

# JavaScript Concepts

“

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.

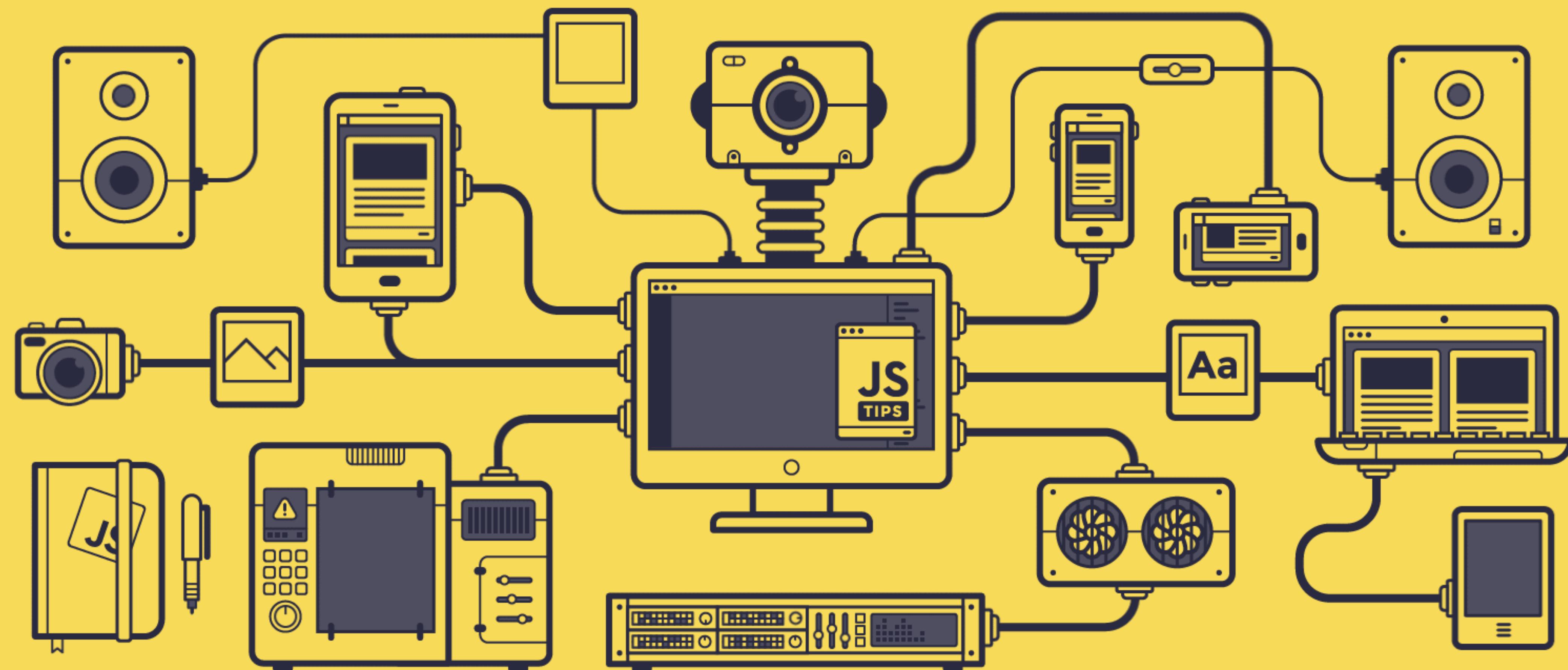
”

**Lex Fridman**  
Computer Scientist

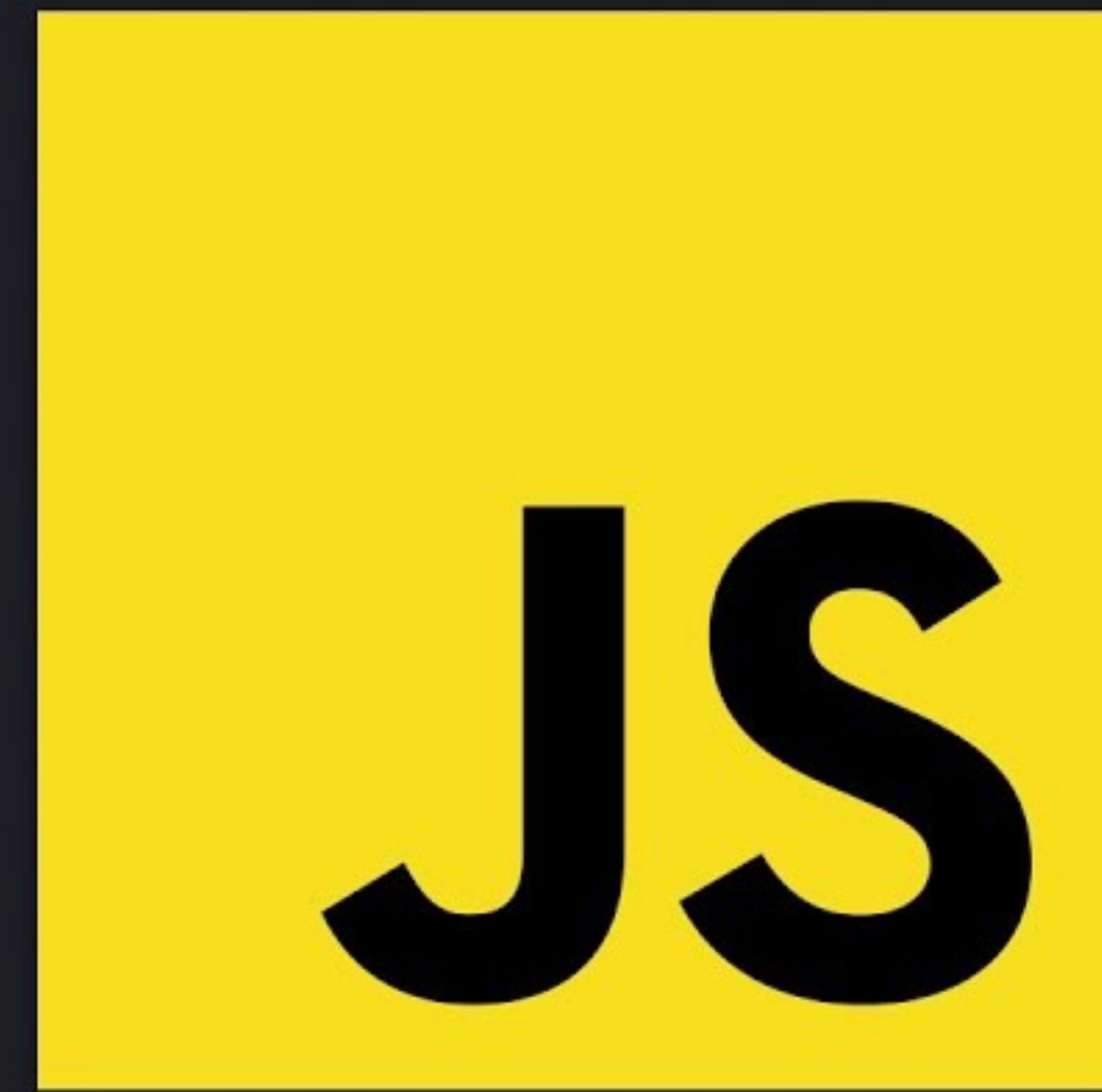
<https://www.youtube.com/watch?v=GLhyjVZp0cw&t=252s>

# Content

- What is JavaScript?
- DOM Manipulation
- Variables
- Data Types
- Objects
- Arrays
- Loops
- Conditional Statements
- Ternary operator
- The use of “signs & symbols”
- Array Methods
- Template String
- Functions
- Destructuring
- Spread Operator
- Modules, import & Export
- Form



# **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>
14
```

# 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 

# DOM Manipulation

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

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

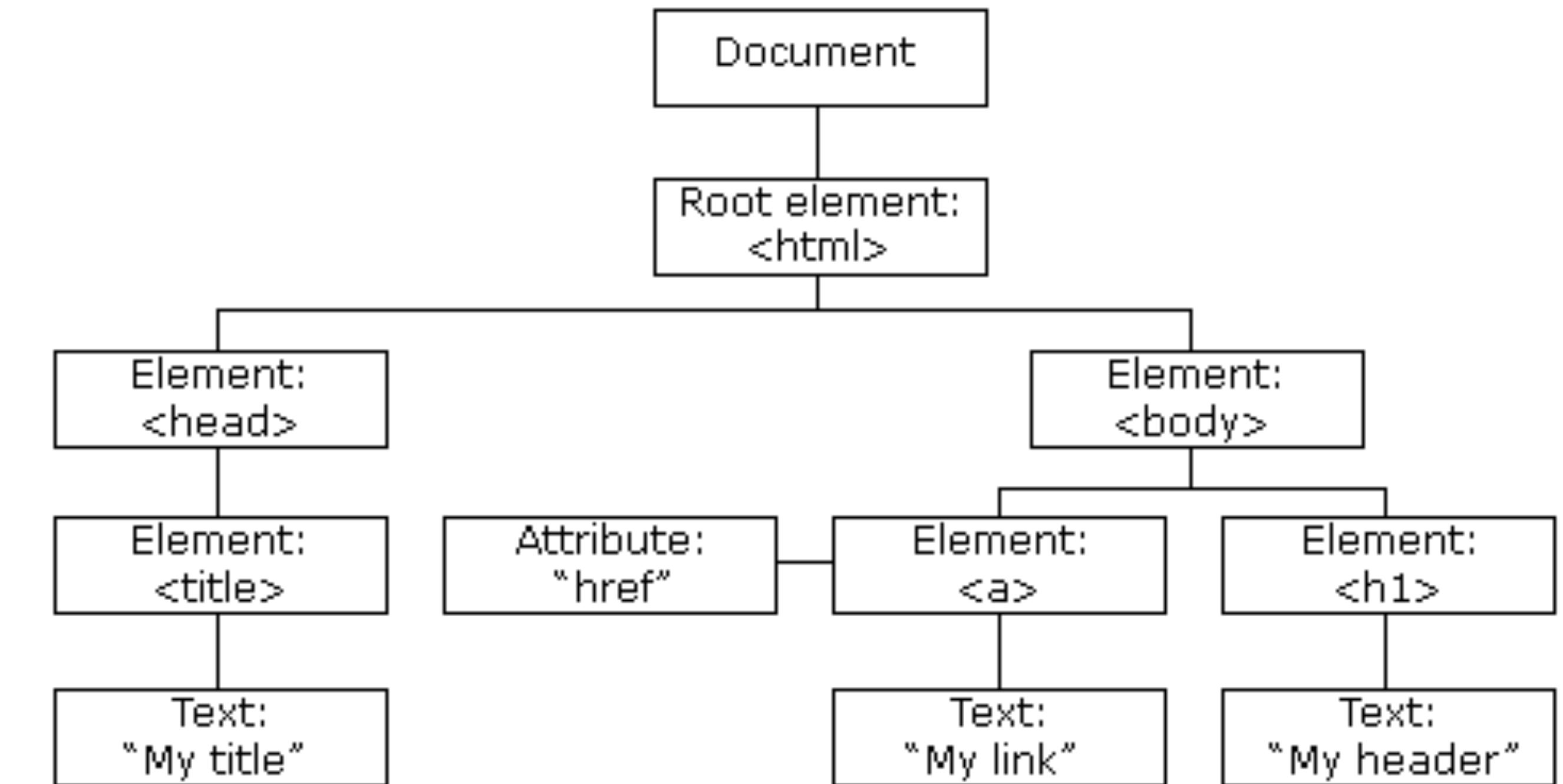
// appending the variable (the string) to the DOM element #content
document.querySelector("#content").innerHTML = message;
```

