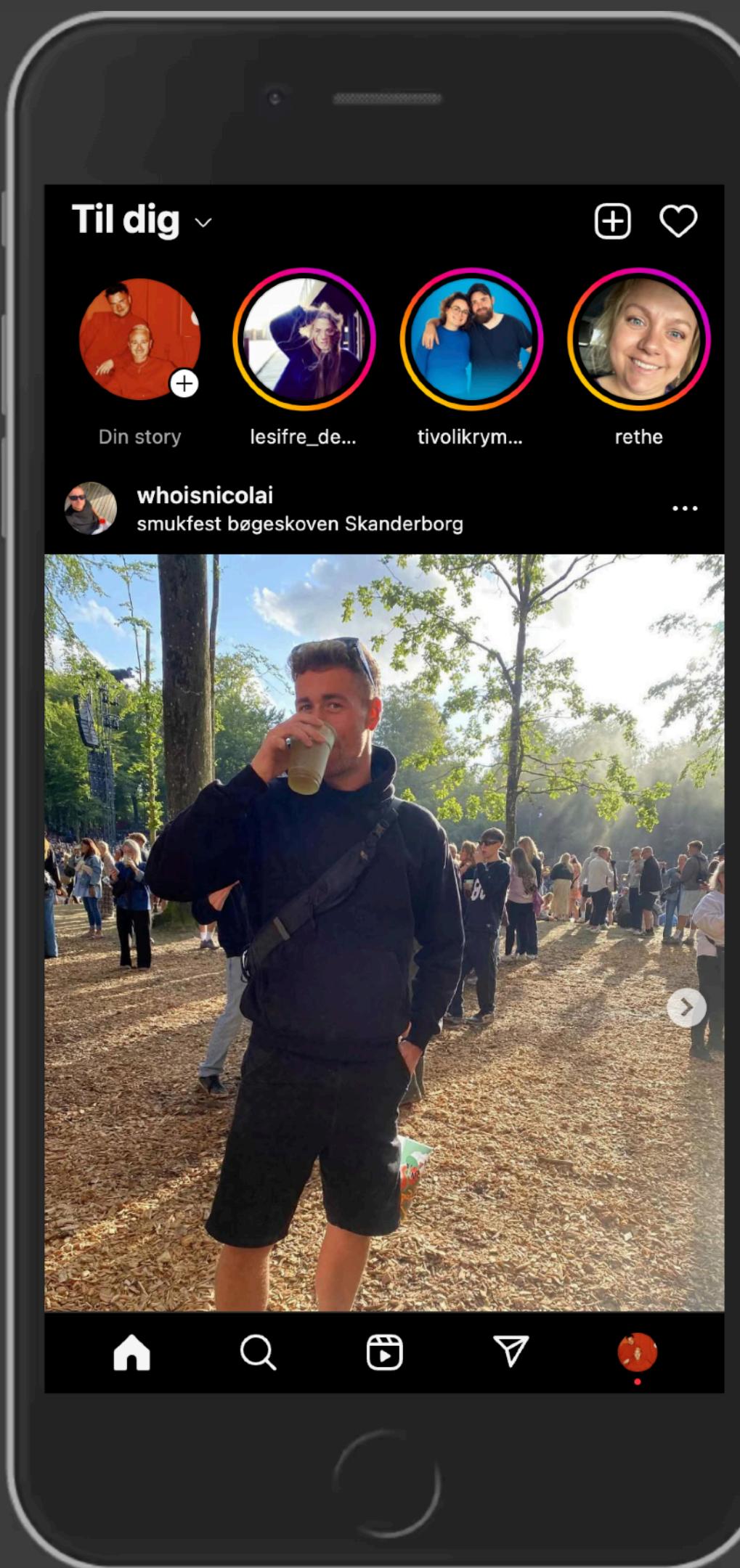
Kim Elkjær Marcher-Jepsen

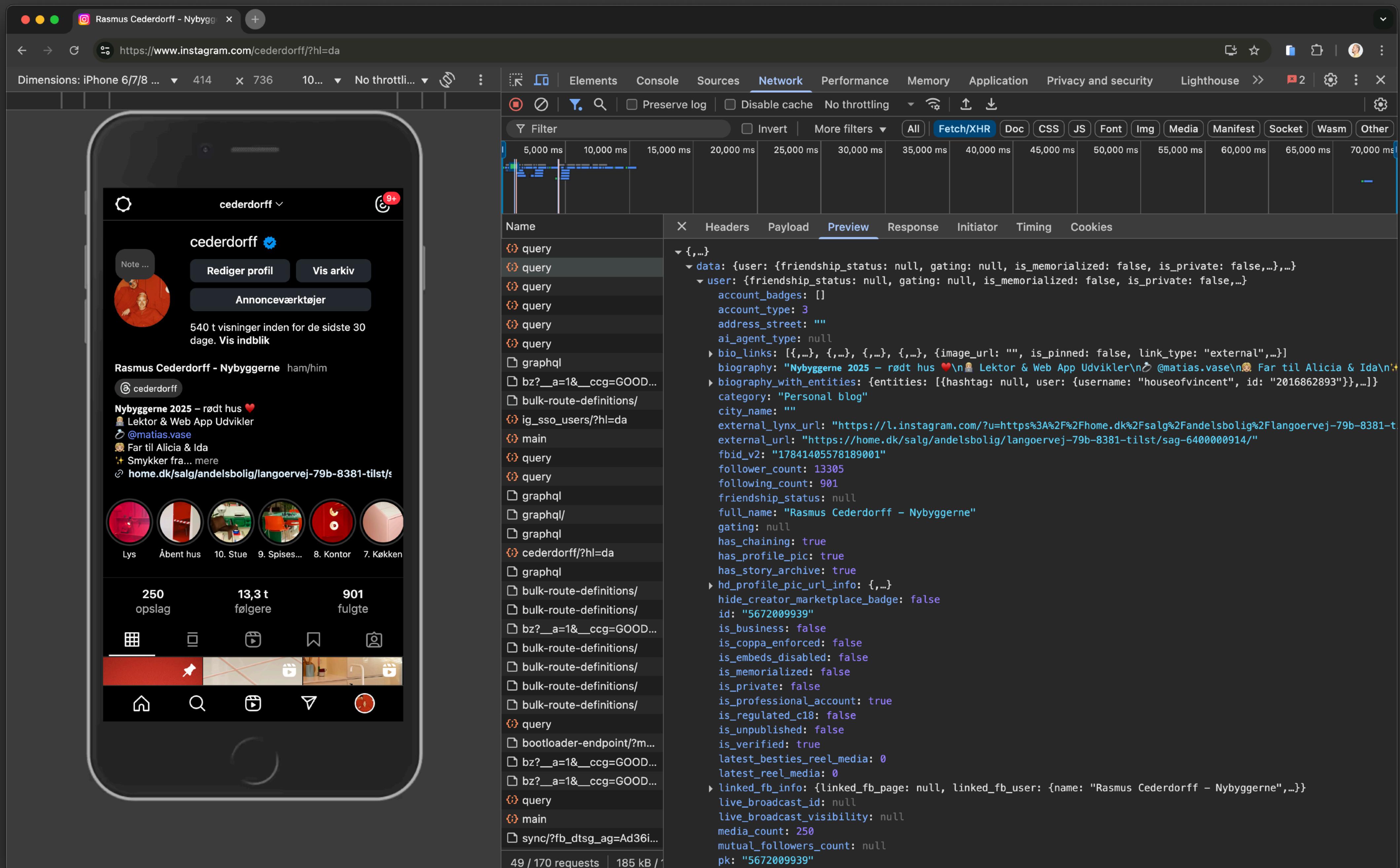
Lecturer
kje@baaa.dk

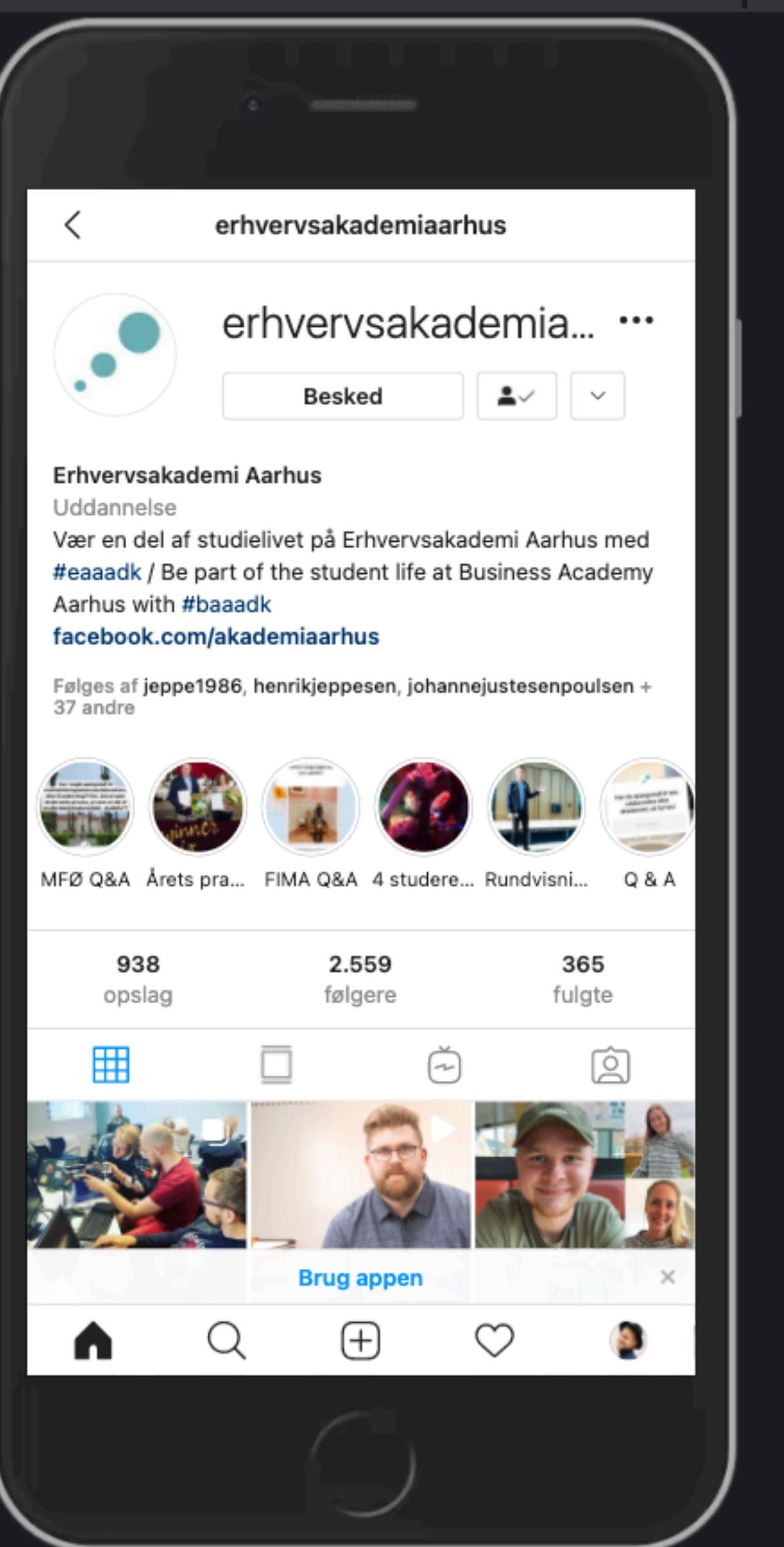


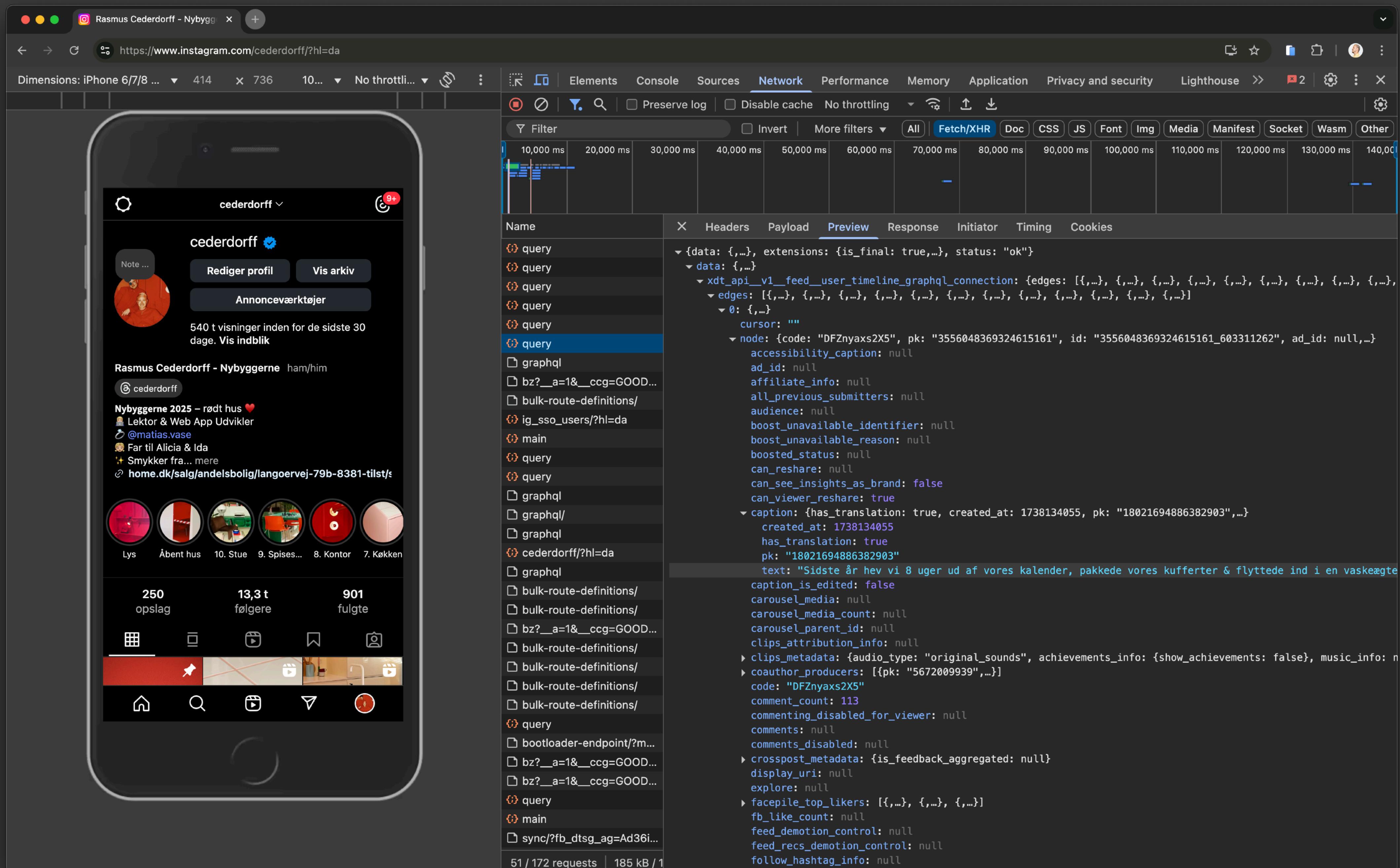
Rasmus Cederdorff

Lecturer
race@baaa.dk









Course roster: WU-E22a - 1. se X +

https://eaaa.instructure.com/courses/15482/users

WU-E22a > People

60 Student view

Home Announcements Modules Assignments Discussions People BigBlueButton Grades Pages Files Syllabus Outcomes Rubrics Quizzes Collaborations Settings

Everyone Groups

+ Group set

Search people All roles + People

Name	Login ID	SIS ID	Section	Role	Last Activity	Total Activity
Clara Juul Birk	eaaclbi@students.eaaa.dk	WU-E22a - 1.	Student semester	Student	24 Aug at 13:16	01:04:21
Martin Rieper Boesen	eaamrbo@students.eaaa.dk	WU-E22a - 1.	Student semester	Student	24 Aug at 7:54	01:07:06
Dan Okkels Brendstrup	dob@eaaa.dk	WU-E22a - 1.	Teacher semester	Teacher	3 Aug at 8:55	
Rasmus Cederdorff	race@eaaa.dk	WU-E22a - 1.	Teacher semester	Teacher	25 Aug at 9:28	01:19:23
Jeffrey David Serio	jds@eaaa.dk	WU-E22a - 1.	Teacher semester	Teacher	17 Aug at 16:39	
Charlotte Meng Emanuel Dyrholm	eaacmed@students.eaaa.dk	WU-E22a - 1.	Student semester	Student	23 Aug at 16:59	22:24

Elements Components Network 22 Preset log Disable cache No throttling Invert Hide data URLs

All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other

Has blocked cookies Blocked Requests 3rd-party requests

5000 ms 10000 ms 15000 ms 20000 ms 25000 ms 30000 ms 35000 ms

Name key Headers Payload Preview Response Initiator >

value

?sentry_ke... en-GB-84e... unread_co... group_cate... users?inclu... unread_co...

[{id: "23974", name: "Clara Juul Birk", created_at: "2020-08-14T10:45:00+02:00", login_id: "eaaclbi@students.eaaa.dk", email: "dob@eaaa.dk", integration_id: null, enrollment_id: null, short_name: "Clara Juul Birk", sortable_name: "Birk, Clara Juul"}, {id: "36267", name: "Martin Rieper Boesen", created_at: "2020-08-14T10:45:00+02:00", login_id: "eaamrbo@students.eaaa.dk", email: "dob@eaaa.dk", integration_id: null, enrollment_id: null, short_name: "Boesen, Martin Rieper"}, {id: "29923", name: "Dan Okkels Brendstrup", created_at: "2021-07-30T00:46:05+02:00", login_id: "dob@eaaa.dk", email: "dob@eaaa.dk", integration_id: null, enrollment_id: "391064", user_id: "29923", course_id: "15482", short_name: "Brendstrup, Dan Okkels"}, {id: "14427", name: "Rasmus Cederdorff", created_at: "2021-07-30T00:46:05+02:00", login_id: "race@eaaa.dk", email: "race@eaaa.dk", integration_id: null, enrollment_id: null, short_name: "Cederdorff, Rasmus"}, {id: "41", name: "Jeffrey David Serio", created_at: "2021-07-30T00:46:05+02:00", login_id: "jds@eaaa.dk", email: "jds@eaaa.dk", integration_id: null, enrollment_id: null, short_name: "Serio, Jeffrey David"}, {id: "24043", name: "Charlotte Meng Emanuel Dyrholm", created_at: "2021-07-30T00:46:05+02:00", login_id: "eaacmed@students.eaaa.dk", email: "eaacmed@students.eaaa.dk", integration_id: null, enrollment_id: null, short_name: "Dyrholm, Charlotte Meng Emanuel"}]

Objects? Arrays?

The screenshot shows the homepage of DR Nyheder. At the top, there are navigation links for NYHEDER, DRTV, and DR LYD. Below the navigation, there are six thumbnail cards for TV shows: DR1: Løvens Hule, DR3: Nationens stærkeste, P1: LSD kælderen, DR LYD: Annas Margrethe, DR3: Du fucker med de forkerte, and A Very British Scandal. Under these, a section titled "Seneste nyt" (Latest news) displays three news items: "EU klager over Kinas hårde kurs over for Litauen" (5 MIN. SIDEN), "Børn og skoleelever opfordres stadig til to ugentlige coronatest" (13 MIN. SIDEN), and "England skrætter størstedelen af coronarestriktionerne fra i dag" (25 MIN. SIDEN). The main content area features a large image of medical supplies (a mask, a thermometer, a syringe, and a bottle of hand sanitizer) against a blue background, with the text "15 lande bakker Danmark op: Danske soldater skal blive i Mali" overlaid. At the bottom, a red banner reads "Regeringen har meldt genåbning - men ikke".

The screenshot shows the "ALLE ERHVERVSAKADEMI-UDDANNELSER" (All Business Academy Programs) page. The page has two main navigation links: "ALLE UDDANNELSER" and "UDDANNELSER UD FRA INTERESSE". Below the title, there is a grid of 12 program profiles, each featuring a student's portrait and the program name. Each profile includes a small arrow icon pointing right, indicating a link to more information. The programs listed are: AUTOMATIONSTEKNOLOG, BYGGEKOORDINATOR, BYGGETEKNIKER, DATAMATIKER, DESIGNTEKNOLOG, ENTREPRENØRSKAB OG DESIGN, EL-INSTALLATOR, ENERGITEKNOLOG, IT-TEKNOLOG, KORT- OG LANDMÅLING, MULTIMEDIEDESIGNER, and VVS-INSTALLATOR.