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



# JavaScript HTML DOM

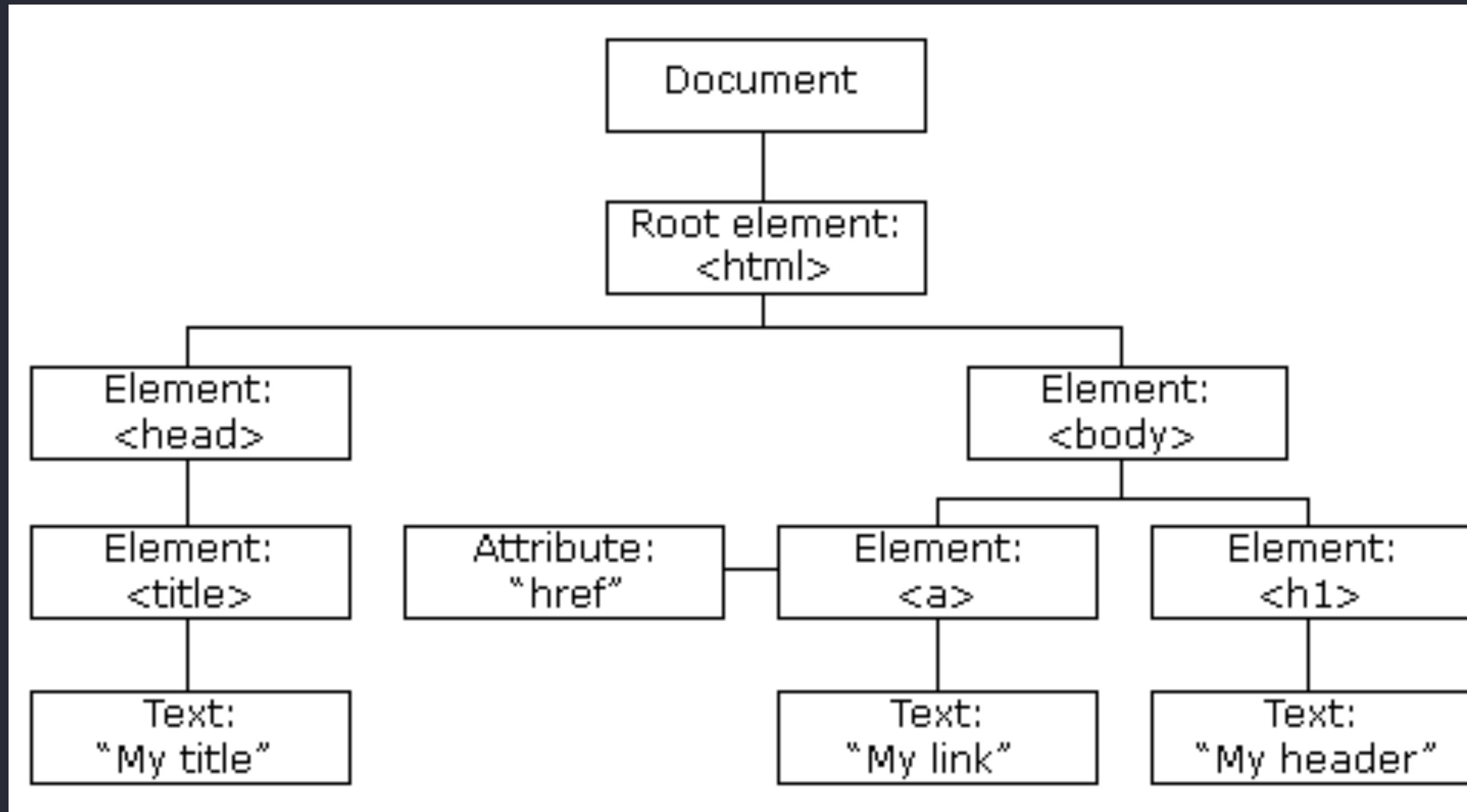
```
index.html *  
1  <!DOCTYPE html>  
2  <html>  
3  | <head>  
4  | | <title>My title</title>  
5  | </head>  
6  |  
7  <body>  
8  | | <h1>My header</h1>  
9  | | <a href="https://cederdorff.com">My link</a>  
10 | </body>  
11 |  
12 </html>
```



[https://www.w3schools.com/js/js\\_htmldom.asp](https://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://www.w3schools.com/js/js_htmldom.asp)  
<https://javascript.info/dom-nodes>  
<https://javascript.info/dom-navigation>

## DOM

```
● ○ ● Elements  
1 <html>  
2   <head></head>  
3   <body>  
4     <div id="app">  
5       <h1>Develop. Preview.  
6     Ship. 🚀</h1>  
7     </div>  
8     <script type="text/  
9   javascript">...</script>  
10    </body>  
11  </html>
```

## SOURCE CODE (HTML)

```
● ○ ● index.html  
1 <html>  
2   <head></head>  
3   <body>  
4     <div id="app"></div>  
5     <script type="text/  
6   javascript">...</script>  
7   </body>  
8 </html>  
9  
10  
11
```

# 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

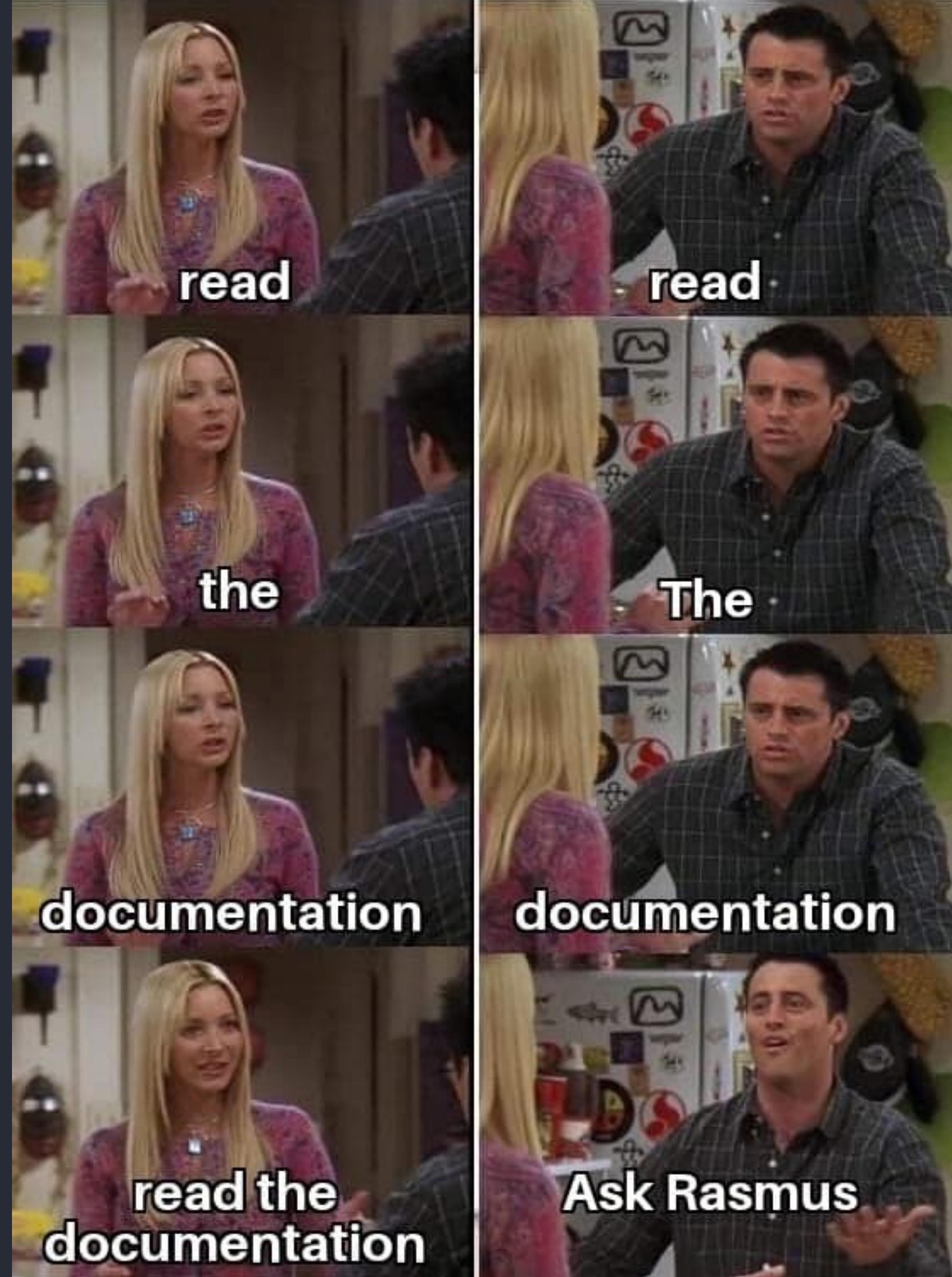
`getElements*` or `querySelector*`?