Objects with properties in arrays

The screenshot shows a web browser window for the Business Academy Aarhus website (baaa.dk/programmes/). The page displays various study programs:

- Programmes at Business Academy Aarhus**
 - Study start in August**
 - Multimedia Design**: AP degree - 2 years. For those who would like to work with digital communication and interactive design. The programme is the first part of a Bachelor's programme.
 - Digital Concept Development**: Bachelor's top-up degree - 1½ years. Get additional qualifications to develop concepts for digital platforms - at both the strategic and the practical level.
 - Study start in January**
 - IT Technology**: AP degree (Final intake with study start in January 2022). Would you like to work with computers, server and network technology? The programme is the first part of a Bachelor's programme.
 - Chemical and Biotechnical Technology and Food Technology**: Bachelor's top-up degree (Final intake with study start in January 2022). Be successful in both national and international laboratory environments, and get updated on the
 - Web Development**: Bachelor's top-up degree (Final intake with study start in January 2022). Focus on the development of web technologies within several application fields and distribution platforms.
 - Programmes that no longer accept new applicants**
 - Chemical and Biotechnical Science**: AP degree (We no longer accept new applicants for this programme).
 - Marketing Management**: AP degree (We no longer accept new applicants for this programme).

A "Chat now" button is located in the bottom right corner.

It's all objects &
arrays!

Data Types & Data Structures

Objects & Arrays

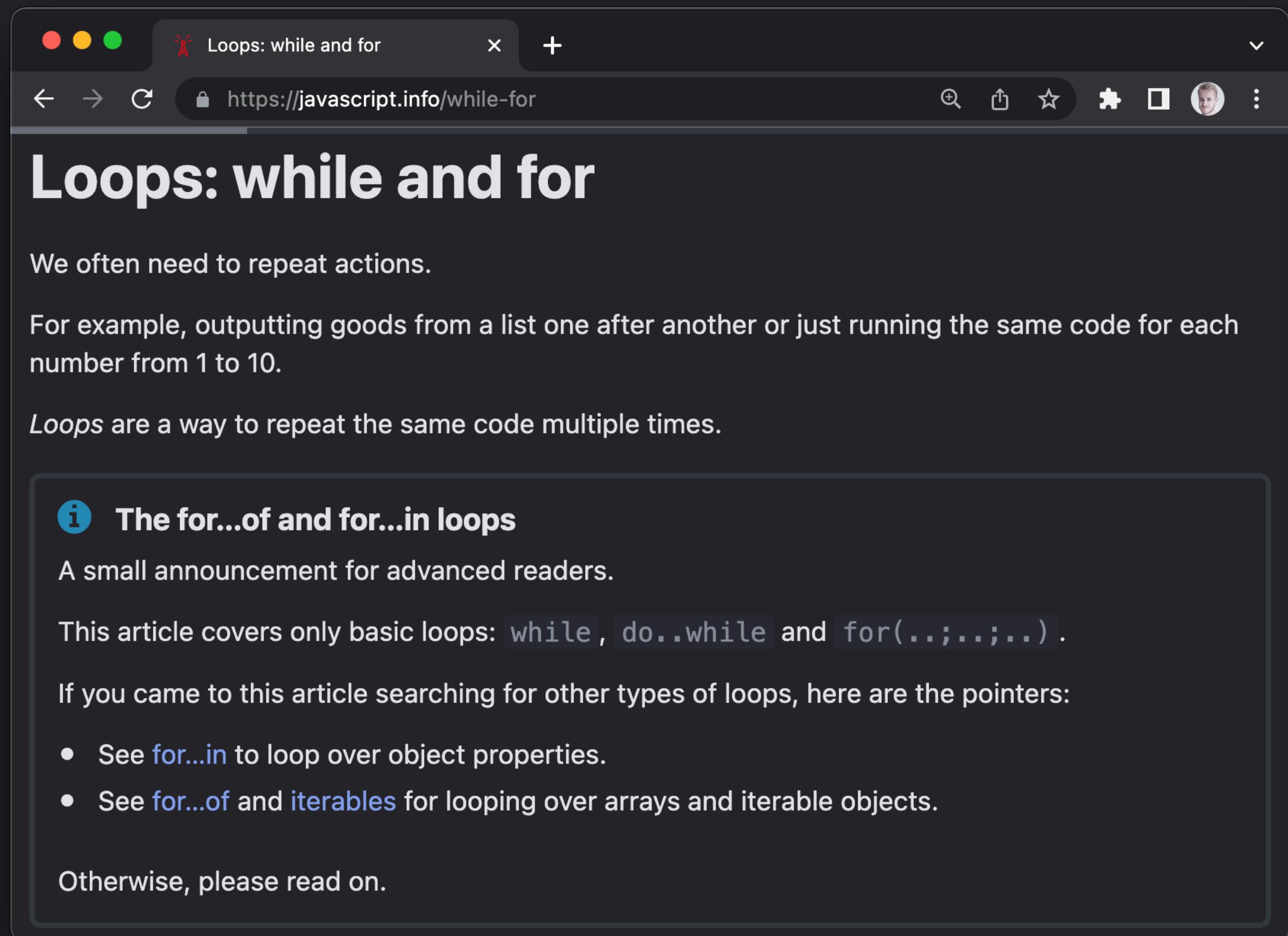
Arrays

Loops

```
for (let teacher of teachers) {  
  console.log(teacher);  
}
```

```
▶ {name: "Birgitte Kirk Iversen", mail: "bki@baaa.dk"}  main.js:20  
▶ {name: "Michael Hvidtfeldt", mail: "mhv@baaa.dk"}  main.js:20  
▶ {name: "Rasmus Cederdorff", mail: "race@baaa.dk"}  main.js:20
```

Loops



The screenshot shows a dark-themed web browser window. The title bar reads "Loops: while and for". The address bar shows the URL "https://javascript.info/while-for". The main content area has a large heading "Loops: while and for". Below it, a paragraph states "We often need to repeat actions. For example, outputting goods from a list one after another or just running the same code for each number from 1 to 10. Loops are a way to repeat the same code multiple times." A callout box with a blue border and rounded corners contains the following text: "i The for...of and for...in loops A small announcement for advanced readers. This article covers only basic loops: `while`, `do..while` and `for(..;..;..)`. If you came to this article searching for other types of loops, here are the pointers: • See [for...in](#) to loop over object properties. • See [for...of](#) and [iterables](#) for looping over arrays and iterable objects. Otherwise, please read on." The browser interface includes standard controls like back, forward, and search.

Loops: while and for

We often need to repeat actions. For example, outputting goods from a list one after another or just running the same code for each number from 1 to 10. Loops are a way to repeat the same code multiple times.

i The for...of and for...in loops

A small announcement for advanced readers.

This article covers only basic loops: `while`, `do..while` and `for(..;..;..)`.

If you came to this article searching for other types of loops, here are the pointers:

- See [for...in](#) to loop over object properties.
- See [for...of](#) and [iterables](#) for looping over arrays and iterable objects.

Otherwise, please read on.

For of loop

iterate over arrays or other iterable objects

<https://scrimba.com/learn/introductiontojavascript/for-loops-cMMM8U9>

<https://scrimba.com/learn/introductiontojavascript/challenge-for-loops-cPkpJrcv>

Loops

```
for (const familyMember of familyMembers) {  
    console.log(familyMember);  
}
```

```
for (let index = 0; index < familyMembers.length; index++) {  
    const familyMember = familyMembers[index];  
    console.log(familyMember);  
}
```

https://www.w3schools.com/js/js_loop_for.asp
<https://javascript.info/array#loops>
<https://javascript.info/while-for>

JavaScript.info/array#loops

One of the oldest ways to cycle array items is the `for` loop over indexes:

```
1 let arr = ["Apple", "Orange", "Pear"];
2
3 for (let i = 0; i < arr.length; i++) {
4   alert( arr[i] );
5 }
```



But for arrays there is another form of loop, `for..of`:

```
1 let fruits = ["Apple", "Orange", "Plum"];
2
3 // iterates over array elements
4 for (let fruit of fruits) {
5   alert( fruit );
6 }
```



```
const allMovies = [
  {
    id: 1,
    titel: "The Matrix",
    år: 1999,
    rating: 8.7,
    genre: ["Action", "Sci-Fi"],
    instruktører: ["Lana Wachowski", "Lilly Wachowski"]
  },
  {
    id: 2,
    titel: "Inception",
    år: 2010,
    rating: 8.8,
    genre: ["Action", "Thriller", "Sci-Fi"],
    instruktører: ["Christopher Nolan"]
  },
  {
    id: 3,
    titel: "The Dark Knight",
    år: 2008,
    rating: 9.0,
    genre: ["Action", "Crime", "Drama"],
    instruktører: ["Christopher Nolan"]
  }
];

// Loop through all movies
for (const movie of expandedMovieDatabase) {
  console.log(`🎬 ${movie.title} (${movie.year})`);
  console.log(`⭐ Rating: ${movie.rating}`);
  console.log(`🎭 Genre: ${movie.genre[0]}`);
  console.log("----");
}
```

ARRAYS

LOOPS

```
for (let teacher of teachers) {  
  console.log(teacher.mail);  
}
```

bki@baaa.dk

main.js:20

mhv@baaa.dk

main.js:20

race@baaa.dk

main.js:20

LOOPS

... LOOP THROUGH AN ARRAY AND ADD A CONDITION

```
for (let teacher of teachers) {  
  if (teacher.name === "Rasmus Cederdorff") {  
    console.log(teacher);  
  }  
}
```

Functions

Write reusable code

Functions

A block of code to perform a specific task.

A way to make reusable code by storing tasks we can use again and again.

Best practice: write reusable code

```
function log(message) {  
  console.log(message);  
}  
  
log("Hi Frontenders!");
```

<https://javascript.info/function-basics>

Functions

3 different types

```
function log(message) {  
    console.log(message);  
}
```

FUNCTION DECLARATION

```
const log = function (message) {  
    console.log(message);  
};
```

FUNCTION EXPRESSION

```
const log = (message) => {  
    console.log(message);  
};
```

ARROW FUNCTION

Functions

Function declaration

```
console.log("Hi Frontenders!");
console.log("Good job!");
console.log("I'm testing something!");
console.log("Hola");
```

```
function log(message) {
  console.log(message);
}

log("Hi Frontenders!");
log("Good job!");
log("I'm testing something!");
log("Hola");
```

The screenshot shows a web browser window with the title bar "JavaScript Functions". The address bar contains the URL "https://www.w3schools.com/js/js_functions.asp". The navigation bar includes links for Home, HTML, CSS, JAVASCRIPT (which is highlighted in green), SQL, PYTHON, PHP, and BOOTSTRAP. There are also icons for search, refresh, and user profile.

JavaScript Function Syntax

A JavaScript function is defined with the `function` keyword, followed by a **name**, followed by parentheses `()`.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
`(parameter1, parameter2, ...)`

The code to be executed, by the function, is placed inside curly brackets: `{}`

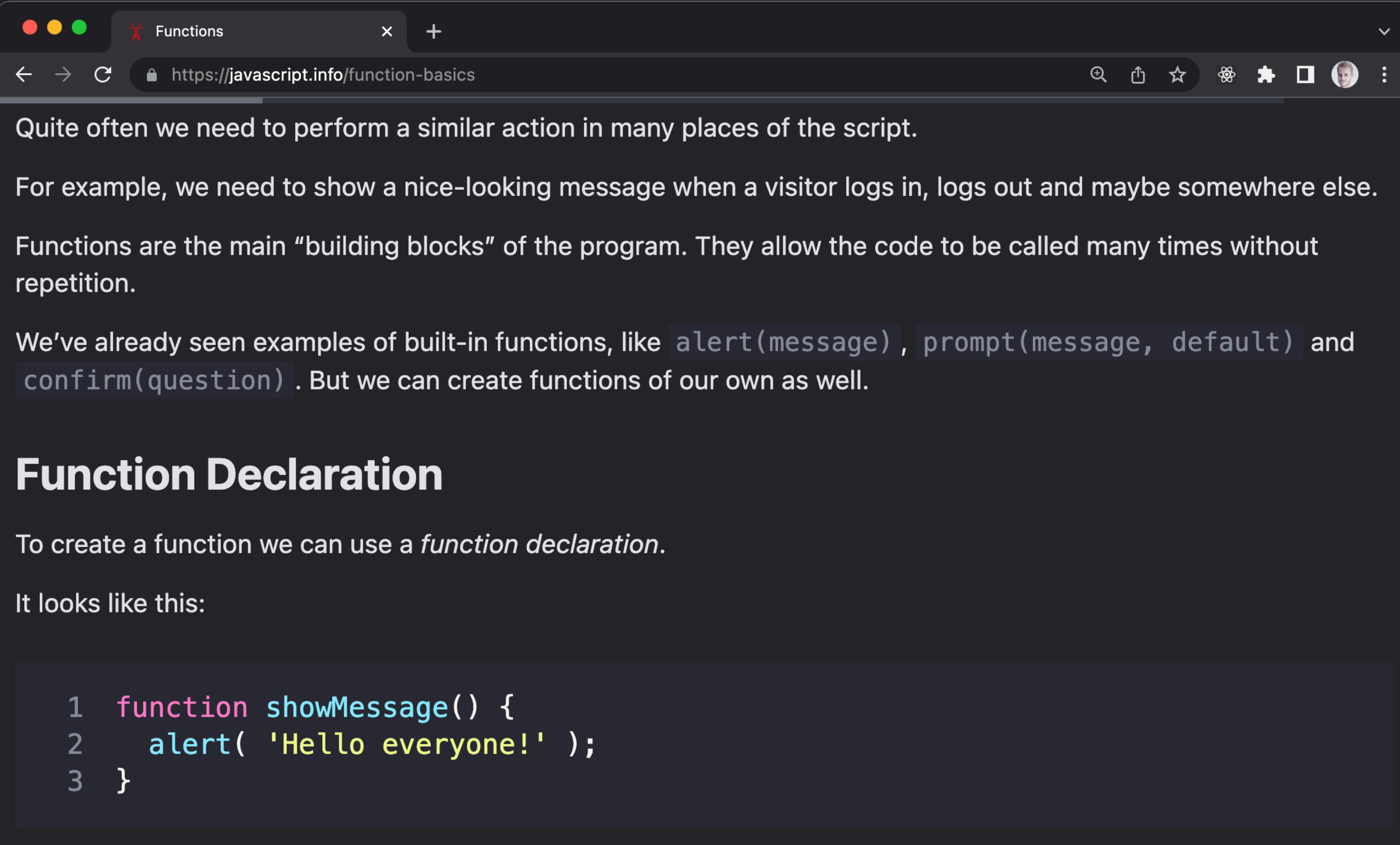
```
function name(parameter1, parameter2, parameter3) {  
    // code to be executed  
}
```

Function **parameters** are listed inside the parentheses `()` in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

JavaScript.info/Function-Basics



The screenshot shows a dark-themed web browser window with the title bar "Functions". The address bar displays the URL "https://javascript.info/function-basics". The main content area contains text explaining the purpose and benefits of functions, followed by a code example.

Quite often we need to perform a similar action in many places of the script.
For example, we need to show a nice-looking message when a visitor logs in, logs out and maybe somewhere else.
Functions are the main “building blocks” of the program. They allow the code to be called many times without repetition.
We've already seen examples of built-in functions, like `alert(message)`, `prompt(message, default)` and `confirm(question)`. But we can create functions of our own as well.

Function Declaration

To create a function we can use a *function declaration*.

It looks like this:

```
1 function showMessage() {  
2     alert( 'Hello everyone!' );  
3 }
```

Functions

Function declaration

The name of the function



```
function showMessage() {  
    alert("Hello everyone!");  
}
```



Body of the function
(code block)

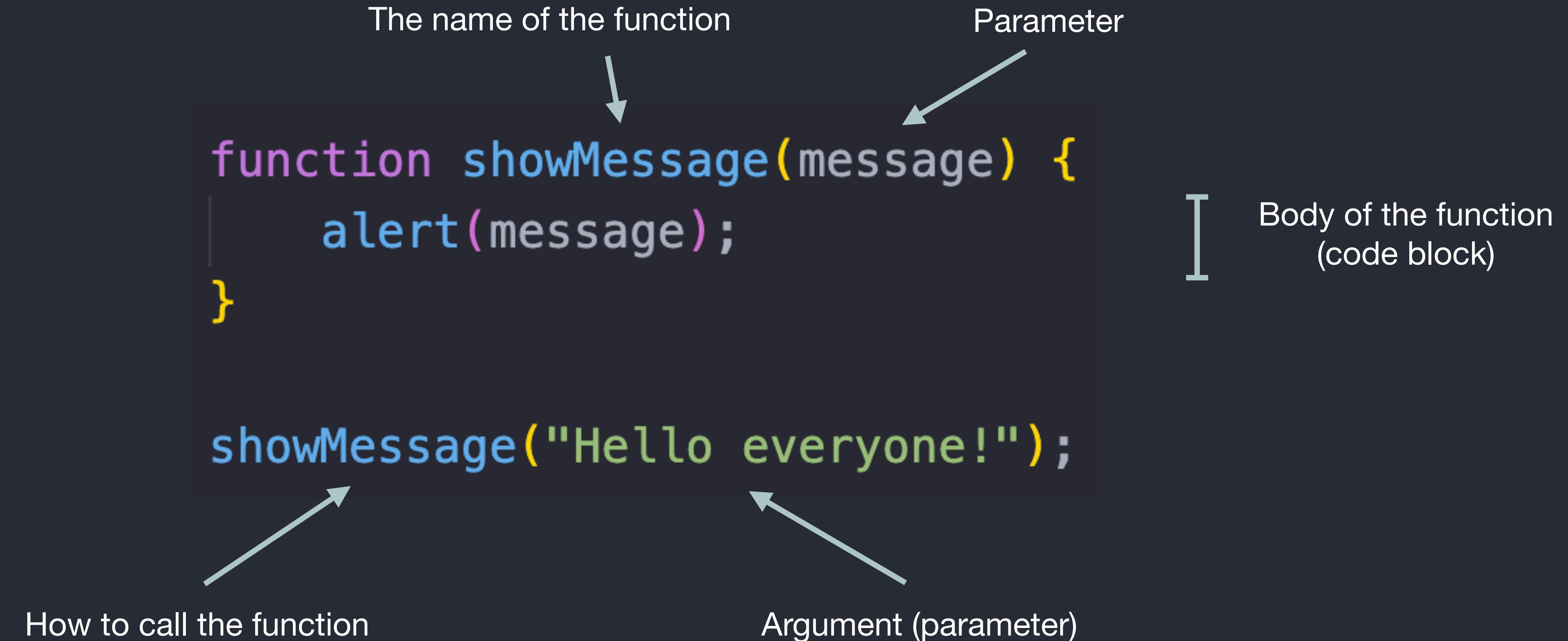
showMessage();

How to call the function



Functions

Function declaration



Functions

Function declaration

```
function showMessage(message) {  
    alert(message);  
}  
  
showMessage("Hello everyone!");  
showMessage("How are you?");  
showMessage("Good, you?");
```



The function can be called as many times as you want
And with different argument values

*"When a value is passed as a function parameter,
it's also called an argument.*

In other words, to put these terms straight:

- *A parameter is the variable listed inside the parentheses in the function declaration (it's a declaration time term).*
- *An argument is the value that is passed to the function when it is called (it's a call time term)."*

JavaScript.info/function-basics#parameters

We can pass arbitrary data to functions using parameters.