**ES6 +**

**Modern JavaScript**

# Modern JavaScript

LET & CONST  
TEMPLATE STRING  
ARROW FUNCTIONS  
FETCH  
PROMISES  
ASYNC & AWAIT  
FOR OF LOOP  
ARRAY.FIND()  
ARRAY.MAP()  
ARRAY.REDUCE()  
ARRAY.FILTER()  
ARRAY.SORT()  
ARRAY.CONCAT()  
DEFAULT PARAMS  
DESTRUCTURING OBJECTS  
DESTRUCTURING ARRAYS  
OBJECT LITERAL  
SPREAD OPERATOR  
CLASSES  
MODULES  
IMPORT & EXPORT



# Programming knowledge

## JavaScript



# Variables

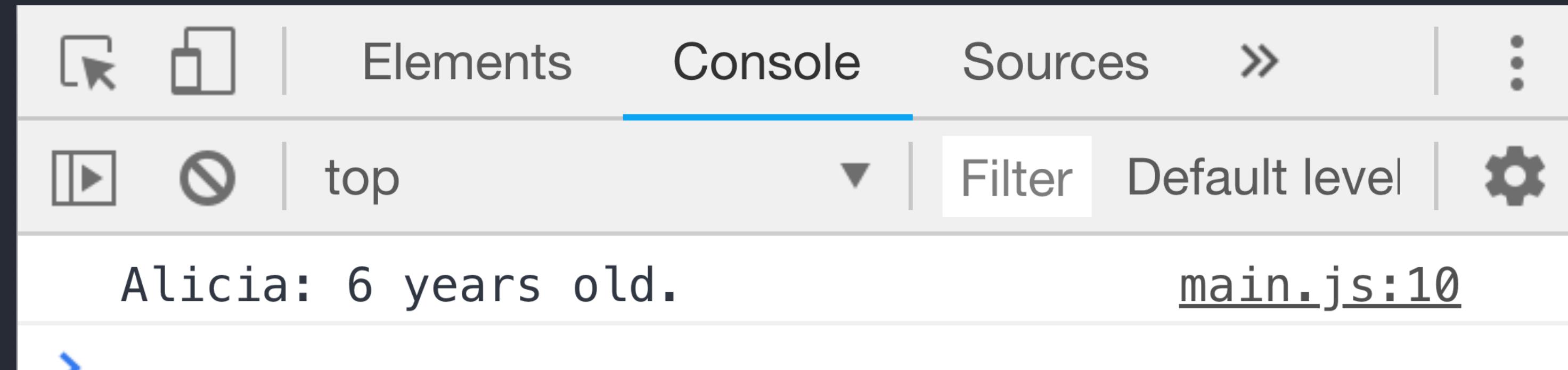
... are used to store data (values, objects, collections) in the memory