In the example below, the function has two parameters: `from` and `text`.

```
1 function showMessage(from, text) { // parameters: from, text
2   alert(from + ': ' + text);
3 }
4
5 showMessage('Ann', 'Hello!'); // Ann: Hello! (*)
6 showMessage('Ann', "What's up?"); // Ann: What's up? (**)
```

Functions

Arrays & Loops

The name of the function



Parameters



```
function appendTeachers(teachers) {  
  for (let teacher of teachers) {  
    console.log(teacher);  
    document.querySelector("#grid-teachers").innerHTML +=  
      "<article>" +  
      "<img src=''" + teacher.img + "'>" +  
      "<h3>" + teacher.name + "</h3>" +  
      teacher.position + "<br>" +  
      "<a href='mailto:" + teacher.mail + "'>" + teacher.mail + "</a>" +  
      "</article>";  
  }  
}  
  
appendTeachers(teachers);
```

How to call the function

Body of the function
(code block)

TEACHERS



Birgitte Kirk Iversen

Senior Lecturer
bki@baaa.dk



Michael Hvidtfeldt

Senior Lecturer
mhv@baaa.dk



Rasmus Cederdorff

Lecturer
race@baaa.dk

DOM Manipulation

Change the content of the website with JavaScript

DOM Manipulation

```
// declaring a variable with a value
let message = "Hi Frontenders!"

//accessing the variable and logging it to the console
console.log(message);

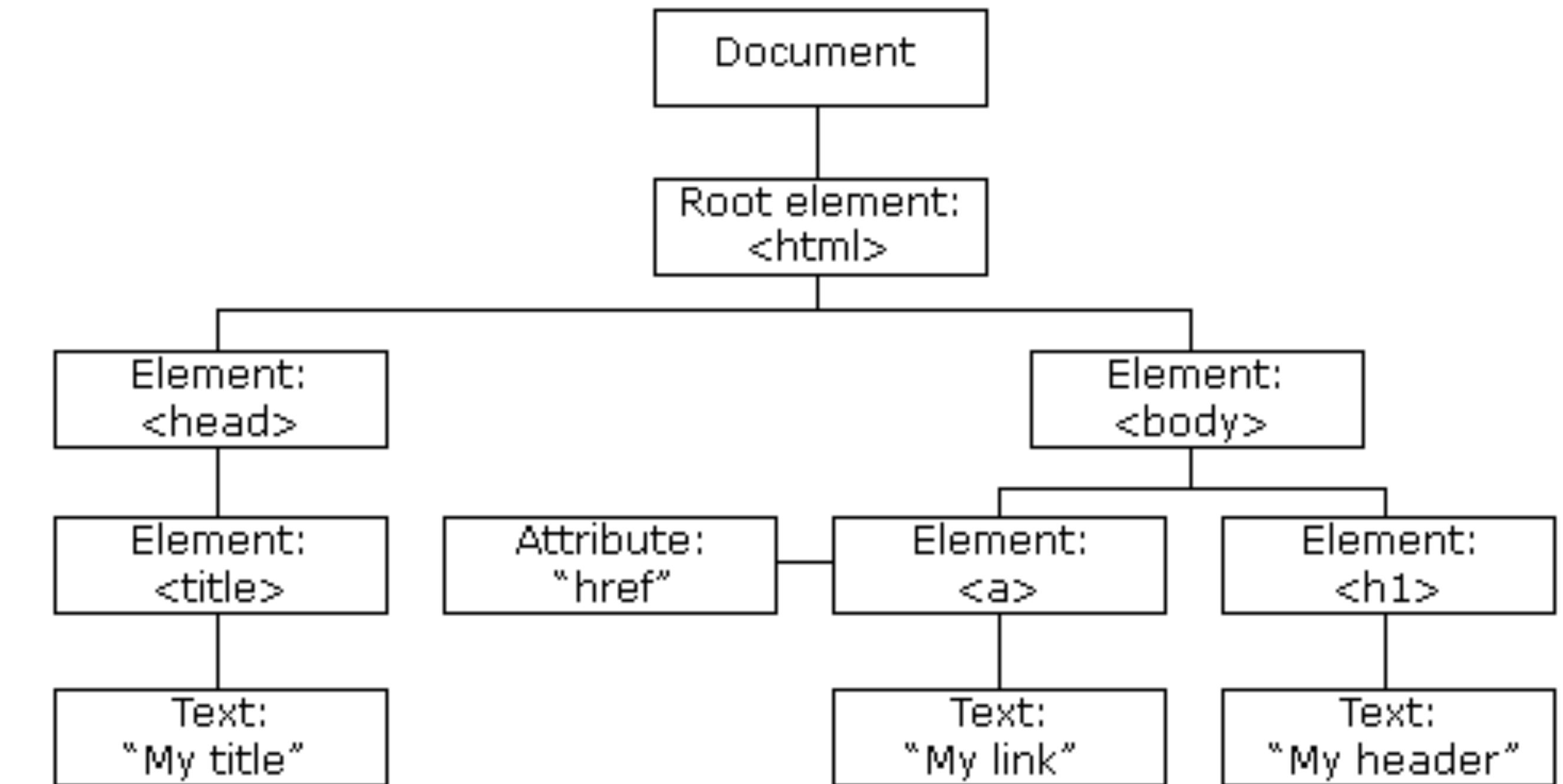
// appending the variable (the string) to the DOM element #content
document.querySelector("#content").innerHTML = message;
```

```
<body>
  <header>
    <h1>PROJECT TEMPLATE</h1>
  </header>
  <section id="content"></section>
  <!-- main is file -->
  <script src="js/main.js"></script>
</body>
```



JavaScript HTML DOM

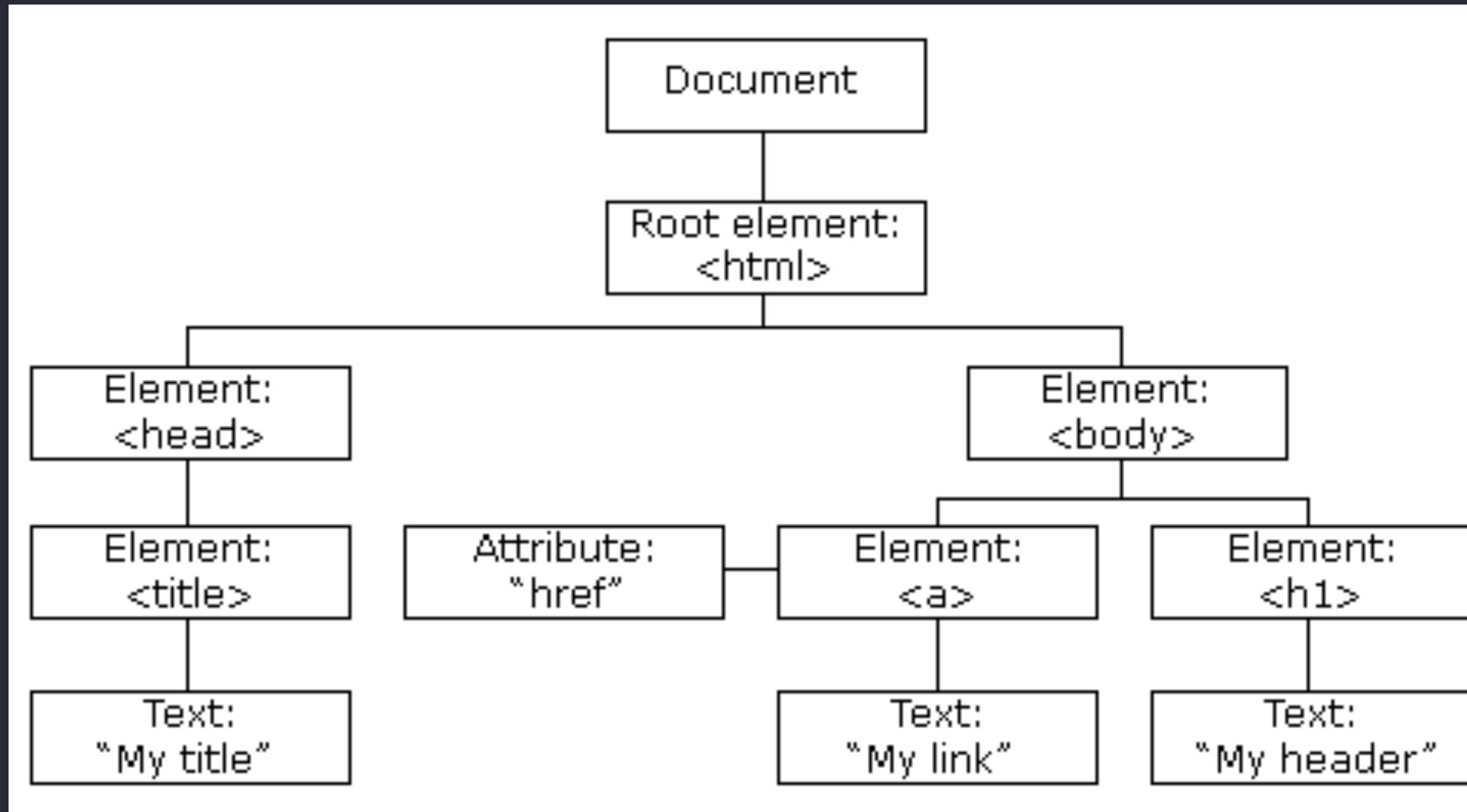
```
index.html *  
1  <!DOCTYPE html>  
2  <html>  
3  | <head>  
4  | | <title>My title</title>  
5  | </head>  
6  |  
7  <body>  
8  | | <h1>My header</h1>  
9  | | <a href="https://cederdorff.com">My link</a>  
10 | </body>  
11 |  
12 </html>
```



https://www.w3schools.com/js/js_htmldom.asp
<https://javascript.info/dom-nodes>
<https://javascript.info/dom-navigation>

The HTML DOM (Document Object Model)

A model as a tree of Objects



- Object Model for HTML:
 - HTML elements as objects
 - Properties for all HTML elements
 - Methods for all HTML elements
 - Events for all HTML elements

JavaScript HTML DOM

Document Object Model

```
index.html ×  
1  <!DOCTYPE html>  
2  <html>  
3  |   <head>  
4  |   |   <title>My title</title>  
5  |   </head>  
6  
7  <body>  
8  |   <h1>My header</h1>  
9  |   <a href="https://cederdorff.com">My link</a>  
10 |</body>  
11 </html>  
12
```

The HTML document as an object

Gives us the power to create dynamic HTML and manipulate with the HTML (the DOM).

JavaScript can:

- ... change all the HTML elements in the page*
- ... change all the HTML attributes in the page*
- ... change all the CSS styles in the page*
- ... remove existing HTML elements and attributes*
- ... add new HTML elements and attributes*
- ... react to all existing HTML events in the page*
- ... create new HTML events in the page*

https://www.w3schools.com/js/js_htmldom.asp

<https://javascript.info/dom-nodes>

<https://javascript.info/dom-navigation>

Searching the DOM: getElement* & querySelector*

```
<section id="elem">
  <article id="elem-content">Element</article>
</section>

<script>
  // get the element
  const element = document.getElementById('elem');
  // make its background red
  element.style.background = 'red';
  // get the elementContent
  const elementContent = document.querySelector('#elem-content');
  // change inner HTML
  elementContent.innerHTML = "<h2>Hi Web Developers!</h2>"
</script>
```

Searching the DOM: getElementsByTagName*

```
<section id="elem">
  <article class="elem-content">Element</article>
  <article class="elem-content">Element</article>
  <article class="elem-content">Element</article>
</section>

<script>
  // get all elements matching the selector - returns an array
  const elements = document.getElementsByTagName('elem-content');
  // loop through all elements
  for (const element of elements) {
    element.innerHTML = "<h2>Hi Web Developers!</h2>";
  }
</script>
```

Searching the DOM: querySelectorAll

```
<section id="elem">
  <article class="elem-content">Element</article>
  <article class="elem-content">Element</article>
  <article class="elem-content">Element</article>
</section>

<script>
  // get all elements matching the selector - returns an array
  const elements = document.querySelectorAll('.elem-content');
  // loop through all elements
  for (const element of elements) {
    element.innerHTML = "<h2>Hi Web Developers!</h2>";
  }
</script>
```

JS HTML DOM

`getElements*` or `querySelector*`?

CSS Grid

CSS Grid is a two-dimensional layout system in CSS that allows you to arrange content in rows and columns with precision and flexibility.

CSS Layouts

Introduction to CSS layout

Normal flow

The display property

Flexbox

Grid

Floats

Positioning

Table layout

Multiple-column layout

Grid & Flexbox

... two CSS layout models that can be used to create layouts with just a couple of CSS rules.

... responsive and flexible layouts.

CSS Grids

CSS Grid Teachers

The screenshot shows a grid layout with three columns. Each column contains a portrait photo and the name of a teacher below it. The first column has 'Birgitte Kirk Iversen' (Senior Lecturer, hki@mail.dk). The second column has 'Martin Aagaard Nøhr' (Lecturer, mnor@mail.dk). The third column has 'Peter Lind' (Senior Lecturer, pell@kea.dk).

Header

The screenshot shows a grid layout with three columns. The top row is labeled 'Header' and the bottom row is labeled 'Footer'. Each column is labeled 'Column'.

Amerikanske chocolate chip cookies

The screenshot shows a grid layout for a cookie recipe. The left side has a 'Cookies' section with text about classic American cookies. The right side has a large image of chocolate chip cookies. Below the image is a 'Sådan gør du' (How to do it) section with a 'Cookies' heading and a numbered list of steps. At the bottom is a 'Bagetid' (Baking time) section.

Cookies

Klassiske amerikanske cookies - sprøde udenpå og bløde indeni... Og hvis der er noget du virkelig kan, derover på den anden side af atlanten, så er det at bage cookies. Om vi kalder dem Chocolate chip cookies eller bare Cookies ger nok ikke den store forskel. Der er chokoladestykker i - og de smager prægtfuldt. Prøv denne opskrift på den perfekte Chocolate chip cookie, næste gang lækkersulten banker på døren!

Ingredienser

Cookies

- Bledd smør, 150 g
- Sukker, (ca. 1½ dl) 125 g
- Brun farin, (ca. 1½ dl) 125 g
- Vanillesukker, 2 tsk
- Æg, 1
- Hvedemel (ca. 4½ dl), 250 g
- Natron, 1 tsk
- Gris salt, ¼ tsk
- Begepulver, ½ tsk

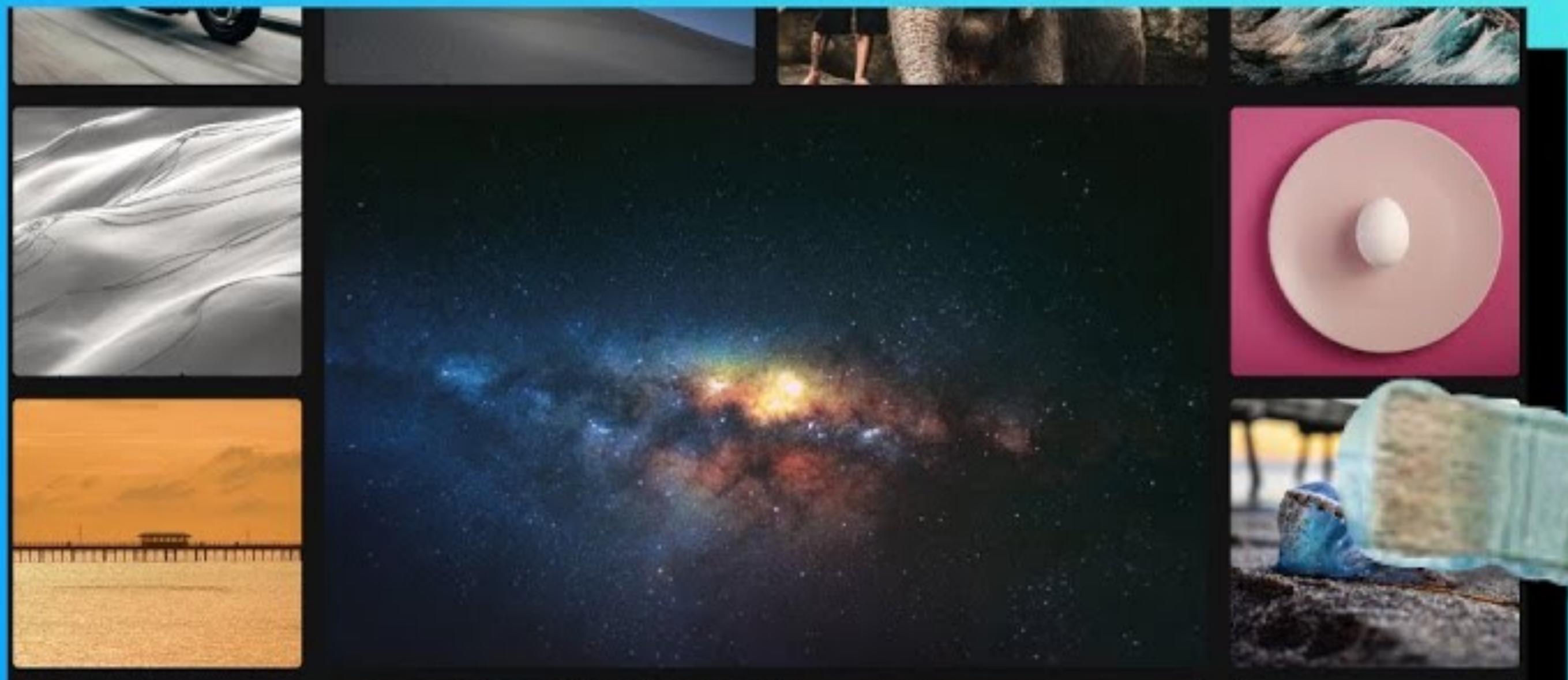
Sådan gør du

1. Pisk smør, sukker, brun farin, vanille og æg godt sammen.
2. Bland hvedemel, natron, salt og begepulver og før det i smør blandingen sammen med chokolade.
3. Del dejen i 20 stykker a ca. 45 g og form dem til kugler.
4. Sat kuglerne på plader med bagepapir - tryk dem let flate. Kagerne fylder en del ud, så sørг for, at der er god plac i mellem dem.
5. Bag kagerne til de er gyldne, men bløde i midten.
6. Lad kagerne afkøle ca. 5 min. påbagepladen og flyt dem så over på en rist.

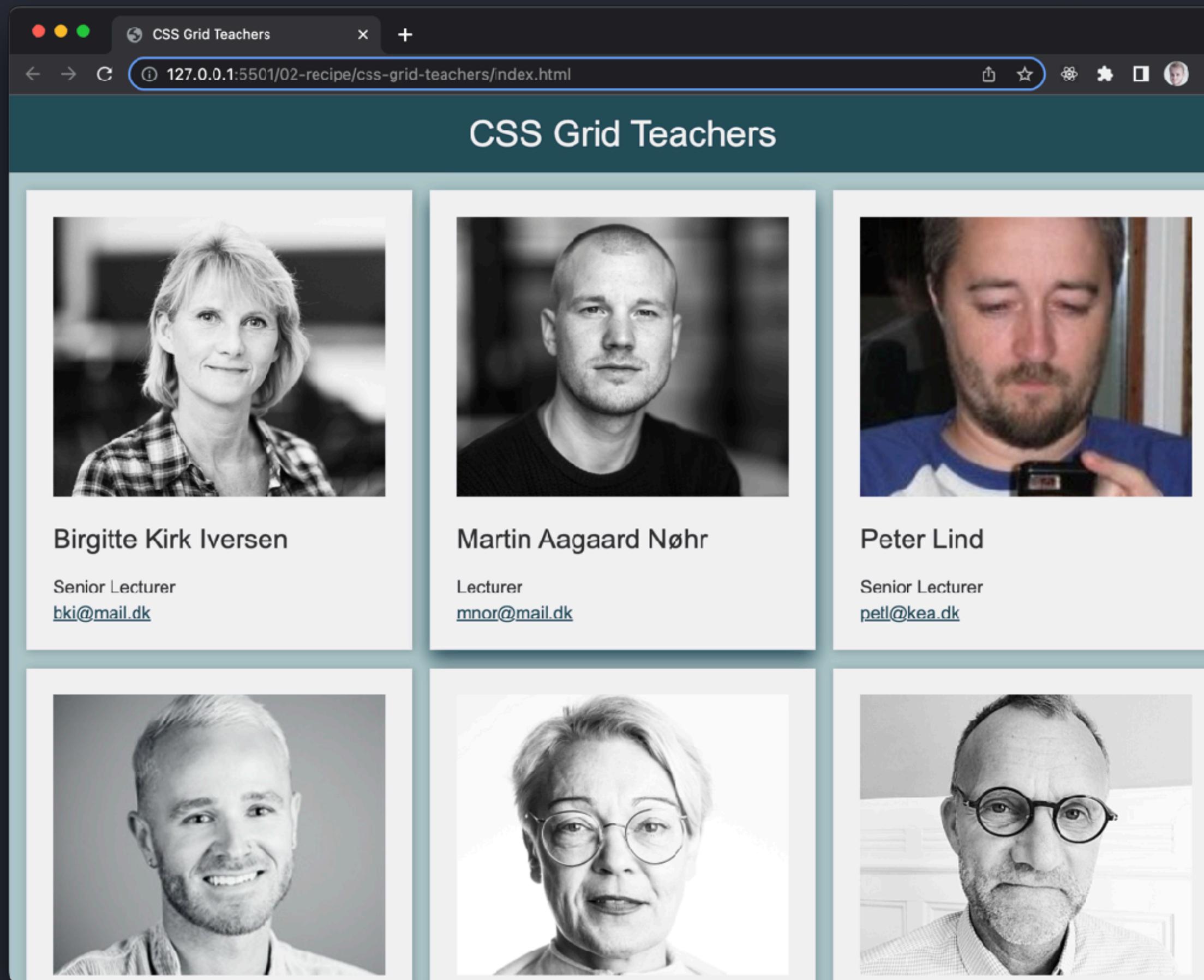
Bagetid

The Joy of

GOING



CSS Grid



```
<section class="grid-container">...</section>

/* ----- grid container styling ----- */
.grid-container {
  display: grid;
  grid-template-columns: 1fr;
  gap: 1em;
  padding: 1em;
}

@media (min-width: 600px) {
  .grid-container {
    grid-template-columns: 1fr 1fr;
  }
}

@media (min-width: 992px) {
  .grid-container {
    grid-template-columns: 1fr 1fr 1fr;
  }
}
```

CSS Grid

The screenshot shows a web page titled "Person Array". It features a 2x2 grid of portrait photos and names. The top-left cell contains a photo of Birgitte Kirk Iversen, with the name "Birgitte Kirk Iversen" and title "Senior Lecturer" below it. The top-right cell contains a photo of Martin Aagaard Nøhr, with the name "Martin Aagaard Nøhr" and title "Lecturer" below it. The bottom-left cell contains a photo of Rasmus Cederdorff, with the name "Rasmus Cederdorff" and title "Senior Lecturer" below it. The bottom-right cell contains a photo of Dan Okkels Brendstrup, with the name "Dan Okkels Brendstrup" and title "Lecturer" below it. The browser's developer tools are open, showing the DOM structure and CSS styles for the grid container.

```
<section id="content" class="grid-container"></section>
```

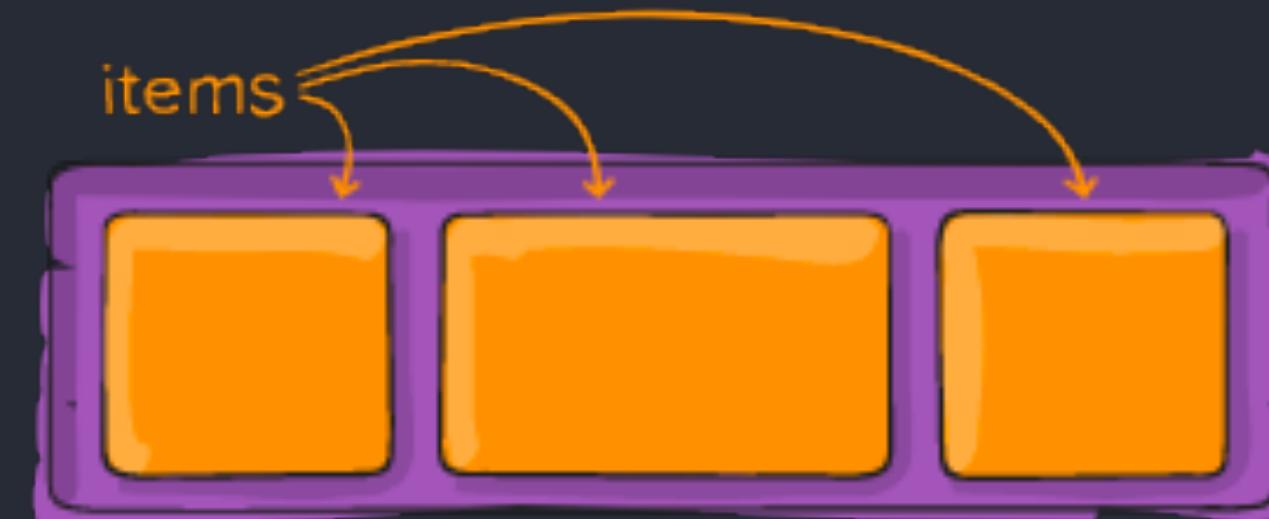
```
/* ----- grid container styling ----- */
.grid-container {
  display: grid;
  grid-template-columns: 1fr;
  gap: 1em;
}

@media (min-width: 600px) {
  .grid-container {
    grid-template-columns: 1fr 1fr;
  }
}

@media (min-width: 992px) {
  .grid-container {
    grid-template-columns: 1fr 1fr 1fr;
  }
}
```

```
section#content.grid-container 910x740.25
grid = $0
grid-template-columns: 1fr
gap: 1em
grid-template-columns: 1fr 1fr
grid-template-columns: 1fr 1fr 1fr
```

CSS Grid



```
/* ----- grid container styling ----- */
.grid-container {
  display: grid;
  grid-template-columns: 1fr;
  gap: 1em;
  padding: 1em;
}

@media (min-width: 600px) {
  .grid-container {
    grid-template-columns: 1fr 1fr;
  }
}

@media (min-width: 992px) {
  .grid-container {
    grid-template-columns: 1fr 1fr 1fr;
  }
}
```

```
.grid-item {
  box-shadow: 1px 1px 8px □rgba(0, 0, 0, 0.25);
  padding: 1.5em;
  background-color: ■#f1f1f4;
  transition: 0.5s;
  animation: fadeIn 0.5s;
}

.grid-item:hover {
  box-shadow: 0 8px 16px 0 □rgb(38, 76, 89);
}

.grid-item img {
  width: 100%;
  height: 250px;
  object-fit: cover;
}

.grid-item p {
  margin: 0.3em 0;
```

Grid Container

CSS Grid Layout

w3schools.com/css/css_grid.asp

HTML CSS JAVASCRIPT

Display Property

An HTML element becomes a grid container when its `display` property is set to `grid` or `inline-grid`.

Example

```
.grid-container {  
  display: grid;  
}
```

[Try it Yourself »](#)

Example

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

[Try it Yourself »](#)

All direct children of the grid container automatically become *grid items*.

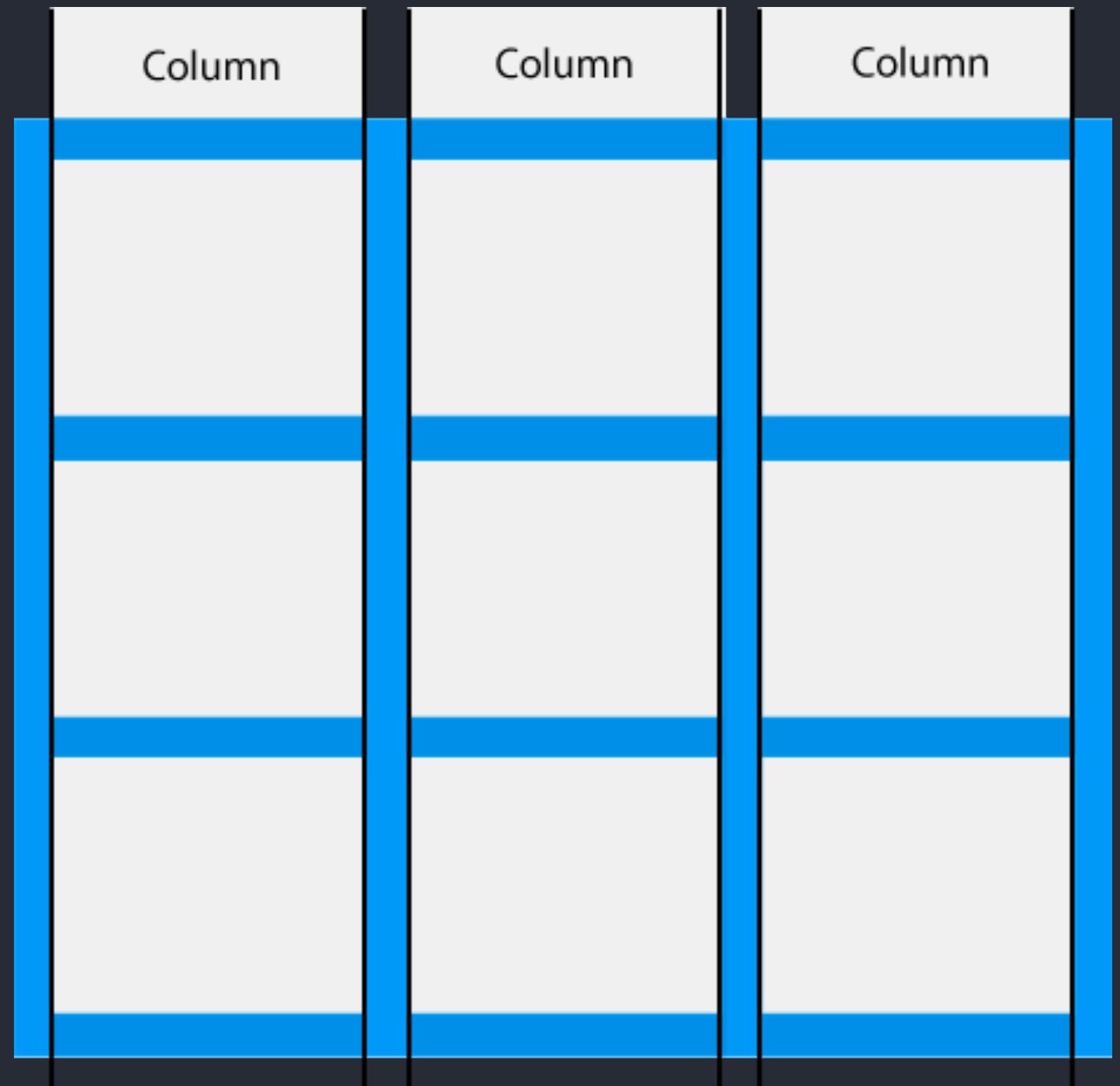
Example

```
<div class="grid-container">  
  <div class="grid-item">1</div>  
  <div class="grid-item">2</div>  
  <div class="grid-item">3</div>  
  <div class="grid-item">4</div>  
  <div class="grid-item">5</div>  
  <div class="grid-item">6</div>  
  <div class="grid-item">7</div>  
  <div class="grid-item">8</div>  
  <div class="grid-item">9</div>  
</div>
```



Grid Columns

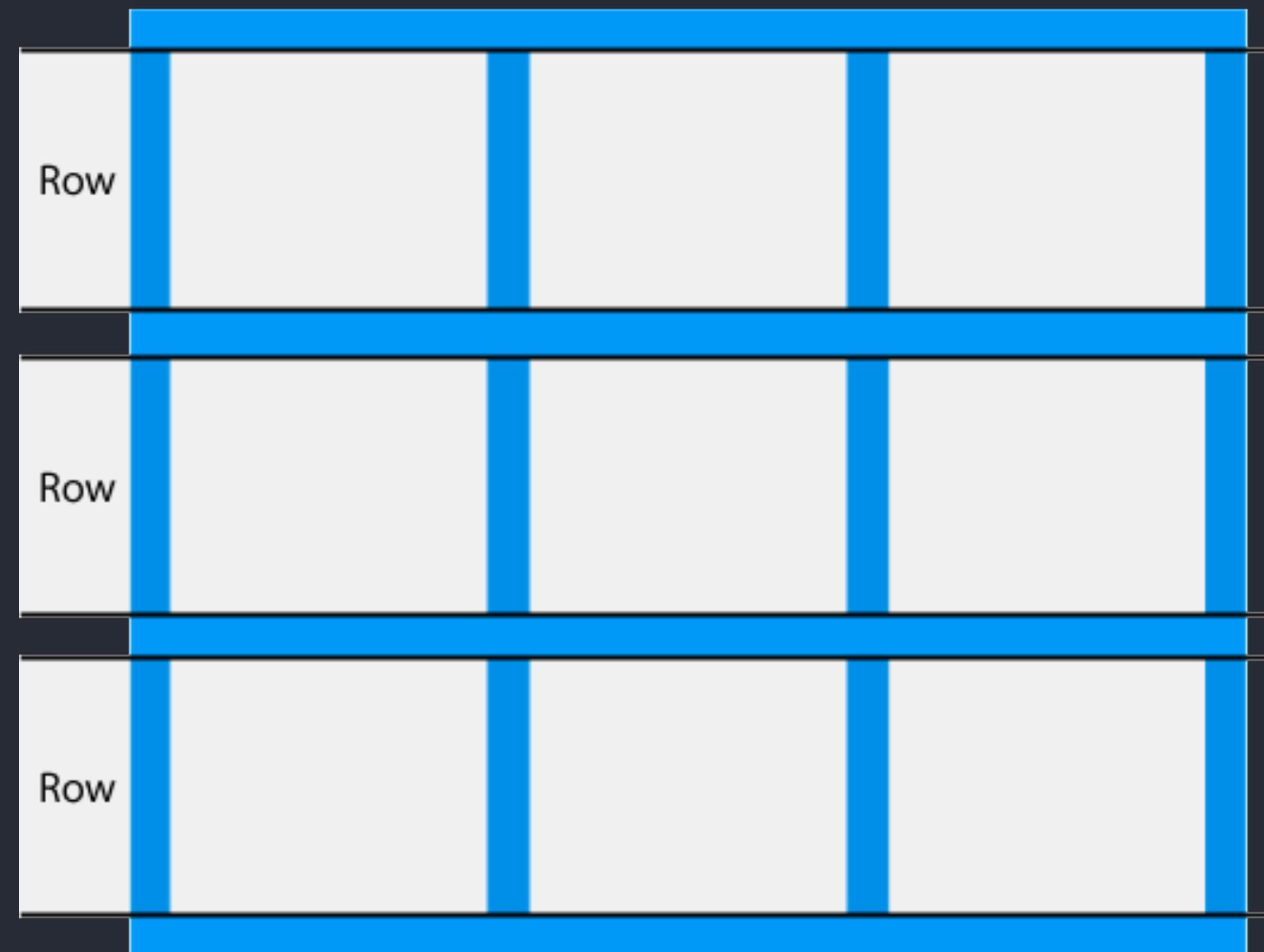
Vertical Lines



```
.grid-container {  
    display: grid;  
    grid-template-columns: auto auto auto auto;  
}
```

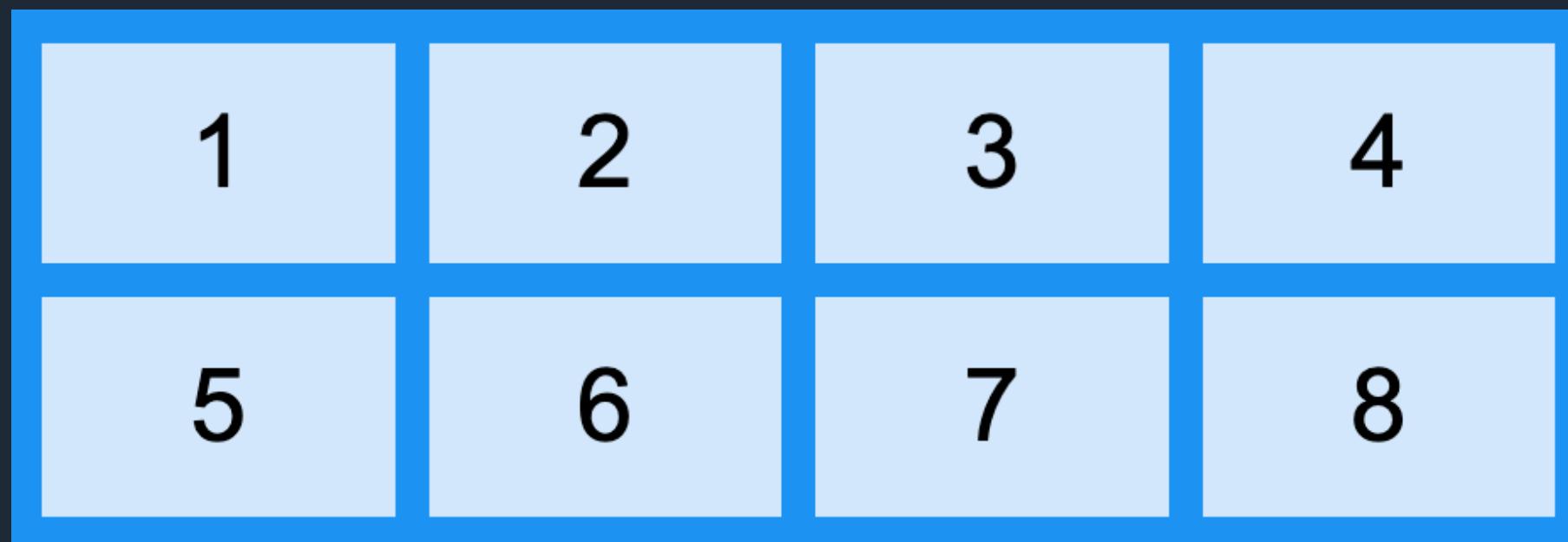
Grid Rows

Horisontal Lines



```
.grid-container {  
    display: grid;  
    grid-template-rows: 100px 300px;  
}
```

grid-template-columns



The grid-template-columns Property

The `grid-template-columns` property defines the number of columns in your grid layout, and it can define the width of each column.

Make a grid with 4 columns:

```
.grid-container {  
    display: grid;  
    grid-template-columns: auto auto auto auto;  
}
```

Set a size for the 4 columns:

```
.grid-container {  
    display: grid;  
    grid-template-columns: 80px 200px auto 40px;  
}
```

https://www.w3schools.com/css/css_grid.asp

grid-template-rows

The grid-template-rows Property

The `grid-template-rows` property defines the height of each row.

1	2	3	4
5	6	7	8

The value is a space-separated-list, where each value defines the height of the respective row:

```
.grid-container {  
    display: grid;  
    grid-template-rows: 80px 200px;  
}
```

https://www.w3schools.com/css/css_grid.asp

The justify-content Property

The `justify-content` property is used to align the whole grid inside the container.

1	2	3
4	5	6

Note: The grid's total width has to be less than the container's width for the `justify-content` property to have any effect.

Example

```
.grid-container {  
  display: grid;  
  justify-content: space-around;  
}
```

[Try it Yourself »](#)

The align-content Property

The `align-content` property is used to *vertically* align the whole grid inside the container.

1	2	3
4	5	6

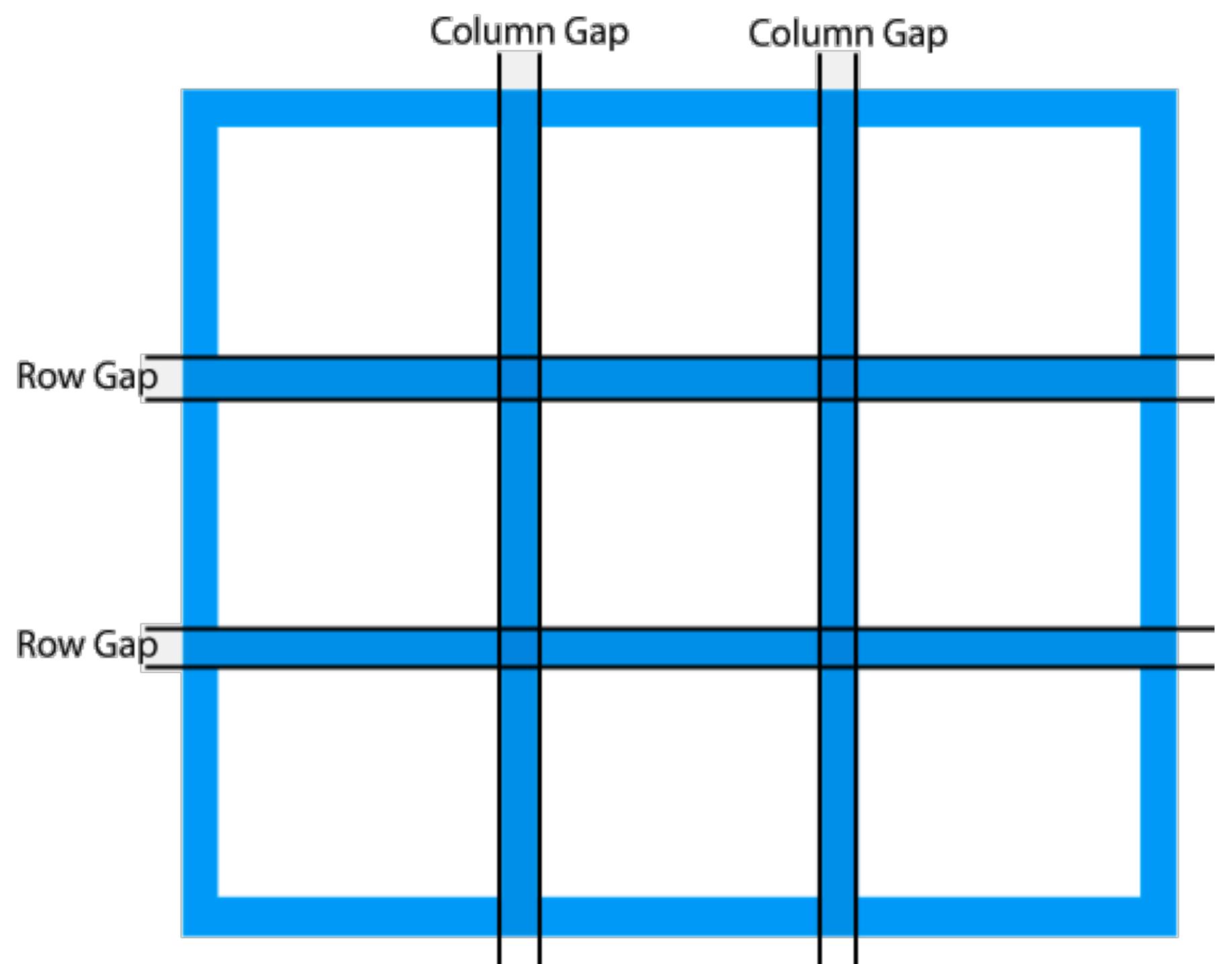
Note: The grid's total height has to be less than the container's height for the `align-content` property to have any effect.