# Variables

Store data in the memory

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

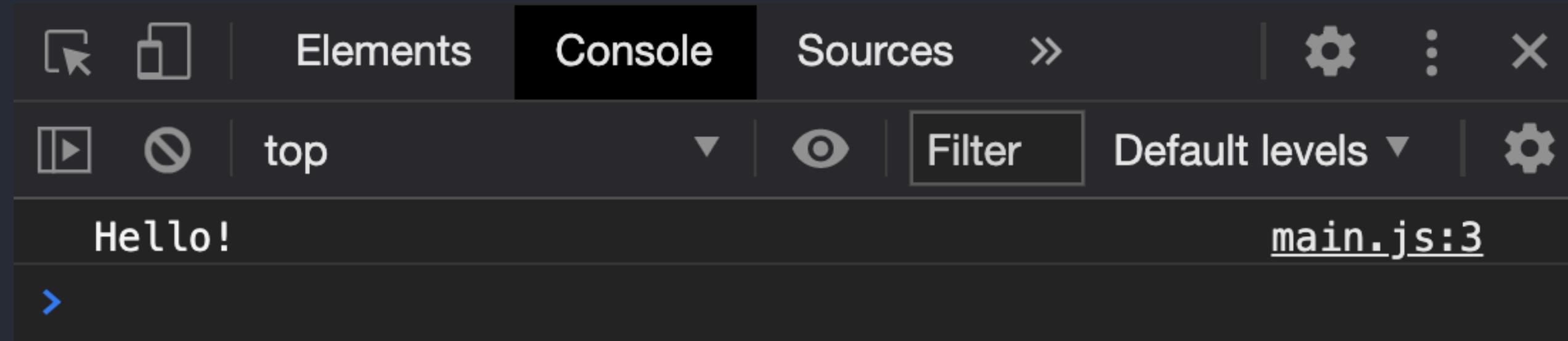
console.log(name + ": " + age + " years old.");
```



# Variables

Store data in the memory

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

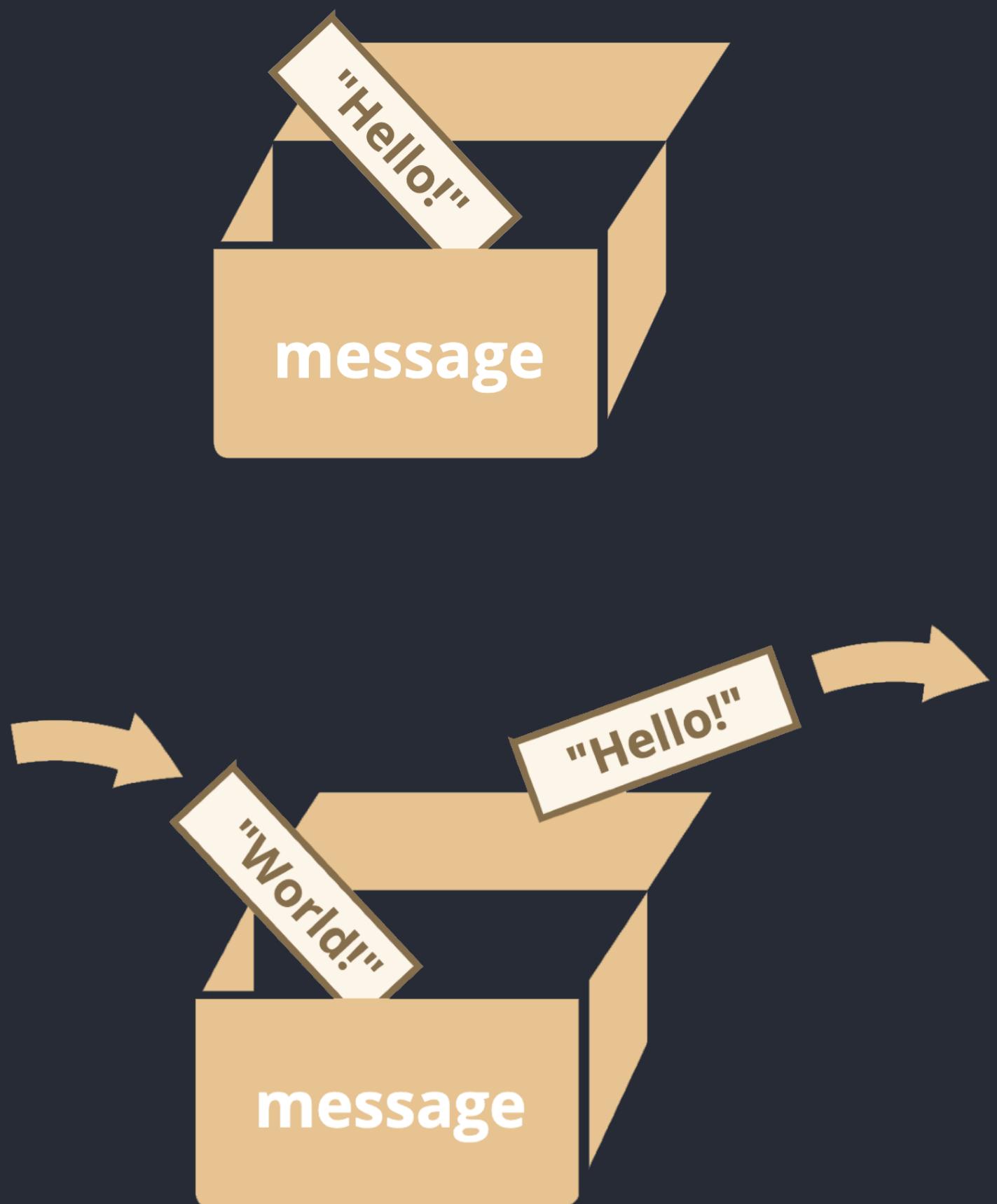


# Variables

Store data in the memory

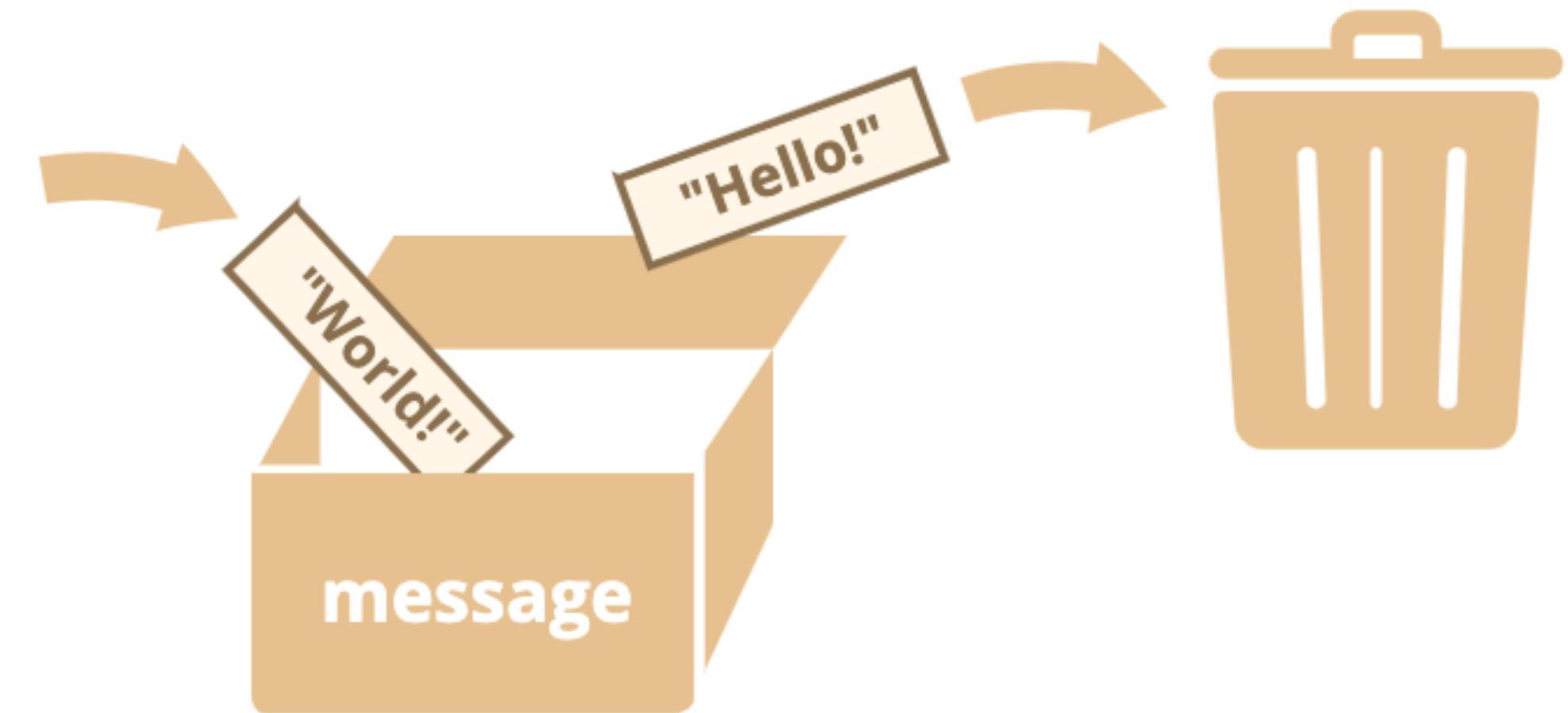
```
let message = "Hello!";
console.log(message);
```

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



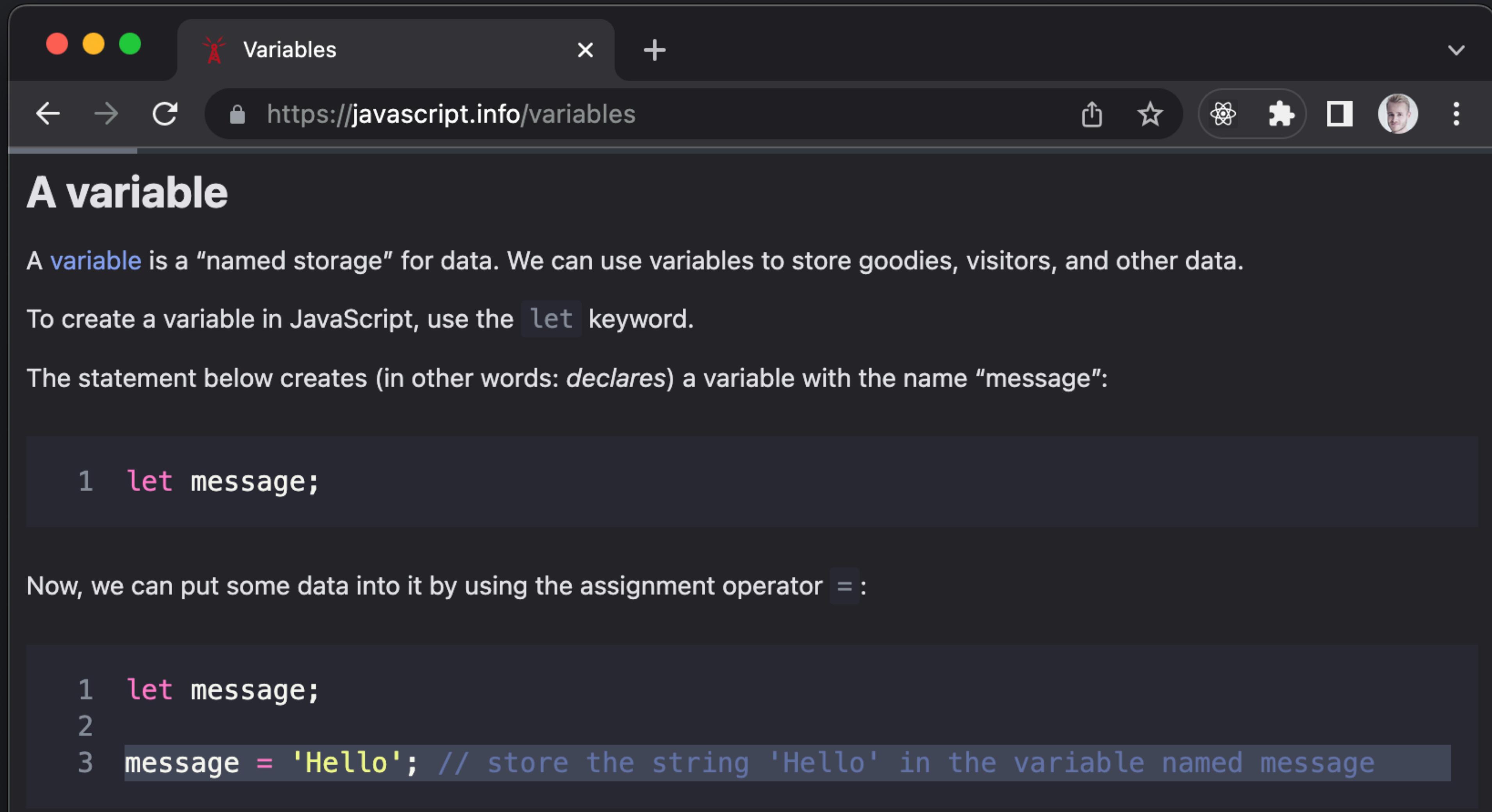
# Variable

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.

# JavaScript.info/Variables



The screenshot shows a dark-themed web browser window. The title bar reads "Variables". The address bar shows the URL "https://javascript.info/variables". The main content area displays the following text:

## 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 `=`:

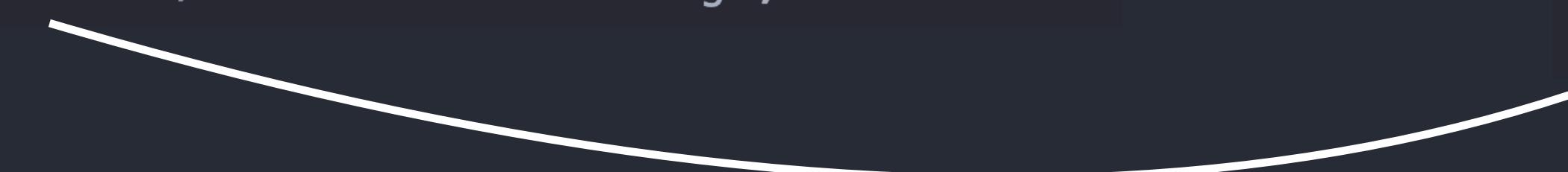
```
1 let message;
2
3 message = 'Hello'; // store the string 'Hello' in the variable named message
```

```
// 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>
```

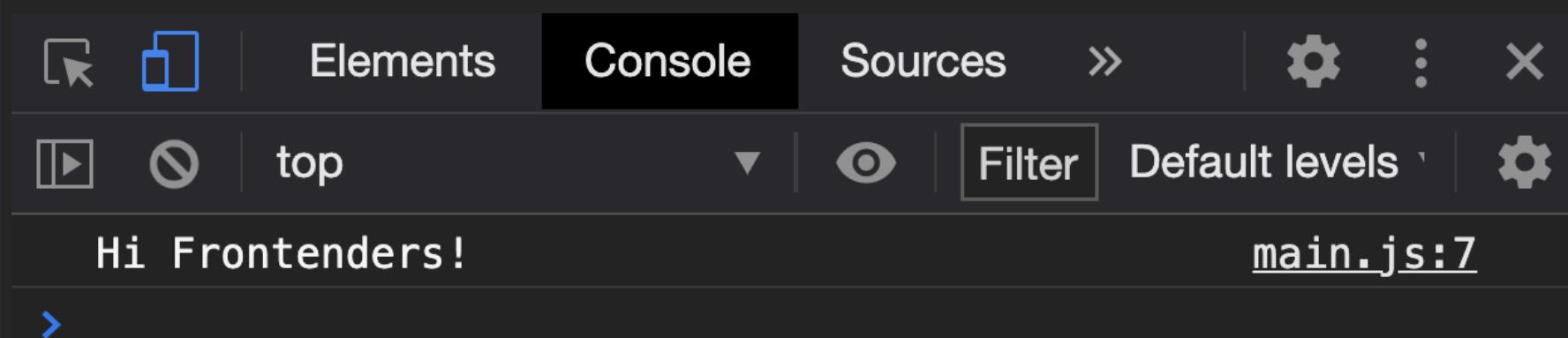


```
// declaring a variable with a value
let message = "Hi Frontenders!"

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

// changing the value of the variable
message = "Hello World";

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



# 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
```

var-vs-let

# 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.

# 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>

## 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`.

# 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.

# 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

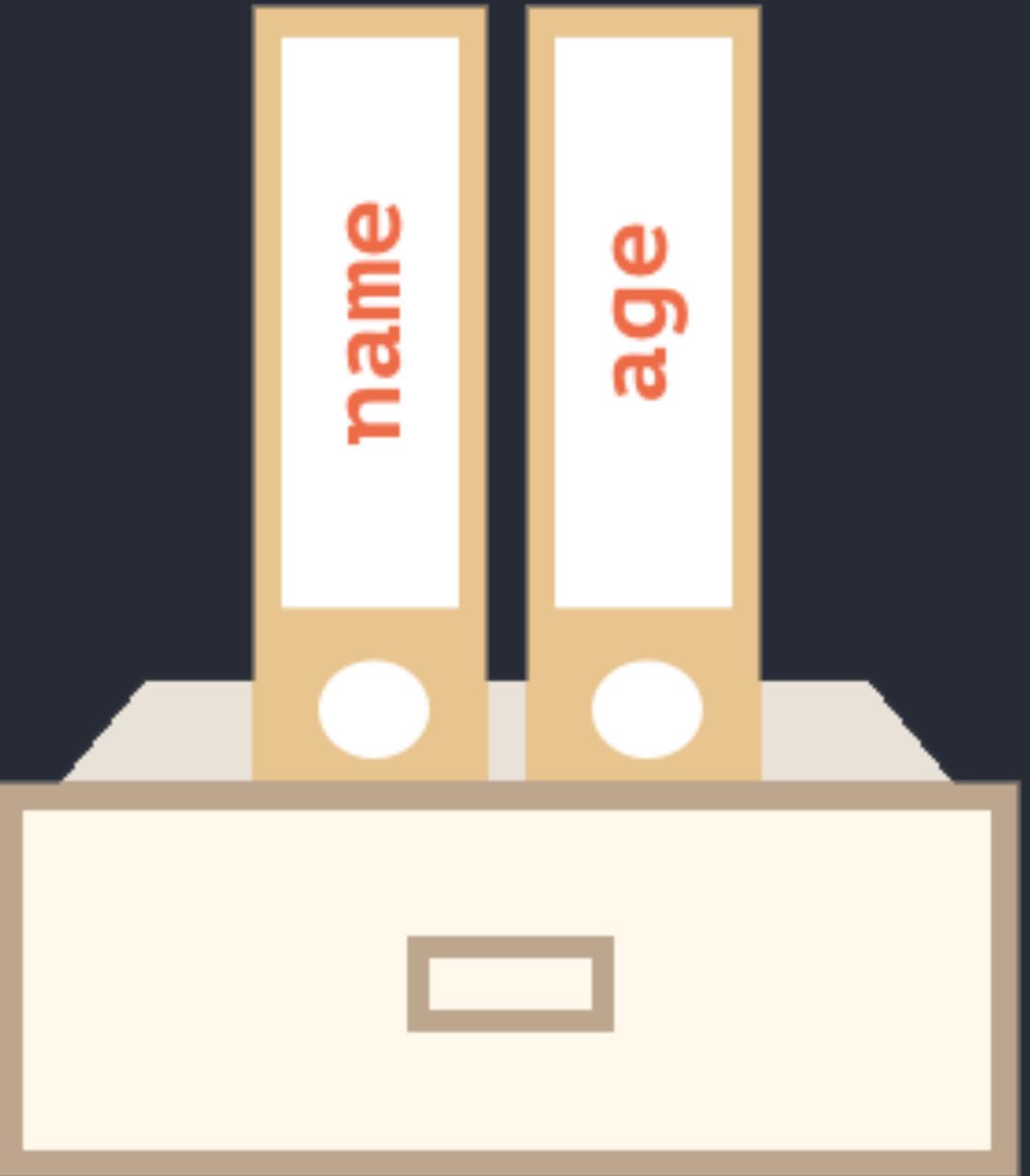
# Objects

A set of named values

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

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

user

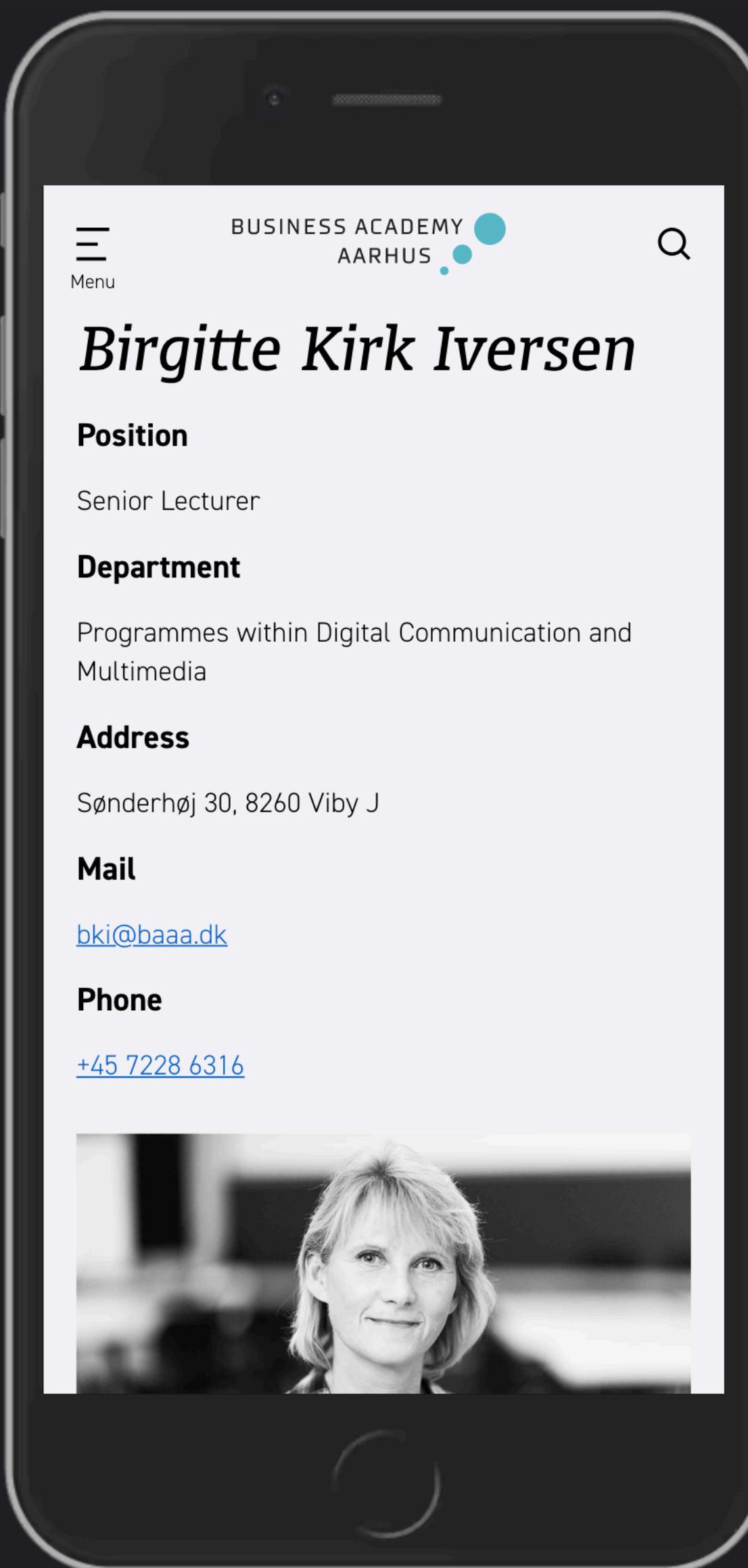


Alicia is 6 years old.

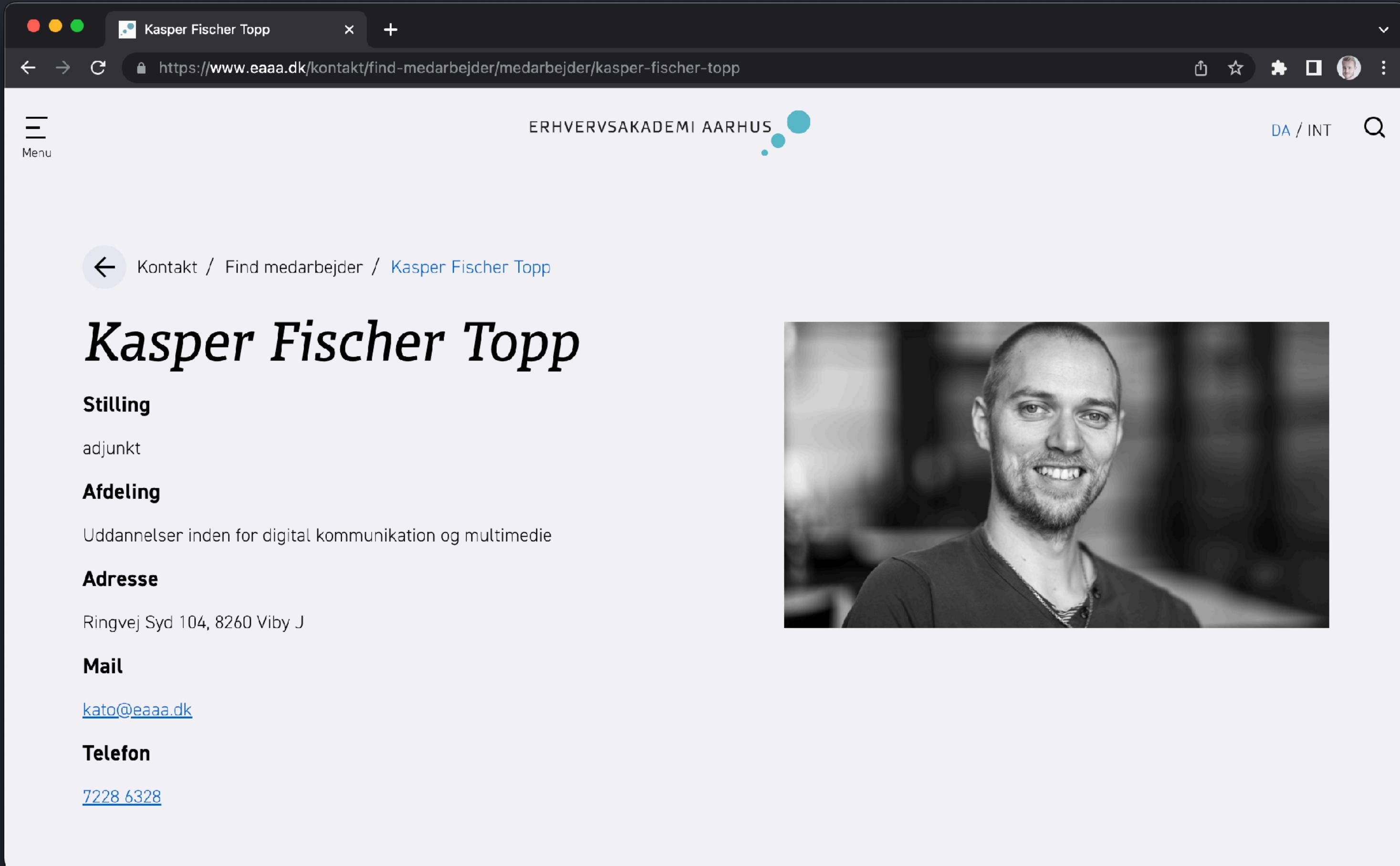
main.js:11

# Objects

# A set of named values



# Objects



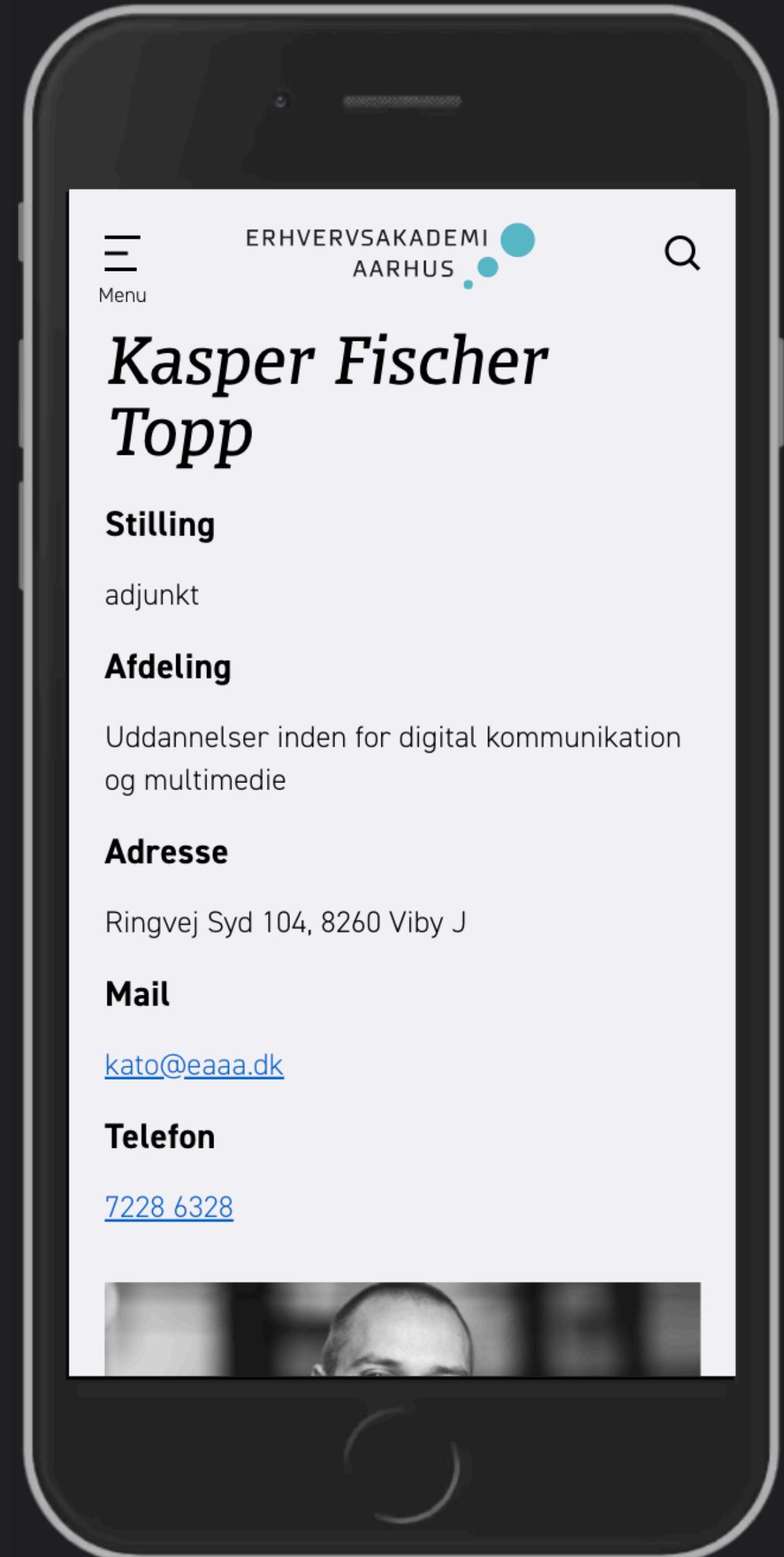
A screenshot of a web browser window displaying a profile page for Kasper Fischer Topp. The browser has a dark theme with red, yellow, and green window control buttons. The title bar shows the page is titled "Kasper Fischer Topp". The address bar contains the URL <https://www.eaaa.dk/kontakt/find-medarbejder/medarbejder/kasper-fischer-topp>. The page itself is from ERHVERVSAKADEMI AARHUS, featuring a logo with three blue dots. The main content area includes a navigation menu, a search bar, and a breadcrumb trail: "Kontakt / Find medarbejder / Kasper Fischer Topp". The profile section features a large black and white portrait of a smiling man with a beard. Below