Example

```
.grid-container {  
  display: grid;  
  height: 400px;  
  align-content: center;  
}
```

[Try it Yourself »](#)

Grid Gap



The `column-gap` property sets the gap between the columns:

```
.grid-container {  
    display: grid;  
    column-gap: 50px;  
}
```

The `row-gap` property sets the gap between the rows:

```
.grid-container {  
    display: grid;  
    row-gap: 50px;  
}
```

The `gap` property is a shorthand property for the `row-gap` and the `column-gap` properties:

```
.grid-container {  
    display: grid;  
    gap: 50px 100px;  
}
```

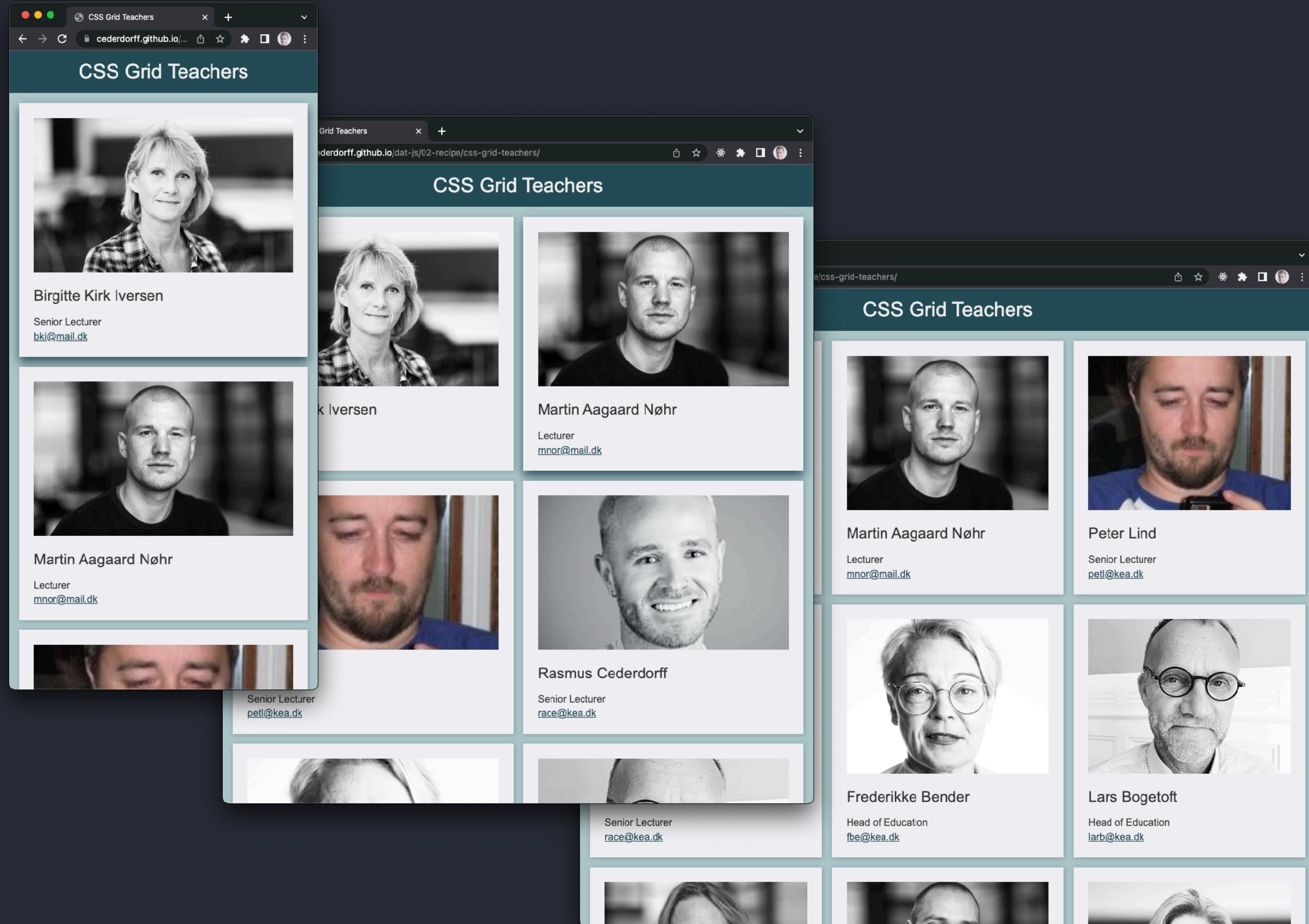
The `gap` property can also be used to set both the row gap and the column gap in one value:

```
.grid-container {  
    display: grid;  
    gap: 50px;  
}
```

100 seconds of CSS

G R O O D
L A Y O U T

CSS Grid & media queries



[css-grid-teachers](#)

```
/* ----- grid container styling ----- */
.grid-container {
  display: grid;
  grid-template-columns: 1fr;
  gap: 1em;
  padding: 1em;
}

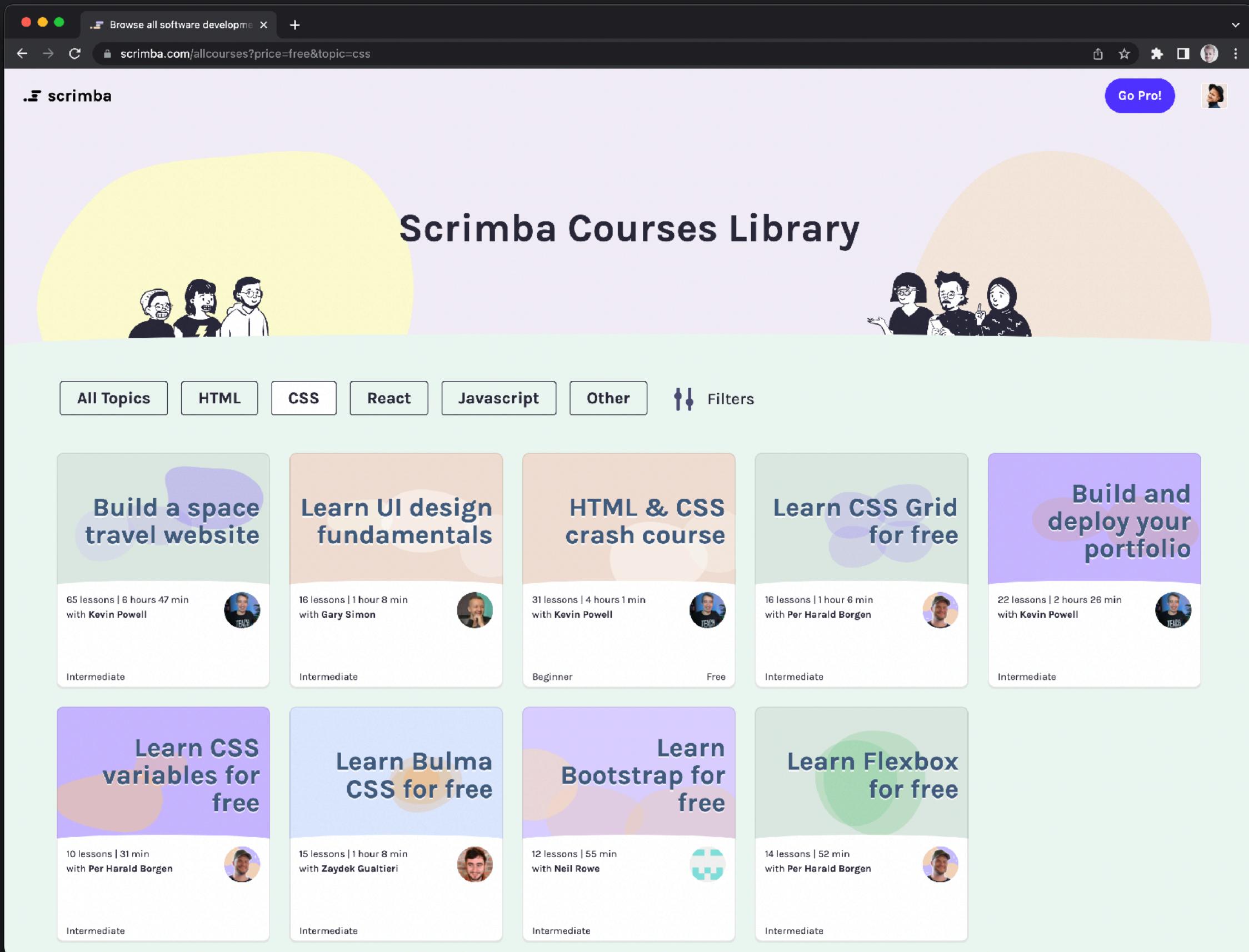
@media (min-width: 600px) {
  .grid-container {
    grid-template-columns: 1fr 1fr;
  }
}

@media (min-width: 992px) {
  .grid-container {
    grid-template-columns: 1fr 1fr 1fr;
  }
}
```

Scrimba Courses

Learn CSS Grid for free: <https://scrimba.com/learn/cssgrid>

Learn Flexbox for free: <https://scrimba.com/learn/flexbox>



<https://scrimba.com/allcourses>

CSS Grid Layout Module

The screenshot shows the main page of the CSS Grid Layout Module. At the top, there's a navigation bar with links for Tutorials, References, Exercises, Sign Up, and a Login button. Below the navigation is a secondary menu with links for HTML, CSS (which is highlighted in green), JAVASCRIPT, SQL, PYTHON, and JAVA. A search bar is also present. The main content area has a title "CSS Grid Layout Module". Below the title is a diagram of a grid layout with four columns and three rows. The columns are labeled "Header", "Main", "Right", and "Footer". The rows are labeled "Menu", "Main Content", and "Footer". A "Try it Yourself" button is located at the bottom left.

The screenshot shows the "Grid Container" page. The top navigation bar includes links for HTML, CSS (highlighted in green), JAVASCRIPT, SQL, PYTHON, and JAVA. The main content starts with a section titled "Grid Container" which explains that to make an element a grid container, you set the `display` property to `grid` or `inline-grid`. It mentions that grid containers consist of grid items placed in columns and rows. Below this is a section titled "The grid-template-columns Property" which describes how to define the number of columns and their widths. It includes an example code snippet for creating a grid with 4 columns:

```
.grid-container {  
  display: grid;  
  grid-template-columns: auto auto auto auto;  
}
```

The screenshot shows the "CSS Grid Item" page. The top navigation bar includes links for HTML, CSS (highlighted in green), JAVASCRIPT, SQL, PYTHON, and JAVA. The main content area has a title "CSS Grid Item". It features a large diagram of a grid layout with 5 items labeled 1 through 5. A "Try it Yourself" button is located at the bottom left. Below the diagram, there's a section titled "Child Elements (Items)" with the text: "A grid container contains grid items."

CSS Templates

w3schools.com/css/css_templates.asp

HTML CSS JAVASCRIPT SQL PYTHON PHP BOOTSTRAP HOW TO W3.CSS JAVA JQUERY C++

CSS Multiple Columns
CSS User Interface
CSS Variables
CSS Box Sizing
CSS Media Queries
CSS MQ Examples
CSS Flexbox

CSS Responsive
RWD Intro
RWD Viewport
RWD Grid View
RWD Media Queries
RWD Images
RWD Videos
RWD Frameworks
RWD Templates

CSS Grid
Grid Intro
Grid Container
Grid Item

CSS SASS
SASS Tutorial

CSS Examples
CSS Templates
CSS Examples
CSS Quiz
CSS Exercises
CSS Certificate

CSS References
CSS Reference
CSS Selectors

CSS Layout Templates

We have created some responsive starter templates with CSS.

You are free to modify, save, share, and use them in all your projects.

Header, equal columns and footer:

Try it (using float) »

Try it (using flexbox) »

Try it (using grid) »

Header, unequal columns and footer:

Try it (using float) »

Try it (using flexbox) »

Try it (using grid) »

Topnav, content and footer:

Sidenav and content:

NetSikker inkluderet
sikrer dig hjælp ved digitale krænkelser
Se vilkår på telenor.dk

NEW
We just launched W3Schools videos
Explore now

COLOR PICKER

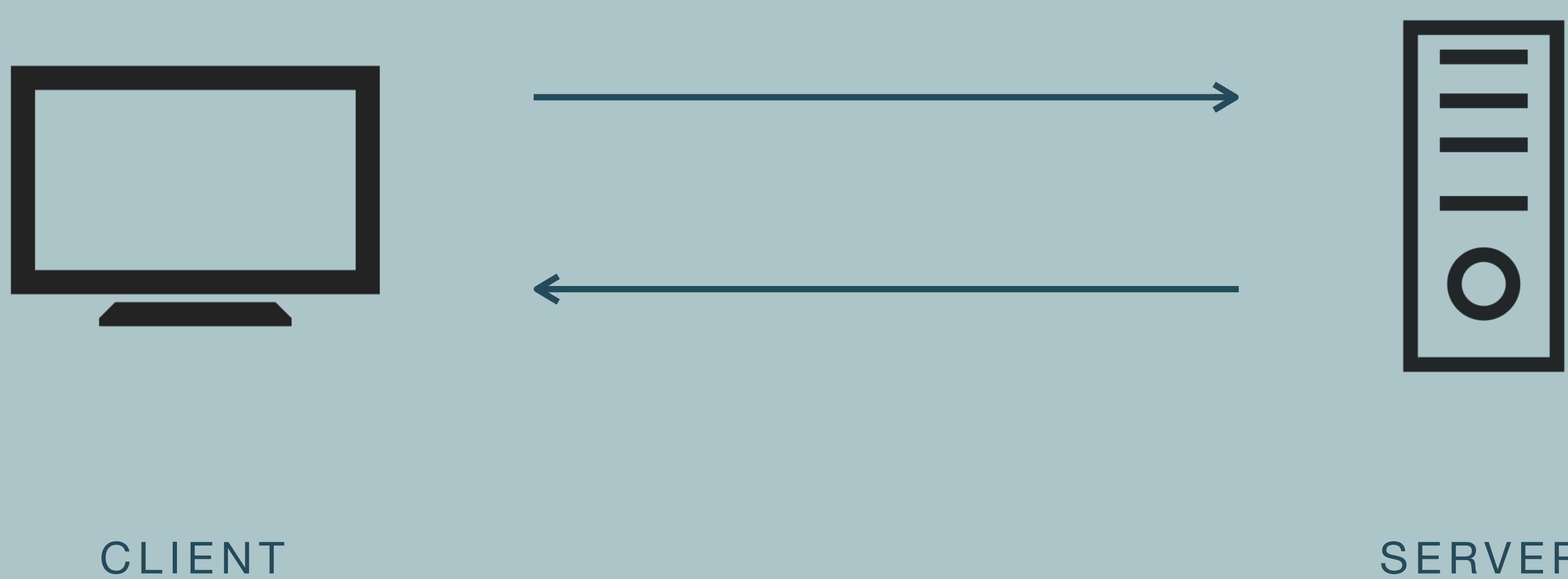
LIKE US

Fetch

Fetch is a built-in JavaScript API (functionality) for requesting data from a server.
Fetch = get data from a server.

Client-Server Model

Communication between web **clients** and web **servers**.



fetch(...)

HTTP Requests in
JavaScript.

A way to get and post data
from and to data sources.

```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/callback.js")
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data);
  });
// or with promises
const response = fetch("https://cederdorff.github.io/web-frontend/promise.js");
const data = response.json();
console.log(data);
```

fetch(...)

HTTP Requests in
JavaScript.

A way to get and post data
from and to data sources.

getCharacters fetches a list of characters
from a JSON data source, parses the JSON
to JS and returns the data.

```
async function getCharacters() {  
  const response = await fetch(url);  
  const data = await response.json();  
  console.log(data);  
  return data;  
}
```

fetch(...)

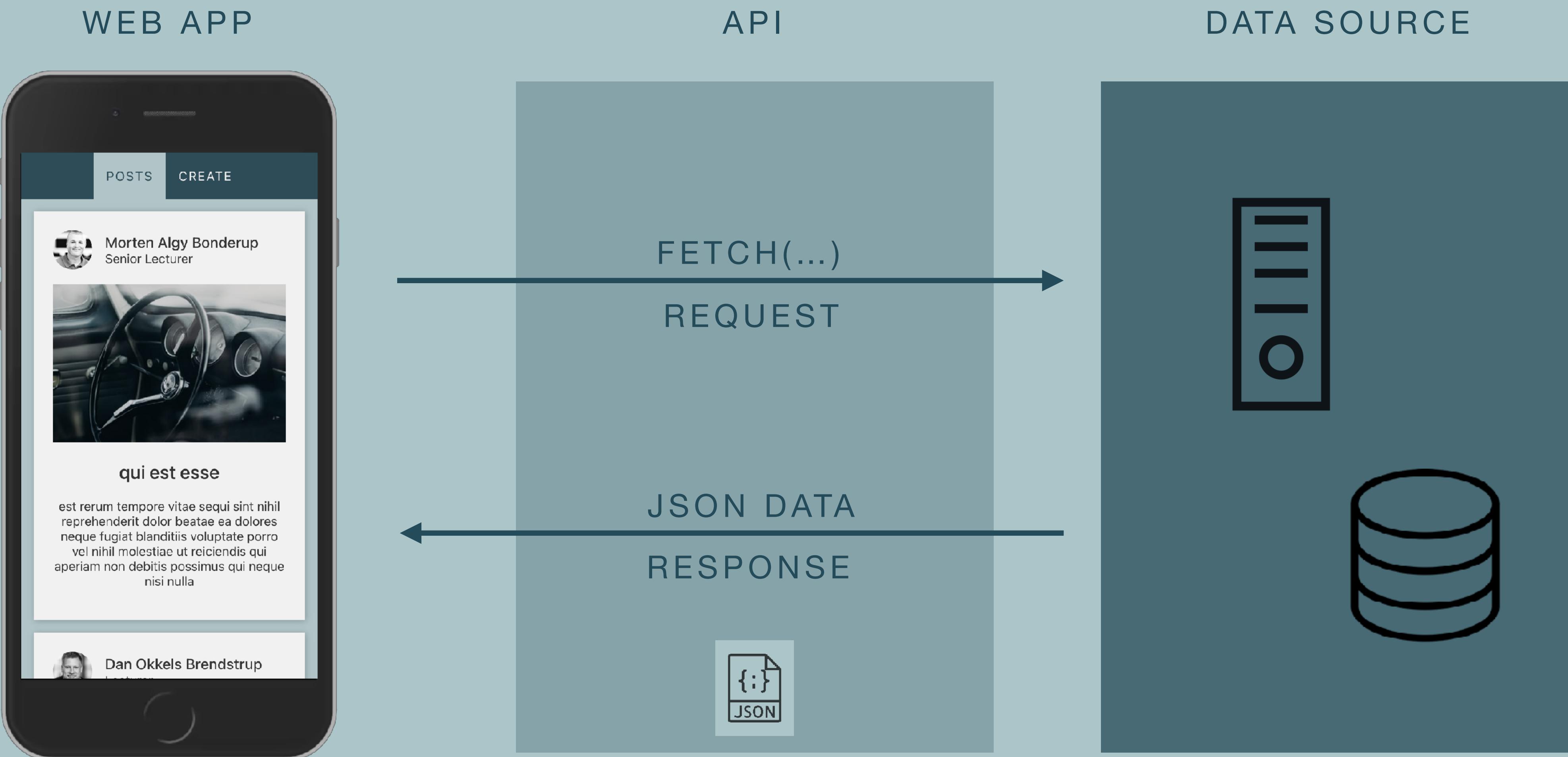
... get & post data from and to a data source
... can perform network requests to a server

```
1 let promise = fetch(url, [options])
```