the photo, the name "Kasper Fischer Topp" is displayed in a large, bold, italicized font. Underneath the name, there are sections for "Stilling" (adjunkt), "Afdeling" (Uddannelser inden for digital kommunikation og multimedie), "Adresse" (Ringvej Syd 104, 8260 Viby J), "Mail" (kato@eaaa.dk), and "Telefon" (7228 6328).

<https://www.eaaa.dk/kontakt/find-medarbejder/medarbejder/kasper-fischer-topp>

# Objects

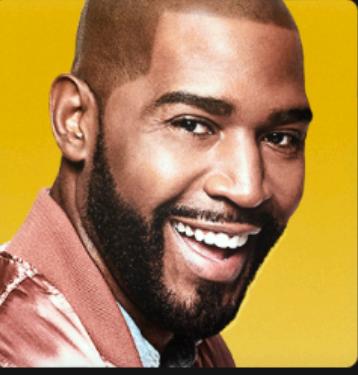
A set of named values

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



NETFLIX

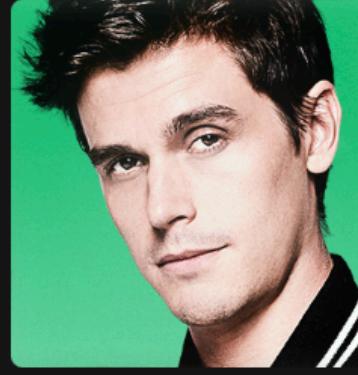
# Hvem ser?



Personen der  
rent faktisk  
betaler for  
profilen



Nasser 1



Nasser 2



Nasser 3



Nasser 4 Khader

[Administrer profiler](#)

● ○ ● | □ | < > ⌂ +

netflix.com

NETFLIX Start Serier Film Nyt og populært Min liste

N SERIE

# TOO HOT TO HANDLE

**TOP 10** Nr. 4 i Danmark i dag

På paradisets kyst mødes de lækkere singler og mingler. Men der er et tvist. For at vinde den attraktive pengepræmie, må de give afkald på at have sex.

Afspil Mere info

13+

Kun på Netflix

TOO HOT TO HANDLE NYE EPISODER

EMILY IN PARIS

QUEER EYE more than a makeover

The Woman in the House Across the Street From the Girl in the Window

BRIDGERTON NYE EPISODER

Se videre med profilen Nasser 1

the office

TIGER KING

Don't Look UP

JEFFREY EPSTEIN: FILTHY RICH

THE MIND explained

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

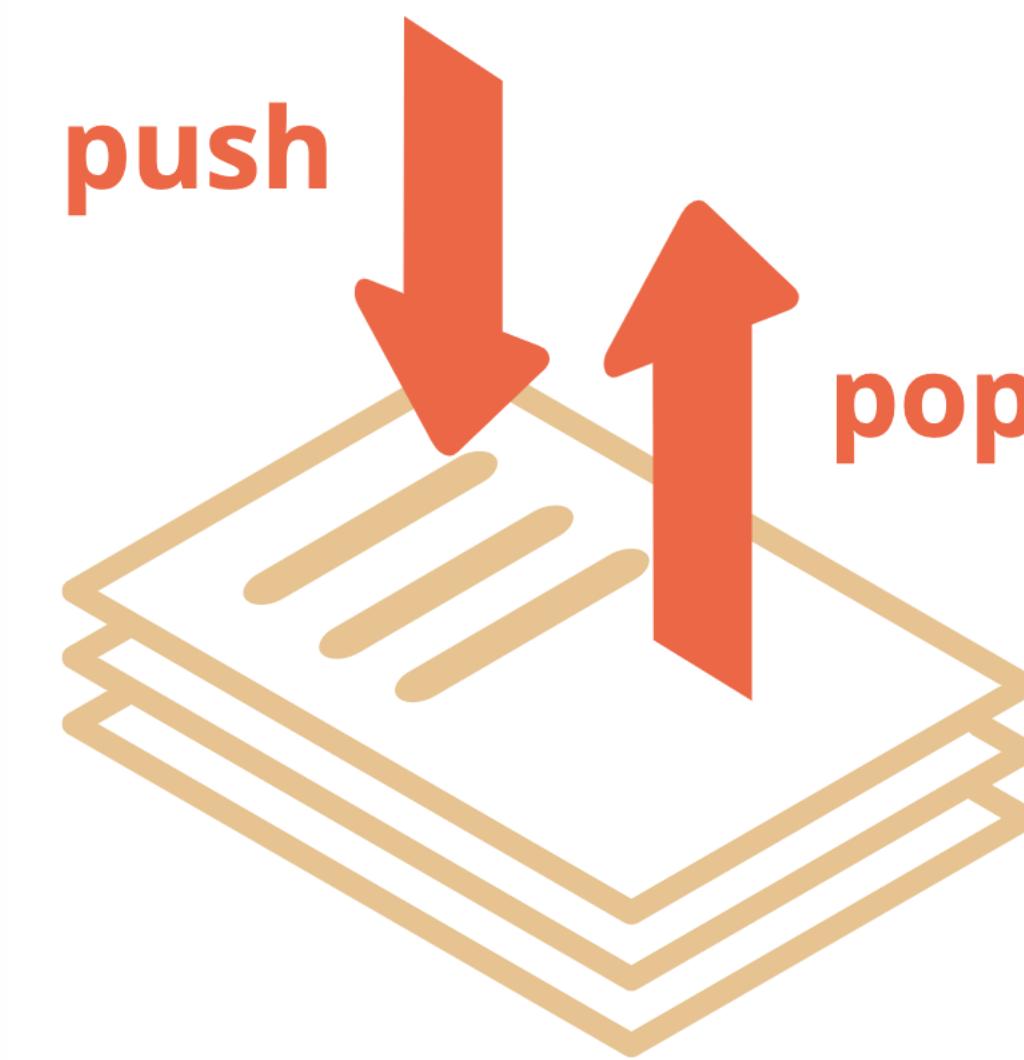
with the following properties

name, age, mail, phone, city, address

# Arrays

## 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.