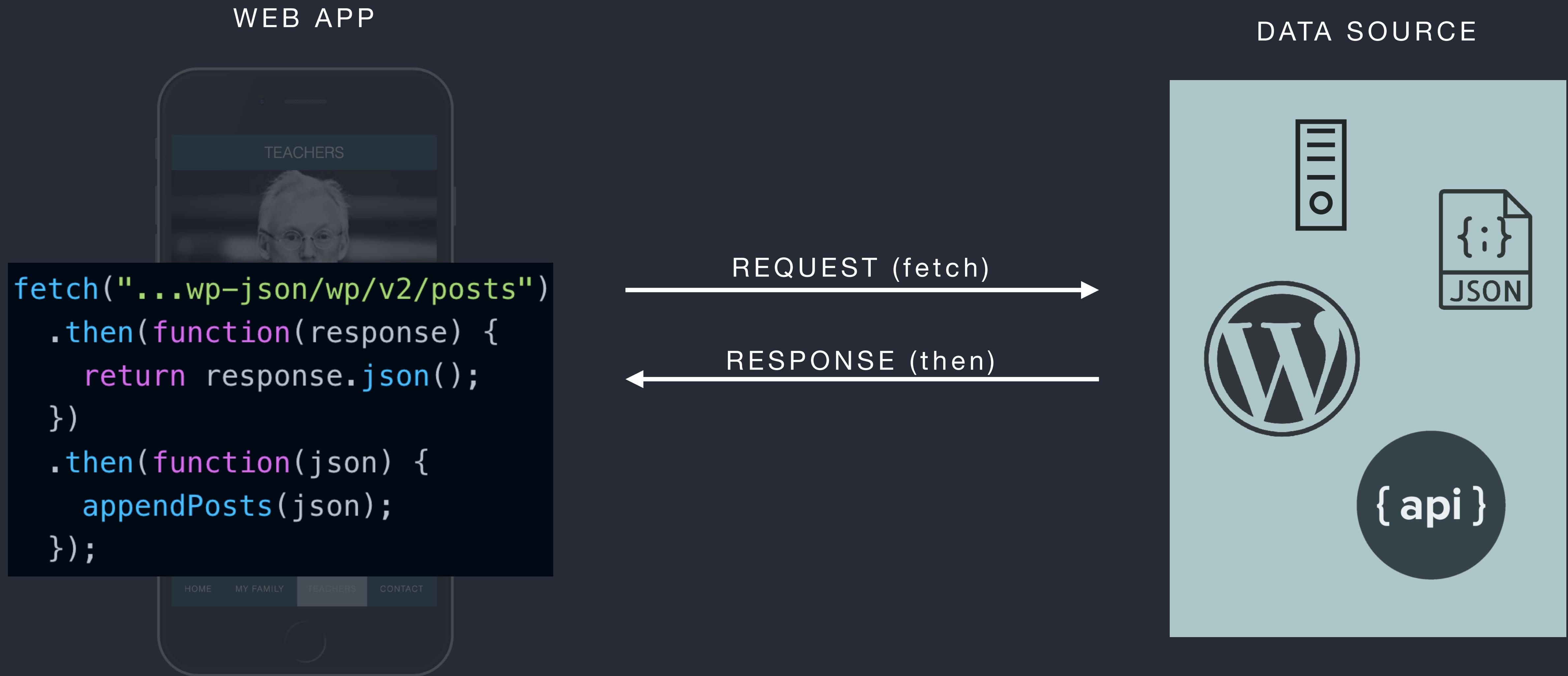
- `url` – the URL to access.
- `options` – optional parameters: method, headers etc.

Without `options`, this is a simple GET request, downloading the contents of the `url`.

Fetch, fetch, fetch



Fetch



```

/*
Fetches json data from the file persons.json
*/
fetch('json/persons.json')
  .then(function (response) {
    return response.json();
  })
  .then(function (jsonData) {
    console.log(jsonData);
    appendPersons(jsonData)
  });

/*
Appends json data to the DOM
*/
function appendPersons(persons) {
  let htmlTemplate = "";
  for (let person of persons) {
    htmlTemplate += /*html*/
      `


        <h4>${person.name}</h4>
        <p>${person.age} years old</p>
        <p>Hair color: ${person.hairColor}</p>
        <p>Relation: ${person.relation}</p>

`;
  }
  document.querySelector("#persons").innerHTML = htmlTemplate;
}

```

```

[
  {
    "name": "Peter Madsen",
    "age": 52,
    "hairColor": "blonde",
    "relation": "dad",
    "img": "img/dad.jpg"
  },
  {
    "name": "Ane Madsen",
    "age": 51,
    "hairColor": "brown",
    "relation": "mom",
    "img": "img/ane.jpg"
  },
  {
    "name": "Rasmus Madsen",
    "age": 28,
    "hairColor": "blonde",
    "relation": "brother",
    "img": "img/IMG_0526_kvadrat.jpg"
  },
  {
    "name": "Mie Madsen",
    "age": 25,
    "hairColor": "brown",
    "relation": "blonde",
    "img": "img/mie.jpg"
  },
  {
    "name": "Mads Madsen",
    "age": 18,
    "hairColor": "dark",
    "relation": "blonde",
    "img": "img/mads.jpg"
  },
  {
    "name": "Jens Madsen",
    "age": 14,
    "hairColor": "blonde",
    "relation": "uncle",
    "img": "img/jenspeter.jpg"
  }
]

```

```

/*
Fetches json data from the file persons.json
*/
fetch('json/persons.json')
  .then(function (response) {
    return response.json();
  })
  .then(function (jsonData) {
    console.log(jsonData);
    appendPersons(jsonData);
  });

/*
Appends json data to the DOM
*/
function appendPersons(persons) {
  let htmlTemplate = "";
  for (let person of persons) {
    htmlTemplate += /*html*/
      <article>
        
        <h4>${person.name}</h4>
        <p>${person.age} years old</p>
        <p>Hair color: ${person.hairColor}</p>
        <p>Relation: ${person.relation}</p>
      </article>
  }
  document.querySelector("#persons").innerHTML = htmlTemplate;
}

```

REQUEST (fetch)

RESPONSE (then)

```
[
  {
    "name": "Peter Madsen",
    "age": 52,
    "hairColor": "blonde",
    "relation": "dad",
    "img": "img/dad.jpg"
  },
  {
    "name": "Ane Madsen",
    "age": 51,
    "hairColor": "brown",
    "relation": "mom",
    "img": "img/ane.jpg"
  },
  {
    "name": "Rasmus Madsen",
    "age": 28,
    "hairColor": "blonde",
    "relation": "brother",
    "img": "img/IMG_0526_kvadrat.jpg"
  },
  {
    "name": "Mie Madsen",
    "age": 25,
    "hairColor": "brown",
    "relation": "blonde",
    "img": "img/mie.jpg"
  },
  {
    "name": "Mads Madsen",
    "age": 18,
    "hairColor": "dark",
    "relation": "blonde",
    "img": "img/mads.jpg"
  },
  {
    "name": "Jens Madsen",
    "age": 14,
    "hairColor": "blonde",
    "relation": "uncle",
    "img": "img/jenspeter.jpg"
  }
]
```

```
/*
Fetches json data from the file persons.json
*/
fetch('json/persons.json')
  .then(function (response) {
    return response.json();
  })
  .then(function (jsonData) {
    console.log(jsonData);
    appendPersons(jsonData)
  });

/*
Appends json data to the DOM
*/
function appendPersons(persons) {
  let htmlTemplate = "";
  for (let person of persons) {
    htmlTemplate += /*html*/
      `


        <h4>${person.name}</h4>
        <p>${person.age} years old</p>
        <p>Hair color: ${person.hairColor}</p>
        <p>Relation: ${person.relation}</p>

`;
  }
  document.querySelector("#persons").innerHTML = htmlTemplate;
}
```

```
},
{
  "name": "Rasmus Madsen",
  "age": 28,
  "hairColor": "blonde",
  "relation": "brother",
  "img": "img/IMG_0526_kvadrat.jpg"
},
{
  "name": "Mie Madsen",
  "age": 25,
  "hairColor": "brown",
  "relation": "blonde",
  "img": "img/mie.jpg"
},
{
  "name": "Mads Madsen",
  "age": 18,
  "hairColor": "dark",
  "relation": "blonde",
  "img": "img/mads.jpg"
},
{
```

request (fetch)

The diagram illustrates the flow of data from the `app.js` code to the resulting JSON response. A curved arrow originates from the URL in the `fetch` statement in the `getPerson` function and points to the JSON data on the right. Another curved arrow originates from the `persons` variable in the `displayPersons` function and points back to the `app.js` code.

```
app.js — web-diplom-frontend
JS app.js ×
fetch-persons-grid > JS app.js > ...
1  let persons = [] // global variable
2
3  async function getPerson() {
4      const response = await fetch(
5          "https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json"
6      ); // fetch request - fetch data from a given url
7      persons = await response.json(); // setting global variable with fetched data
8      displayPersons(persons); // calling displayPersons with persons as parameter
9  }
10
11 function displayPersons(listOfPersons) {
12     let html = ""; // variable to store html
13     //loop through all persons and create an article with content for each
14     for (const person of listOfPersons) {
15         html += /*html*/
16             `

17                 
18                 <h2>${person.name}</h2>
19                 <p>${person.title}</p>
20                 <a href="mailto:${person.mail}">${person.mail}</a>
21             </article>
22         `; // generate and save html for every person in html variable
23     }
24     // set grid container content with person <article> elements
25     // saved in html
26     document.querySelector("#content").innerHTML = html;
27 }
28
29 getPerson(); // execute get persons to start the fun
30


```

Ln 1, Col 1 Spaces: 4 UTF-8 LF {} JavaScript Go Live ✓ Prettier

A screenshot of a browser window showing the JSON data received from the URL `https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json`. The JSON array contains eight objects, each representing a person with properties: name, mail, title, and img.

```
[{"name": "Birgitte Kirk Iversen", "mail": "bkj@mail.dk", "title": "Senior Lecturer", "img": "https://www.eaaa.dk/media/u4gorzsd/birgitte-kirk-iversen2.jpg?width=800&height=450"}, {"name": "Martin Aagaard N\u00f8hr", "mail": "mnor@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/oayjq02h/martin-n%C3%B8hr.jpg?width=800&height=450"}, {"name": "Rasmus Cederdorff", "mail": "race@mail.dk", "title": "Senior Lecturer", "img": "https://www.eaaa.dk/media/devlvgj/rasmus-cederdorff.jpg?width=800&height=450"}, {"name": "Dan Okkels Brendstrup", "mail": "dob@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstrup.jpg?width=800&height=450"}, {"name": "Line Skj\u00f8dt", "mail": "lskj@mail.dk", "title": "Senior Lecturer & Internship Coordinator", "img": "https://www.eaaa.dk/media/14qpfeq4/line-skj%C3%B8dt.jpg?width=800&height=450"}, {"name": "Kasper Fischer Topp", "mail": "kato@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/lxzcybme/kasper-topp.jpg?width=800&height=450"}, {"name": "Anne Kirketerp", "mail": "anki@mail.dk", "title": "", "img": ""}]
```

fetch-persons-grid

request (fetch)

The diagram illustrates the flow of data from the `app.js` code to the final JSON response. A curved arrow labeled "request (fetch)" points from the `fetch` call in the `getPerson` function to the browser window. Another curved arrow labeled "response (JSON)" points from the `response.json` call to the JSON data displayed in the browser.

```
app.js — web-diplom-frontend
JS app.js ×
fetch-persons-grid > JS app.js > ...
1 let persons = [] // global variable
2
3 async function getPerson() {
4     const response = await fetch(
5         "https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json"
6     ); // fetch request - fetch data from a given url
7     persons = await response.json(); // setting global variable with fetched data
8     displayPersons(persons); // calling displayPersons with persons as parameter
9 }
10
11 function displayPersons(listOfPersons) {
12     let html = ""; // variable to store html
13     //loop through all persons and create an article with content for each
14     for (const person of listOfPersons) {
15         html += /*html*/
16             `

17                 
18                 <h2>${person.name}</h2>
19                 <p>${person.title}</p>
20                 <a href="mailto:${person.mail}">${person.mail}</a>
21             </article>
22         `; // generate and save html for every person in html variable
23     }
24     // set grid container content with person <article> elements
25     // saved in html
26     document.querySelector("#content").innerHTML = html;
27 }
28
29 getPerson(); // execute get persons to start the fun
30


```

Ln 1, Col 1 Spaces: 4 UTF-8 LF {} JavaScript Go Live ✓ Prettier

A screenshot of a browser window showing the JSON response from the fetch request. The URL is `https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json`. The response is a JSON array containing eight objects, each representing a person with properties: name, mail, title, and img.

```
[{"name": "Birgitte Kirk Iversen", "mail": "bkj@mail.dk", "title": "Senior Lecturer", "img": "https://www.eaaa.dk/media/u4gorzsd/birgitte-kirk-iversen2.jpg?width=800&height=450"}, {"name": "Martin Aagaard N\u00f8hr", "mail": "mnor@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/oayjq02h/martin-n%C3%B8hr.jpg?width=800&height=450"}, {"name": "Rasmus Cederdorff", "mail": "race@mail.dk", "title": "Senior Lecturer", "img": "https://www.eaaa.dk/media/devlvgj/rasmus-cederdorff.jpg?width=800&height=450"}, {"name": "Dan Okkels Brendstrup", "mail": "dob@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstrup.jpg?width=800&height=450"}, {"name": "Line Skj\u00f8dt", "mail": "lskj@mail.dk", "title": "Senior Lecturer & Internship Coordinator", "img": "https://www.eaaa.dk/media/14qpfeq4/line-skj%C3%B8dt.jpg?width=800&height=450"}, {"name": "Kasper Fischer Topp", "mail": "kato@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/lxzcybme/kasper-topp.jpg?width=800&height=450"}, {"name": "Anne Kirketerp", "mail": "anki@mail.dk", "title": "", "img": ""}]
```

The diagram illustrates the data flow in a web application. On the left, a screenshot of a code editor shows the file `app.js` with the following code:

```
1 let persons = [] // global variable
2
3 async function getPerson() {
4     const response = await fetch(
5         "https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json"
6     ); // fetch request - fetch data from a given url
7     persons = await response.json(); // setting global variable with fetched data
8     displayPersons(persons); // calling displayPersons with persons as parameter
9 }
10
11 function displayPersons(listOfPersons) {
12     let html = ""; // variable to store html
13     //loop through all persons and create an article with content for each
14     for (const person of listOfPersons) {
15         html += /*html*/
16             `

17                 
18                 <h2>${person.name}</h2>
19                 <p>${person.title}</p>
20                 <a href="mailto:${person.mail}">${person.mail}</a>
21             </article>
22         `; // generate and save html for every person in html variable
23     }
24     // set grid container content with person <article> elements
25     // saved in html
26     document.querySelector("#content").innerHTML = html;
27 }
28
29 getPerson(); // execute get persons to start the fun
30


```

On the right, a screenshot of a browser window shows the JSON data source at `https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json`. The JSON array contains the following objects:

```
[{"name": "Birgitte Kirk Iversen", "mail": "bki@mail.dk", "title": "Senior Lecturer", "img": "https://www.eaaa.dk/media/u4gorzsd/birgitte-kirk-iversen2.jpg?width=800&height=450"}, {"name": "Martin Aagaard N\u00f8hr", "mail": "mnor@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/oayjq02h/martin-n%C3%B8hr.jpg?width=800&height=450"}, {"name": "Rasmus Cederdorff", "mail": "race@mail.dk", "title": "Senior Lecturer", "img": "https://www.eaaa.dk/media/devlvgj/rasmus-cederdorff.jpg?width=800&height=450"}, {"name": "Dan Okkels Brendstrup", "mail": "dob@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstrup.jpg?width=800&height=450"}, {"name": "Line Skj\u00f8dt", "mail": "lskj@mail.dk", "title": "Senior Lecturer & Internship Coordinator", "img": "https://www.eaaa.dk/media/14qpfq4/line-skj%C3%B8dt.jpg?width=800&height=450"}, {"name": "Kasper Fischer Topp", "mail": "kato@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/lxzcybme/kasper-topp.jpg?width=800&height=450"}, {"name": "Anne Kirketerp", "mail": "anki@mail.dk", "title": "", "img": ""}]
```

fetch-persons-grid

index.html — web-diplom-frontend

JS app.js

```
fetch-persons-grid > JS app.js > ...
1 let persons = [] // global variable
2
3 async function getPerson() {
4     const response = await fetch(
5         "https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json"
6     ); // fetch request - fetch data from a given url
7     persons = await response.json(); // setting global variable with fetched data
8     displayPersons(persons); // calling displayPersons with persons as parameter
9 }
10
11 function displayPersons(listOfPersons) {
12     let html = ""; // variable to store html
13     //loop through all persons and create an article with content for each
14     for (const person of listOfPersons) {
15         html += /*html*/ `
16             <article>
17                 
18                 <h2>${person.name}</h2>
19                 <p>${person.title}</p>
20                 <a href="mailto:${person.mail}">${person.mail}</a>
21             </article>
22         `; // generate and save html for every person in html variable
23     }
24     // set grid container content with person <article> elements
25     // saved in html
26     document.querySelector("#content").innerHTML = html;
27 }
28
29 getPerson(); // execute get persons to start the fun
30
```

index.html

```
fetch-persons-grid > index.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width, initial-scale=1.0">
8     <link rel="stylesheet" href="app.css">
9     <title>Fetch Persons</title>
10
11 </head>
12 <body>
13     <header>
14         <h1>Fetch Persons</h1>
15     </header>
16     <main>
17         <section id="content" class="grid-container"></section>
18     </main>
19     <script src="app.js"></script>
20
21 </body>
22 </html>
```

DOM Manipulation

Ln 17, Col 9 Spaces: 4 UTF-8 LF HTML ⚡ Go Live ✅ Prettier

fetch-persons-grid

Fetch

... get & post data from and to a data source

```
// Simple javascript 😞  
  
//Synchronous fetch using async/await.  
  
// Usual way  
✓ const jsonData = fetch('URL')  
    .then(response => response.json())  
    .then(json => console.log(json));  
  
// Using await  
✓ const jsonData = await fetch('URL').then(res => res.json())  
  
// Shorter syntax 😊  
✓ const jsonData = await (await fetch('URL')).json();
```

<https://www.instagram.com/p/B0nxQjXj9Zi/>

Async JS

JavaScript reads and runs the script from top to bottom.

JavaScript functions are executed in the sequence they are called. Not in the sequence they are defined.

... by default JavaScript is synchronous.

JS is Synchronous & Single-Threaded

```
function myFirst() {  
  console.log("Hello");  
}
```

```
function mySecond() {  
  console.log("Goodbye");  
}
```

```
mySecond();  
myFirst();
```

```
/* ----- Global Variables ----- */
let _users = [];
let _selectedUserId;

/* ----- */

async function fetchUsers() { ... }
function appendUsers(usersArray) { ... }
}

// ===== INIT APP =====

async function initApp() {
    await fetchUsers();
    appendUsers(_users);
}

initApp();
```

With callbacks, we can make JS Asynchronous

```
setTimeout(() => {
  console.log("Hey, I'm async!");
}, 3000);

btn.addEventListener('click', () => {
  alert("Hey, you clicked me!");
});
```

```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas")
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data);
  });
});
```

Fetch is Asynchronous

And it's about making HTTP requests in JavaScript.
... and a way to get & post data from and to a data source.

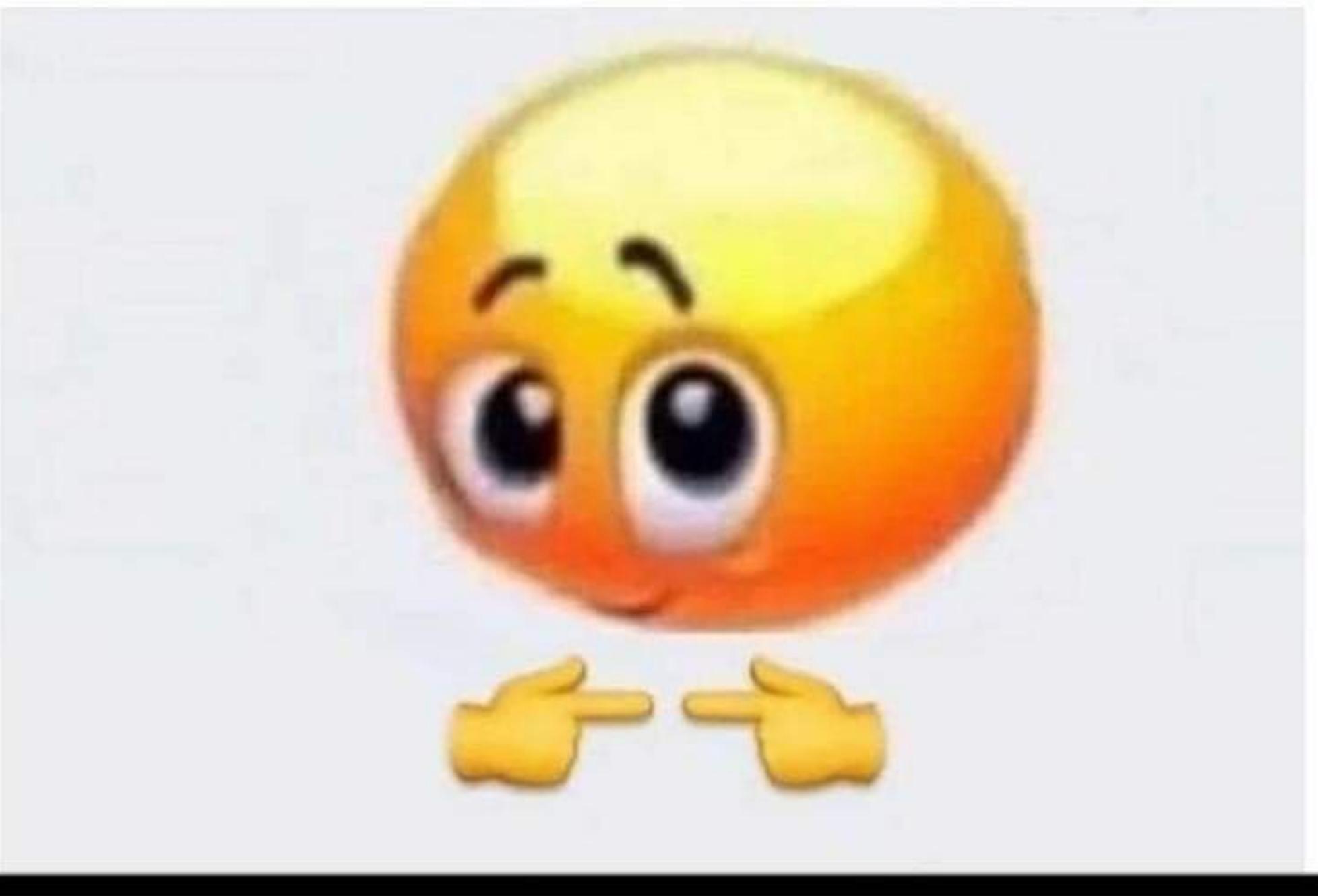
```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json")
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data);
  });

```

Fetch: callback (then) vs async/await

```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json")
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data);
  });
// 
// 
// or with async/await
const response = await fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json");
const data = await response.json();
console.log(data);
```

will you be async to my await?



async & await

- Use await to tell JS to wait for a fetch call to finish and to wait for JSON to parse.
- When using await you must tell JS that inside of the function goes some asynchronous code by wrapping it in an async function.

```
async function getPosts() {  
  const url = "https://raw.githubusercontent.com/.../data.json";  
  const response = await fetch(url);  
  const data = await response.json();  
  setPosts(data);  
}  
  
getPosts();
```

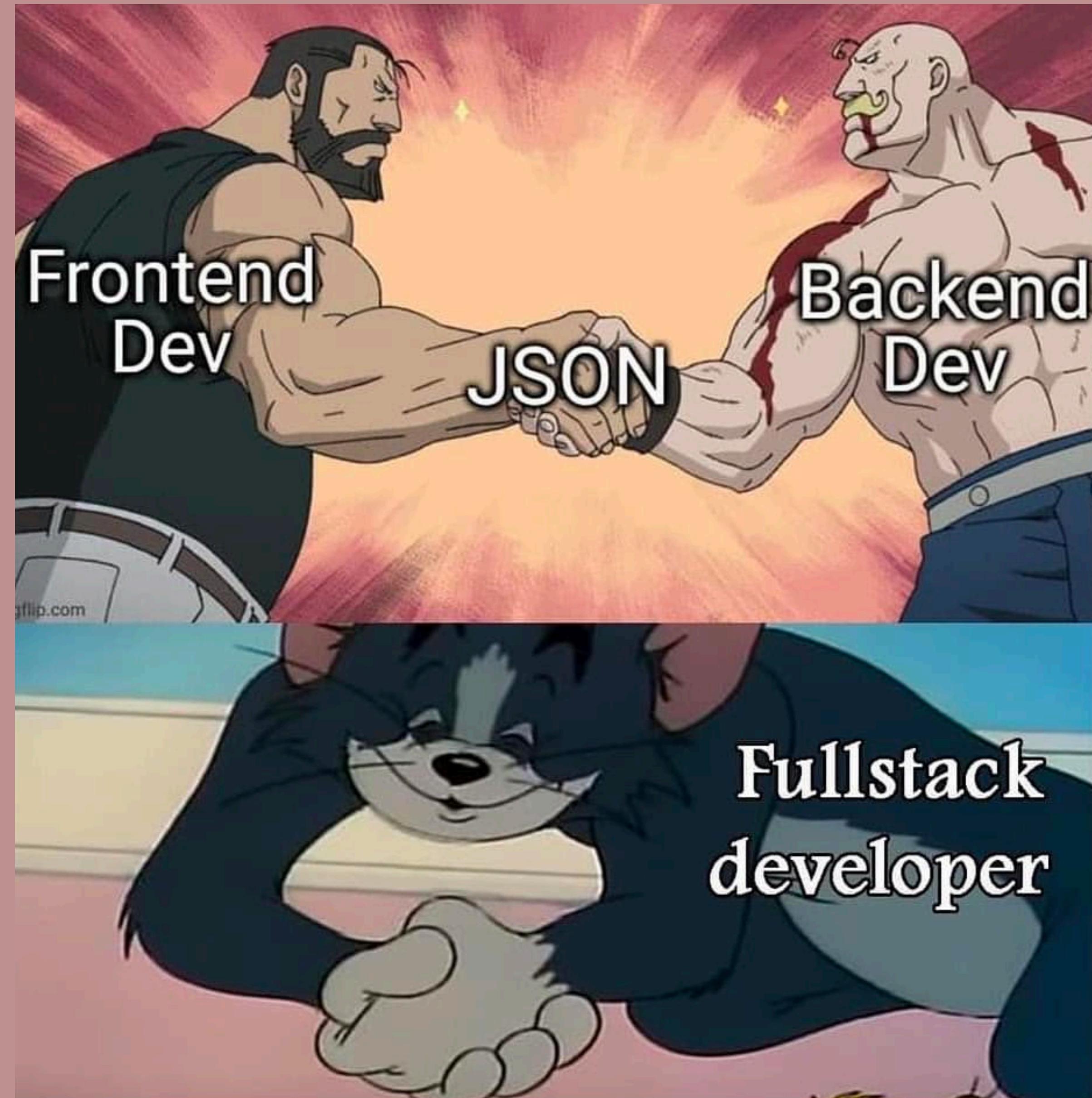
JSON

JavaScript Object Notation

JSON

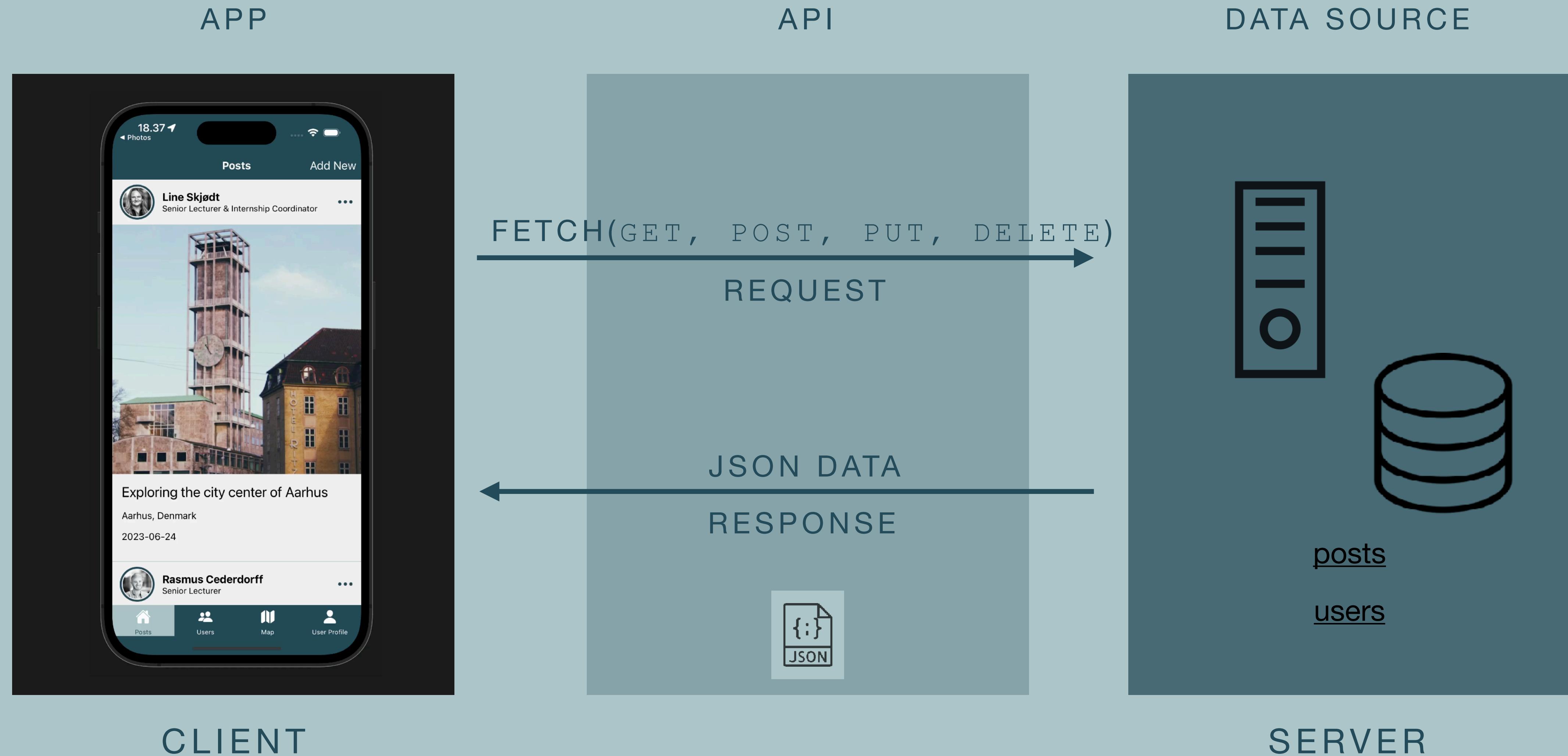
JavaScript Object Notation

... a syntax for storing & exchanging data
over the web



<https://www.instagram.com/p/CVqbCzgsZUF/>

JSON (& API) is the glue



JSON

... a syntax for storing and exchanging data over the web

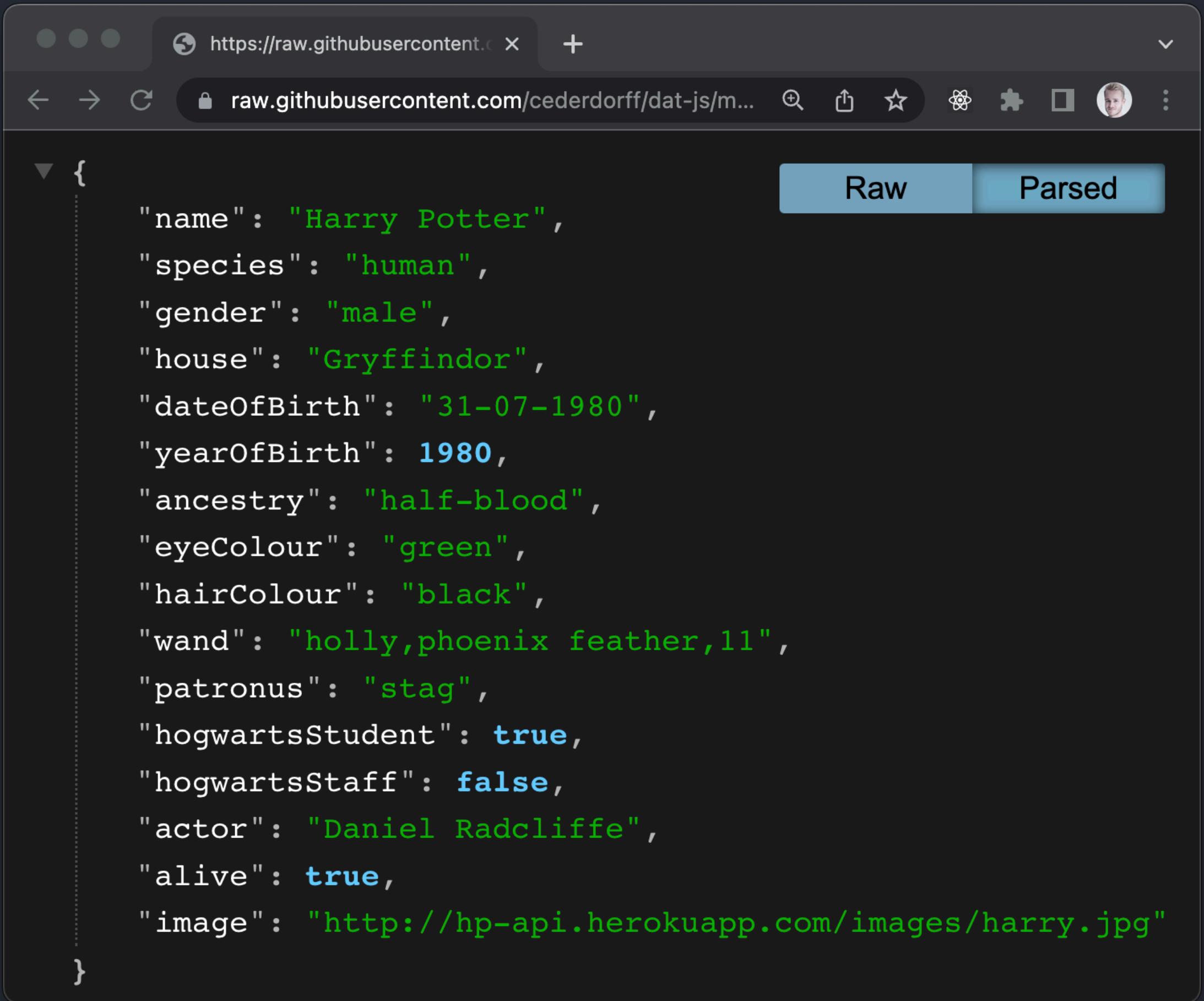
```
{  
  "name": "Alicia",  
  "age": 6  
}
```

JSON OBJECT

```
[{  
  "name": "Alicia",  
  "age": 6  
, {  
  "name": "Peter",  
  "age": 22  
}]
```

LIST OF JSON OBJECTS

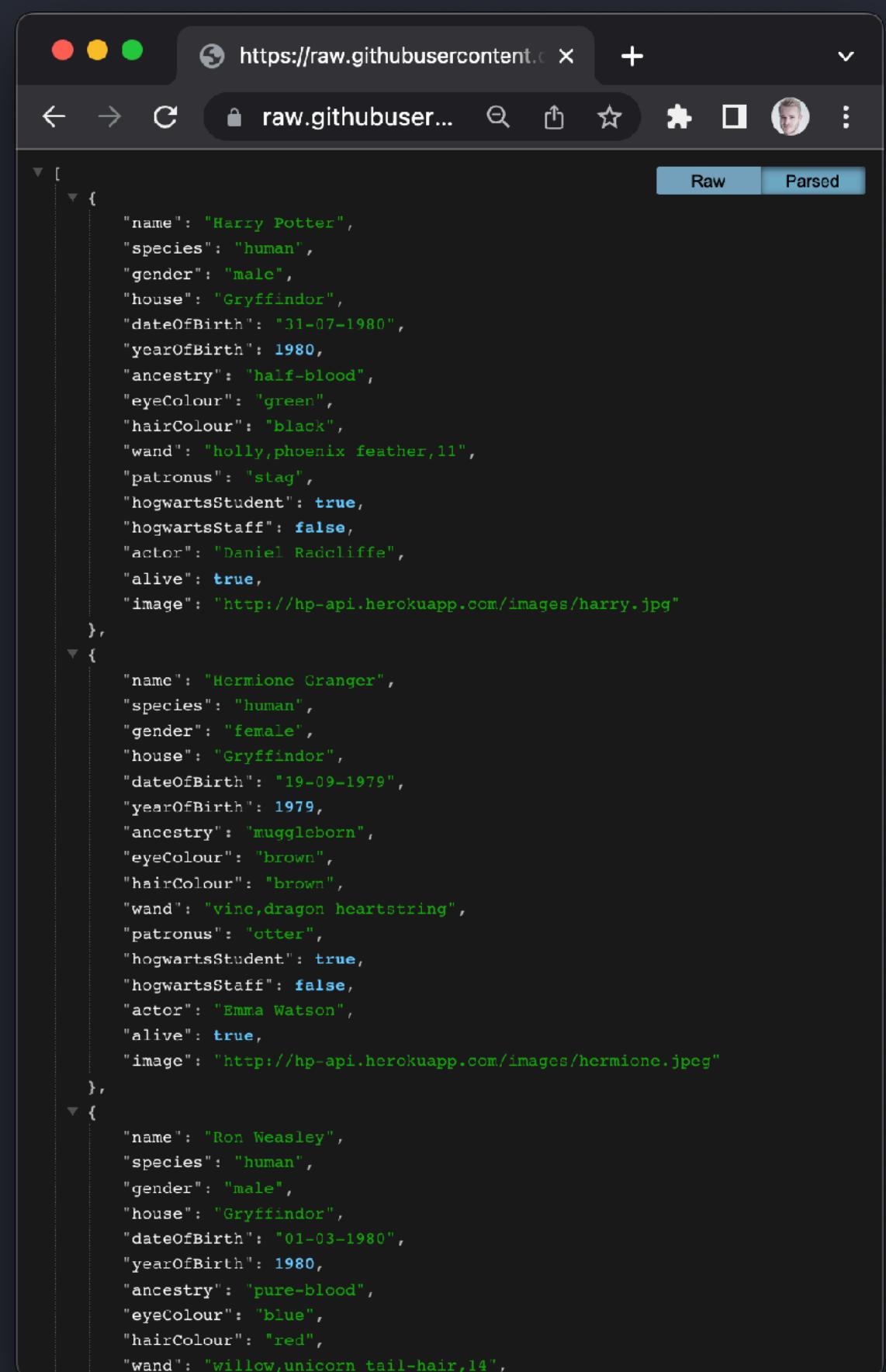
JSON



A screenshot of a web browser window displaying a single JSON object. The URL is <https://raw.githubusercontent.com/cederdorff/dat-js/master/data/harry.json>. The JSON object represents Harry Potter's character details. The 'Raw' tab is selected, showing the raw JSON code, while the 'Parsed' tab shows the same data as an expandable tree structure.

```
{  
  "name": "Harry Potter",  
  "species": "human",  
  "gender": "male",  
  "house": "Gryffindor",  
  "dateOfBirth": "31-07-1980",  
  "yearOfBirth": 1980,  
  "ancestry": "half-blood",  
  "eyeColour": "green",  
  "hairColour": "black",  
  "wand": "holly,phoenix feather,11",  
  "patronus": "stag",  
  "hogwartsStudent": true,  
  "hogwartsStaff": false,  
  "actor": "Daniel Radcliffe",  
  "alive": true,  
  "image": "http://hp-api.herokuapp.com/images/harry.jpg"  
}
```

JSON OBJECT



A screenshot of a web browser window displaying a list of three JSON objects. The URL is <https://raw.githubusercontent.com/cederdorff/dat-js/master/data/characters.json>. The JSON array contains three objects: Harry Potter, Hermione Granger, and Ron Weasley. The 'Raw' tab is selected, showing the raw JSON code, while the 'Parsed' tab shows the data as an expandable tree structure.

```
[  
  {  
    "name": "Harry Potter",  
    "species": "human",  
    "gender": "male",  

```

LIST OF JSON OBJECTS

JSON Object

The diagram illustrates the connection between a JSON object and a web application displaying Harry Potter characters.

Left Side: Web Application

A screenshot of a web browser window titled "Harry Potter Characters". The page displays three cards representing characters:

- Harry Potter**: Shows a portrait of Harry Potter and is labeled "Gryffindor".
- Hermione Granger**: Shows a portrait of Hermione Granger and is labeled "Gryffindor".
- Ron Weasley**: Shows a portrait of Ron Weasley and is labeled "Gryffindor".