```
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	
<b>Position:</b> Lecturer	<i>Michael Hvidtfeldt</i>
<b>Department/</b> Multimedia De Digital Conce	
<b>Address:</b> Ringvej Syd 10	<b>Position:</b> Lecturer <i>Birgitte Kirk Iversen</i>
<b>Mail:</b> <a href="mailto:race@baaa.dk">race@baaa.dk</a>	<b>Department/</b> Multimedia Di
<b>Phone:</b> 7228 6318	<b>Address:</b> Senior Lecturer Ringvej Syd 10
	<b>Department/programme:</b> Multimedia Design
<b>Mail:</b> <a href="mailto:mhv@baaa.dk">mhv@baaa.dk</a>	<b>Address:</b> Sønderhøj 30, 8260 Viby J
<b>Phone:</b> 7228 6328	<b>Mail:</b> <a href="mailto:bki@baaa.dk">bki@baaa.dk</a>
	<b>Phone:</b> 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](mailto:bki@baaa.dk)



Maria Louise Bendixen

Senior Lecturer  
[mlbe@baaa.dk](mailto:mlbe@baaa.dk)



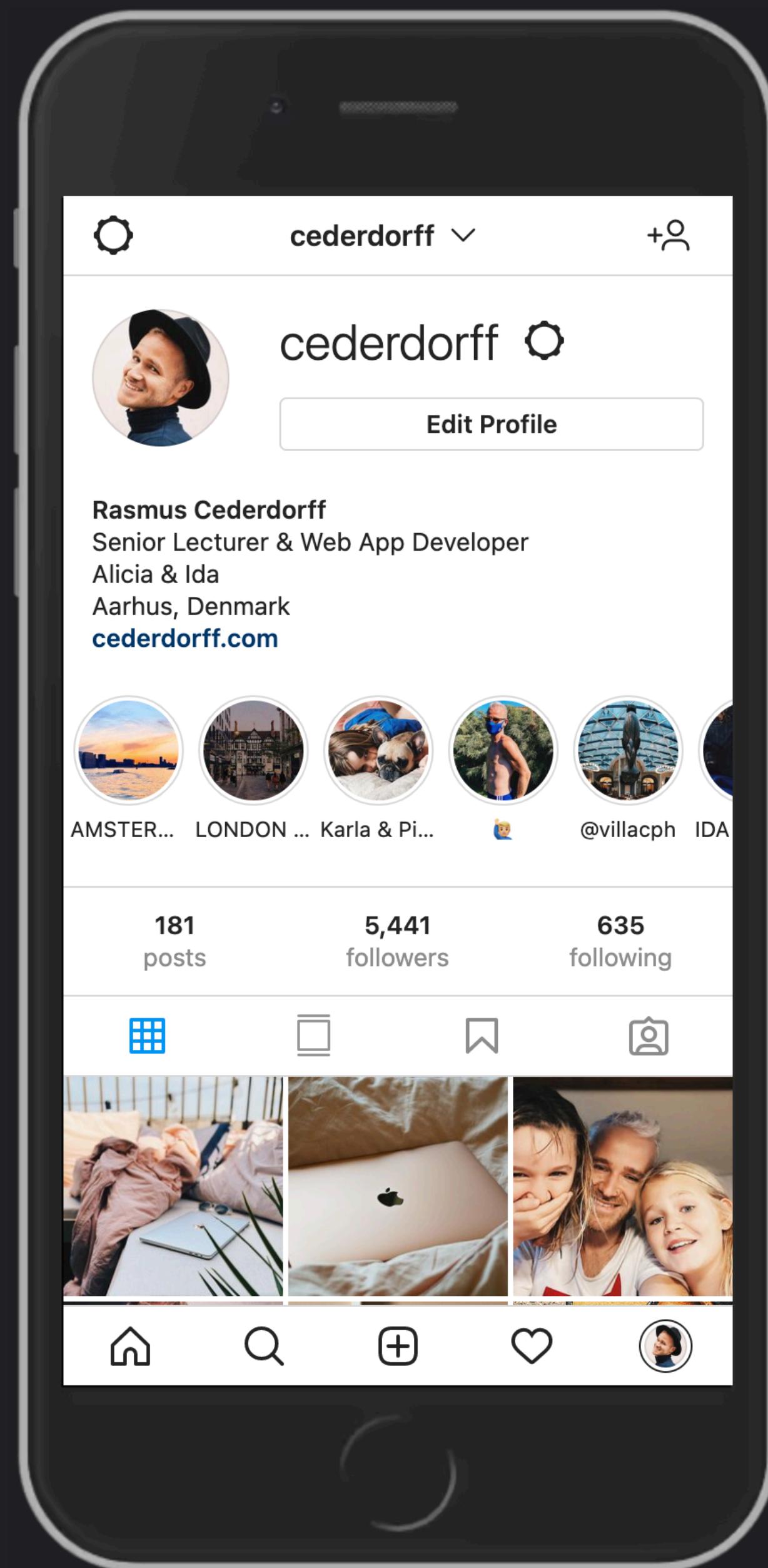
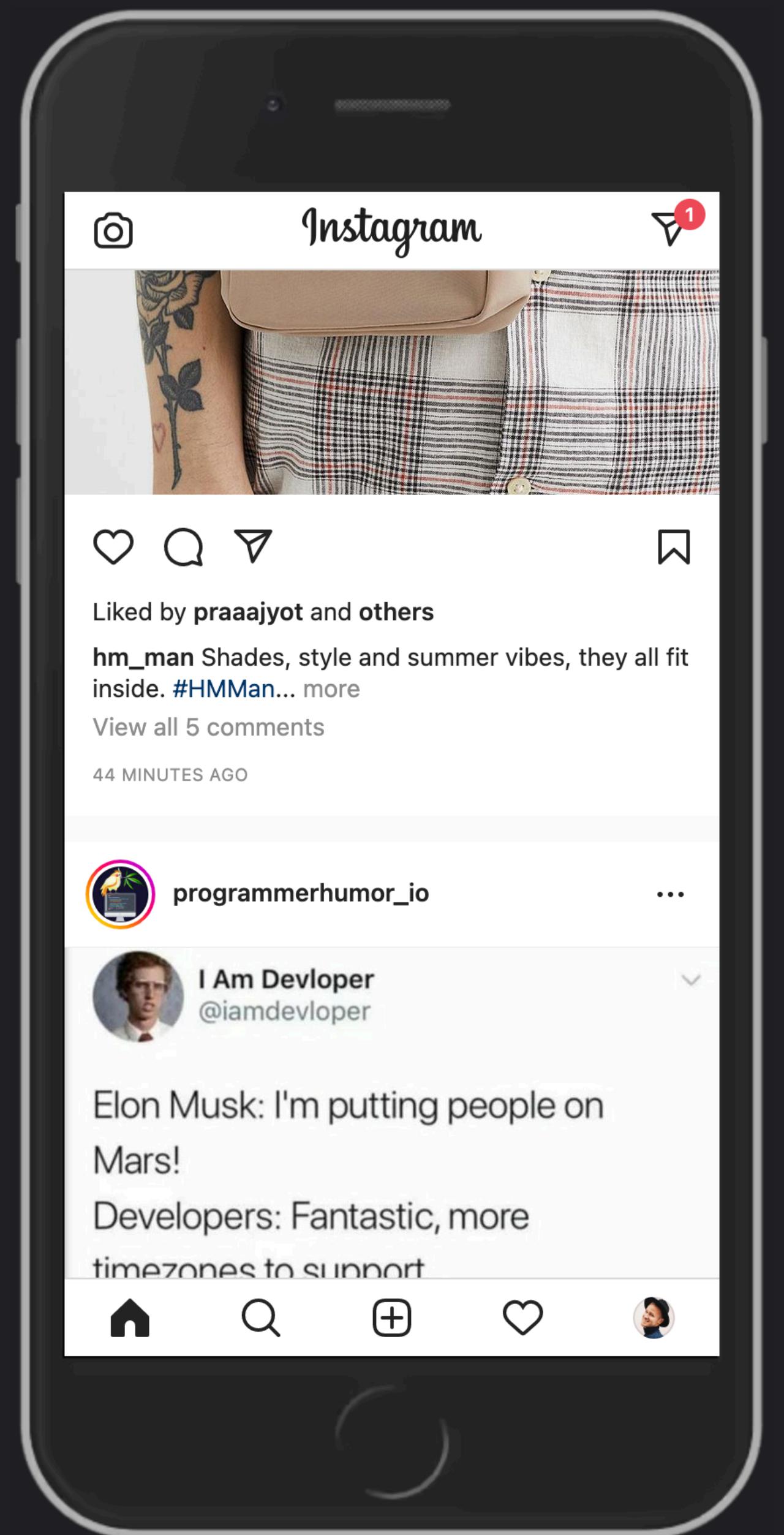
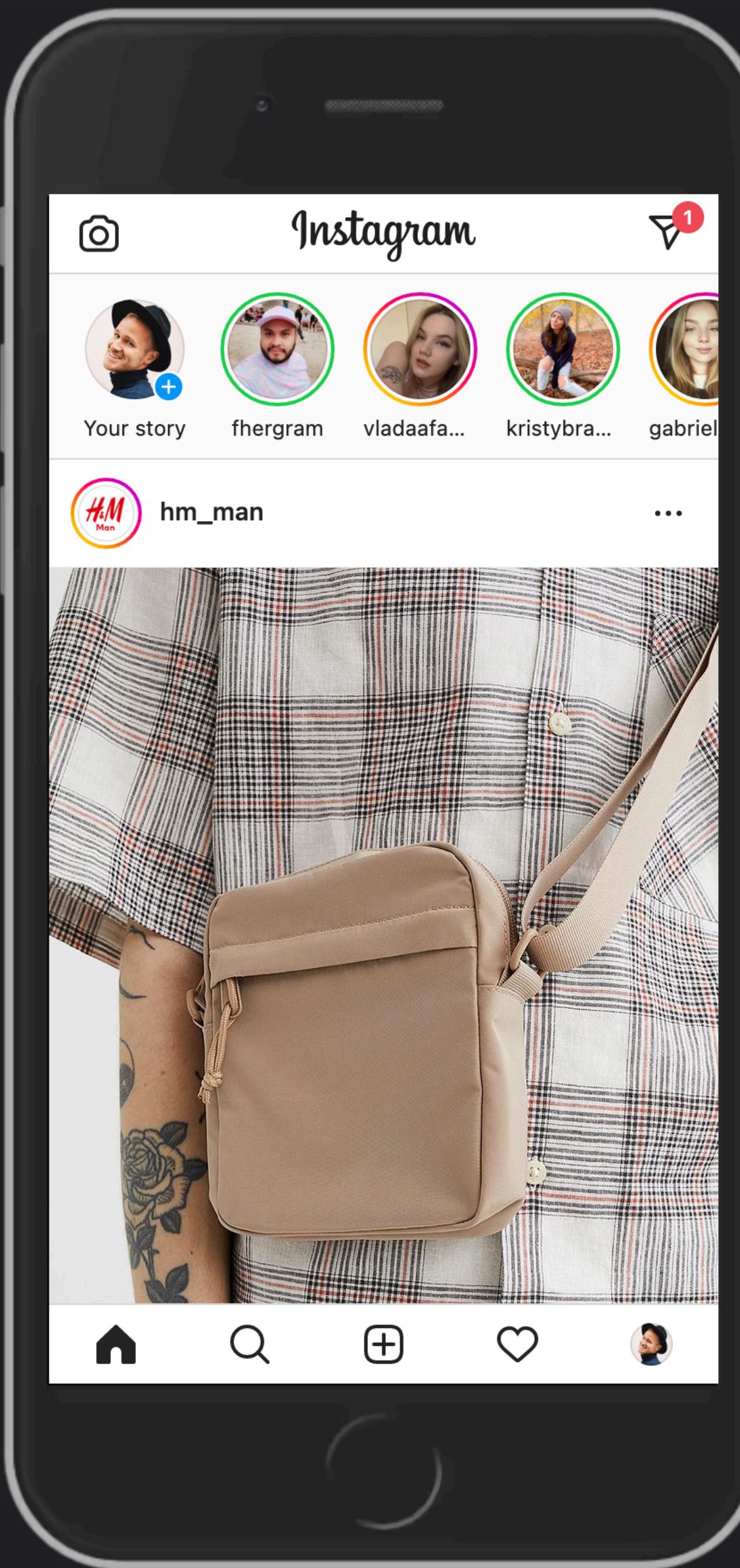
Kim Elkjær Marcher-Jepsen