Right Side: JSON Object

A screenshot of a web browser window showing the JSON representation of the Harry Potter character data. The JSON is displayed in a collapsible tree view with the "Parsed" tab selected. The data for Harry Potter is as follows:

```
{  
  "name": "Harry Potter",  
  "species": "human",  
  "gender": "male",  
  "house": "Gryffindor",  
  "dateOfBirth": "31-07-1980",  
  "yearOfBirth": 1980,  
  "ancestry": "half-blood",  
  "eyeColour": "green",  
  "hairColour": "black",  
  "wand": "holly,phoenix feather,11",  
  "patronus": "stag",  
  "hogwartsStudent": true,  
  "hogwartsStaff": false,  
  "actor": "Daniel Radcliffe",  
  "alive": true,  
  "image": "http://hp-api.herokuapp.com/images/harry.jpg"  
}
```

Red arrows point from the character cards in the application to their corresponding entries in the JSON object, indicating the data mapping.

<https://raw.githubusercontent.com/cederdorff/dat-js/main/data/harry.json>

JSON Object

The diagram illustrates the flow of data between a JavaScript application and a browser's developer tools.

JavaScript Application (Left): A screenshot of a code editor showing a file named `app.js`. The code defines a function `showCharacter` that inserts an HTML article element into a container with the ID `#characters`. The article contains an image and two pieces of text: the character's name and house. The variable `character` is passed to this function.

```
28
29  function showCharacter(character) {
30      document.querySelector("#characters").insertAdjacentHTML(
31          "beforeend",
32          /*html*/
33          `

34              
35              <h2>${character.name}</h2>
36              <p>${character.house}</p>
37          </article>
38      `);
39  }
40 }


```

Browser Console (Right): A screenshot of a browser window displaying a JSON object. The object represents a character named Harry Potter, detailing his species, gender, house, birth dates, ancestry, eye and hair colors, wand, patronus, and current status. An arrow points from the `character` variable in the `showCharacter` function to this JSON object, indicating it is the data being passed.

```
{
  "name": "Harry Potter",
  "species": "human",
  "gender": "male",
  "house": "Gryffindor",
  "dateOfBirth": "31-07-1980",
  "yearOfBirth": 1980,
  "ancestry": "half-blood",
  "eyeColour": "green",
  "hairColour": "black",
  "wand": "holly,phoenix feather,11",
  "patronus": "stag",
  "hogwartsStudent": true,
  "hogwartsStaff": false,
  "actor": "Daniel Radcliffe",
  "alive": true,
  "image": "http://hp-api.herokuapp.com/images/harry.jpg"
}
```

<https://raw.githubusercontent.com/cederdorff/dat-js/main/data/harry.json>

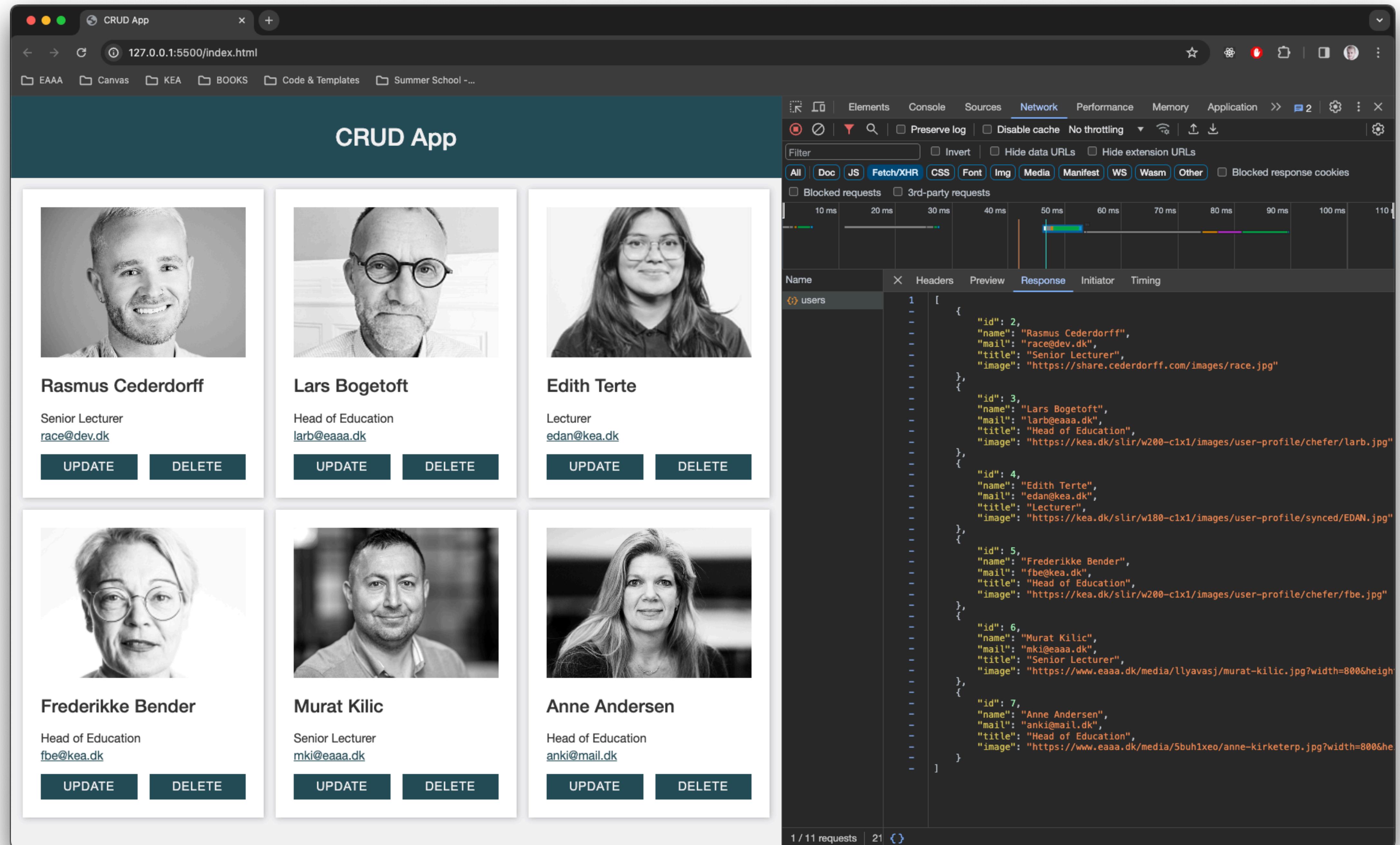
JAVASCRIPT OBJECT NOTATION

- Collection of key-value pair: “key” : “value”
- List of values, collections or objects
- Lightweight data-interchange format
- Syntax / text format for storing and exchanging data over the web
- Human and machine readable **text**: small, fast and simple
- Language independent
- Can be parsed directly to JavaScript Object
- JavaScript Objects can be converted directly to JSON
- The glue between programs (interface between frontend and backend)

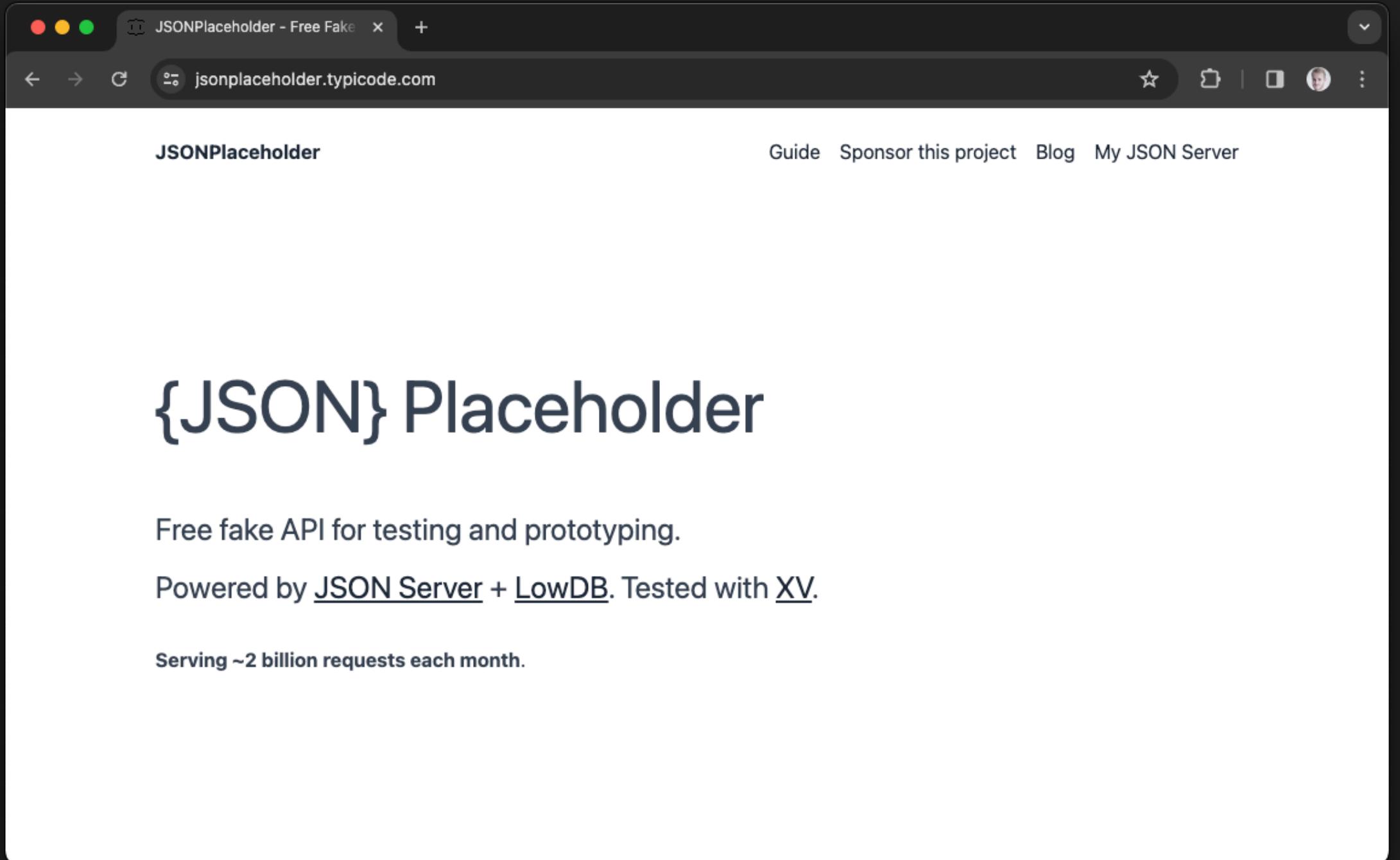
```
[  
 {  
   "id": "1",  
   "firstname": "Kasper",  
   "lastname": "Topp",  
   "age": "34",  
   "haircolor": "Dark Blonde",  
   "countryName": "Denmark",  
   "gender": "Male",  
   "lookingFor": "Female"  
 },  
 {  
   "id": "2",  
   "firstname": "Nicklas",  
   "lastname": "Andersen",  
   "age": "22",  
   "haircolor": "Brown",  
   "countryName": "Denmark",  
   "gender": "Male",  
   "lookingFor": "Female"  
 },  
 {  
   "id": "3",  
   "firstname": " Sarah",  
   "lastname": "Dybvad ",  
   "age": "34",  
   "haircolor": "Blonde",  
   "countryName": "Denmark",  
   "gender": "Female",  
   "lookingFor": "Male"  
 },  
 {  
   "id": "4",  
   "firstname": "Alex",  
   "lastname": "Hansen",  
   "age": "21",  
   "haircolor": "Blonde",  
   "countryName": "Denmark",  
   "gender": "Male",  
   "lookingFor": "Female"  
 }]
```

JSON METHODS

```
const user = {  
    name: "John",  
    age: 30,  
    gender: "male",  
    lookingFor: "female"  
};  
  
// === JSON.stringify === //  
const jsonUser = JSON.stringify(user);  
console.log(jsonUser); // {"name":"John","age":30,"gender":"male","lookingFor":"female"}  
  
// === JSON.parse === //  
const jsonString = '{"name":"John","age":30,"gender":"male","lookingFor":"female"}';  
const userObject = JSON.parse(jsonString);  
console.log(userObject); // logging userObject
```



Test endpoints



<https://jsonplaceholder.typicode.com/>

- Install [JSON Formatter](#)
- Test the following endpoints:
 - <https://jsonplaceholder.typicode.com/posts/>
 - <https://jsonplaceholder.typicode.com/posts/5>
 - <https://jsonplaceholder.typicode.com/users/>
 - <https://jsonplaceholder.typicode.com/users/1>
 - <https://jsonplaceholder.typicode.com/todos>
 - <https://jsonplaceholder.typicode.com/todos/5>

```
{  
  "name": "Rasmus Cederdorff",  
  "birthday": "1990-03-12",  
  "title": "Experienced JavaScript Developer",  
  "experienceYears": 10,  
  "education": {  
    "degree": "Master of Science in IT - Web Communication",  
    "specialization": "Web Architecture"  
  },  
  "skills": ["JavaScript", "React", "Node.js", "UI/UX Design"],  
  "currentPosition": "Senior Lecturer at EAAA",  
  "teachingSubjects": ["Web Development", "JavaScript", "React", "BaaS & Node.js"],  
  "lovesJavaScript": true  
}
```

Define yourself with JSON

- <https://race.notion.site/Define-yourself-with-JSON-391944b6b1a041bfa341a0d46f5c0e4a?pvs=4>

Array Methods

Array methods are built-in JavaScript functions that make it easier to work with lists of data.

Array Methods

Array methods are built-in JavaScript functions that make it easier to work with lists of data.

They can be used to:

- Loop through all items
(`.forEach`, `.map`)
- Filter items based on conditions
(`.filter`)
- Search or find specific items
(`.find`, `.some`, `.every`)
- Sort or reorder data
(`.sort`, `.reverse`)

Array Methods to know

.push (...)

.map (...)

.filter (...)

.find (...)

.sort (...)

Computer science student



Senior developer, 10+ years experience



<https://www.instagram.com/p/BxWAgatgSmn/>

Array methods

<https://javascript.info/array-methods#filter>

- Chapter
- Data types
- Lesson navigation
- Add/remove items
- Iterate: forEach
- Searching in array**
- Transform an array
- Array.isArray
- Most methods support "thisArg"
- Summary
- Tasks (13)
- Comments
- Share
- [Edit on GitHub](#)
- Ads



filter

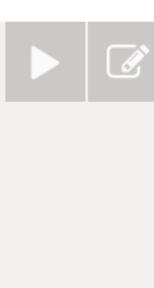
The `find` method looks for a single (first) element that makes the function return `true`. If there may be many, we can use `arr.filter(fn)`.

The syntax is similar to `find`, but `filter` returns an array of all matching elements:

```
1 let results = arr.filter(function(item, index, array) {  
2   // if true item is pushed to results and the iteration continues  
3   // returns empty array if nothing found  
4});
```

For instance:

```
1 let users = [  
2   {id: 1, name: "John"},  
3   {id: 2, name: "Pete"},  
4   {id: 3, name: "Mary"}  
5];  
6  
7 // returns array of the first two users  
8 let someUsers = users.filter(item => item.id < 3);  
9  
10 alert(someUsers.length); // 2
```



■ ■ ■ ■	.map(■ → ●)	→	● ● ● ●
■ ■ ● ■	.filter(■)	→	■ ■ ■
● ● ■ ■	.find(■)	→	■
● ● ● ■	.findIndexof(■)	→	3
■ ■ ■ ■	.fill(1, ●)	→	■ ● ● ●
● ■ ■ ●	.some(■)	→	true
■ ■ ■ ●	.every(■)	→	false

<https://javascript.info/array-methods>

<https://medium.com/@mandeepkaur1/a-list-of-javascript-array-methods-145d09dd19a0>

Arrays

.filter()

```
let users = [  
    { age: 35, name: "John" },  
    { age: 40, name: "Pete" },  
    { age: 44, name: "Mary" }  
];
```

// returns array of with users older than 39

```
let someUsers = users.filter(item => item.age > 39);
```

```
console.log(someUsers);
```

▼ Array(2) ⓘ
▶ 0: {age: 40, name: "Pete"}
▶ 1: {age: 44, name: "Mary"}
length: 2

Arrays

.filter() to search

```
const persons = [
  {
    name: "Birgitte Kirk Iversen",
    mail: "bki@mail.dk",
    title: "Senior Lecturer",
    img: "https://www.eaaa.dk/media/u4gorzsd/birgitte-kirk-iversen2.jpg?width=800&height=450"
  },
  {
    name: "Martin Aagaard Nøhr",
    mail: "mnor@mail.dk",
    title: "Lecturer",
    img: "https://www.eaaa.dk/media/oayjq02h/martin-n%C3%88hr.jpg?width=800&height=450"
  },
  {
    name: "Rasmus Cederdorff",
    mail: "rcae@mail.dk",
    title: "Senior Lecturer",
    img: "https://www.eaaa.dk/media/devlvgj/rasmus-cederdorff.jpg?width=800&height=450"
  },
  {
    name: "Dan Okkels Brendstrup",
    mail: "dob@mail.dk",
    title: "Lecturer",
    img: "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstrup.jpg?width=800&height=450"
  },
  {
    name: "Anne Kirketerp",
    mail: "anki@mail.dk",
    title: "Head of Department",
    img: "https://www.baaa.dk/media/5buh1xeo/anne-kirketerp.jpg?width=800&height=450"
  }
];
```

```
function search(event) {
  const searchValue = event.target.value.toLowerCase(); // input text to lower case
  const results = persons.filter(function (person) {
    return person.name.toLowerCase().includes(searchValue);
  });
  displayPersons(results); // call displayPersons with the filtered results
}

// with arrow function
function search(event) {
  const searchValue = event.target.value.toLowerCase(); // input text to lower case
  const results = persons.filter(person => person.name.toLowerCase().includes(searchValue));
  displayPersons(results); // call displayPersons with the filtered results
}
```

Filter condition:
person.name
must include
searchValue

Arrays

.sort()

JavaScript Demo: Array.sort()

```
1 const months = ['March', 'Jan', 'Feb', 'Dec'];
2 months.sort();
3 console.log(months);
4 // expected output: Array ["Dec", "Feb", "Jan", "March"]
5
6 const array1 = [1, 30, 4, 21, 100000];
7 array1.sort();
8 console.log(array1);
9 // expected output: Array [1, 100000, 21, 30, 4]
10
```

“The `sort()` method sorts the elements of an array in place and returns the reference to the same array, now sorted.”

Arrays w/ objects

.sort()

```
const persons = [
  {
    name: "Birgitte Kirk Iversen",
    mail: "bki@mail.dk",
    title: "Senior Lecturer",
    img: "https://www.eaaa.dk/media/u4gorzsd/birgitte-kirk-iversen2.jpg?width=800&height=450"
  },
  {
    name: "Martin Aagaard Nøhr",
    mail: "mnor@mail.dk",
    title: "Lecturer",
    img: "https://www.eaaa.dk/media/oayjq02h/martin-n%C3%B8hr.jpg?width=800&height=450"
  },
  {
    name: "Rasmus Cederdorff",
    mail: "race@mail.dk",
    title: "Senior Lecturer",
    img: "https://www.eaaa.dk/media/devlvgj/rasmus-cederdorff.jpg?width=800&height=450"
  },
  {
    name: "Dan Okkels Brendstrup",
    mail: "dob@mail.dk",
    title: "Lecturer",
    img: "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstrup.jpg?width=800&height=450"
  },
  {
    name: "Anne Kirketerp",
    mail: "anki@mail.dk",
    title: "Head of Department",
    img: "https://www.baaa.dk/media/5buh1xeo/anne-kirketerp.jpg?width=800&height=450"
  }
];
```

```
// Use localeCompare for strings
persons.sort(function (person1, person2) {
  person1.name.localeCompare(person2.name);
});

// with arrow function
persons.sort((person1, person2) => person1.name.localeCompare(person2.name));
```

Code Every Day



A screenshot of the PhpStorm IDE interface. The title bar reads "PhpStorm". The left sidebar shows a project structure with files like "index.php", "main.js", and "style.css". The main editor area displays a portion of the "main.js" file with code in JavaScript. A large yellow arrow points from the text "Code Every Day" towards the "main.js" file in the editor.

```
</script>
</head>
<body>
<main>
<section id="intro--section">
<article id="intro--section--text">



Hey, ik ben Arnold Francisca!



Ik ben een front-end developer en student applicatiewetenschappen.


<a href="#work--section" class="pink--button scroll">Bekijk mijn werk</a>
</article>

<section id="skills--section">
<h2 class="section--header">Mijn Skills.</h2>
<section id="skills--section--wrap">
```