Lecturer  
[kje@baaa.dk](mailto:kje@baaa.dk)

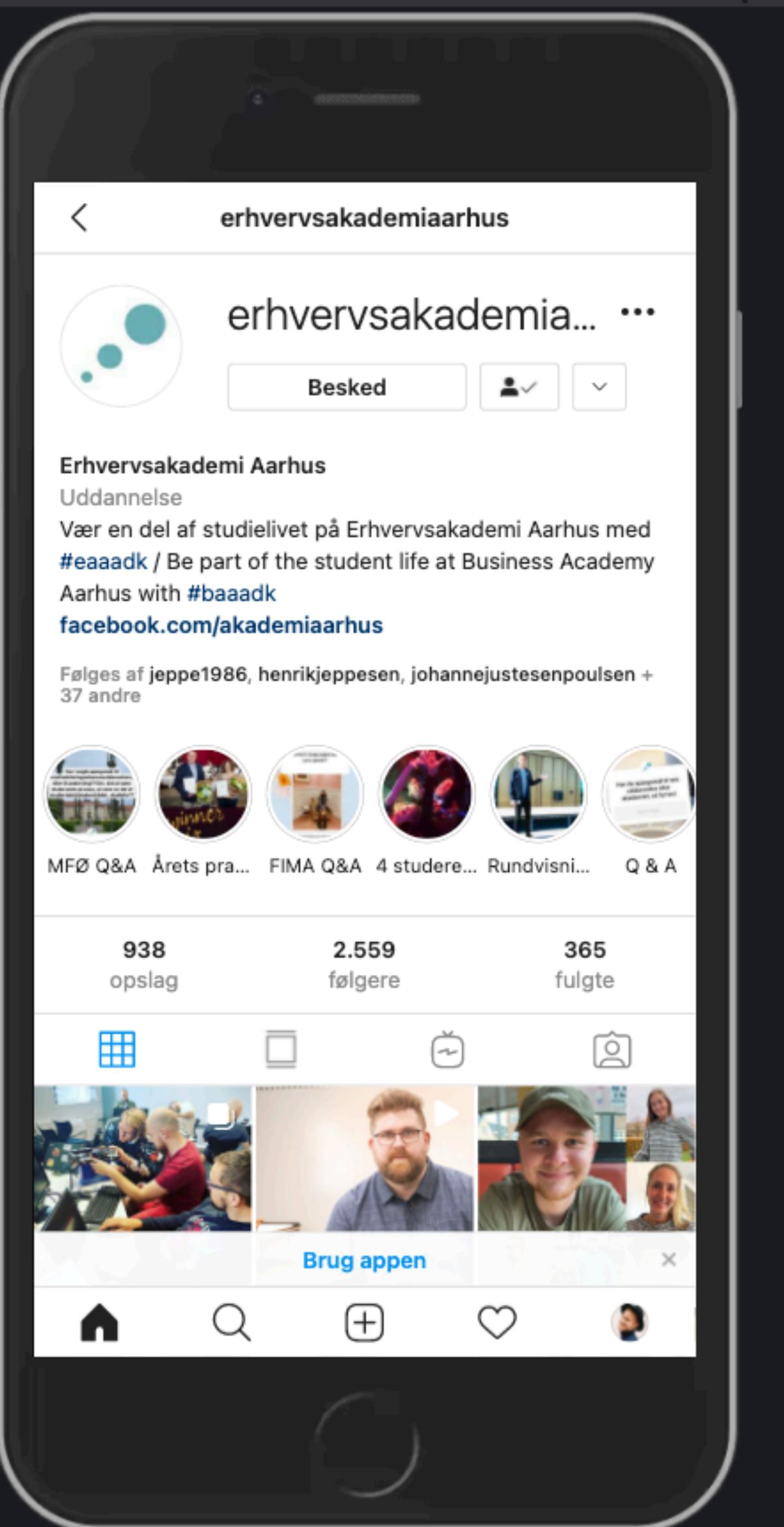


Rasmus Cederdorff

Lecturer  
[race@baaa.dk](mailto:race@baaa.dk)







The screenshot shows the Chrome DevTools Network tab. The timeline at the top indicates request times from 5000 ms to 35000 ms. Below the timeline, a list of requests is shown on the left, and a detailed preview of a selected request is on the right.

**Selected Request Preview:**

- Headers:** Headers for the selected request.
- Preview:** Detailed JSON preview of the response body. The response is a GraphQL query result for a video media type. It includes fields for user information, edge\_web\_feed\_timeline (with edges), node (GraphVideo), and various edge\_media\_\* fields.
- Response:** Raw JSON response body.
- Initiator:** Details about the initiator of the request.
- Timing:** Timing details for the request.
- Cookies:** Cookies associated with the request.

**Request List:**

- reels\_tray/
- ?query\_hash=db...
- ?query\_hash=6ff...
- badge/
- logging\_client\_e...
- bz
- falco
- ?\_\_a=1
- logging\_client\_e...
- falco
- batch\_fetch\_web/
- ?query\_hash=d4...
- ?query\_hash=8c...
- ?query\_hash=8c...
- logging\_client\_e...
- falco

At the bottom, it says "16 / 97 requests | 17".

Course roster: WU-E22a - 1. se

<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

6 / 157 requests

Elements Components Network

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

property Headers Payload Preview Response Initiator

```
[{"id": "23974", "name": "Clara Juul Birk", "created_at": "2020-08-10T10:45:00+02:00", "email": "eaaclbi@students.eaaa.dk", "sis_user_id": null, "short_name": "Clara Juul Birk", "sortable_name": "Birk, Clara Juul", "integration_id": null, "login_id": "eaaclbi@students.eaaa.dk", "custom_links": []}, {"id": "36267", "name": "Martin Rieper Boesen", "created_at": "2021-07-30T00:46:05+02:00", "email": "eaamrbo@students.eaaa.dk", "sis_user_id": null, "short_name": "Martin Rieper Boesen", "sortable_name": "Boesen, Martin Rieper", "integration_id": null, "login_id": "eaamrbo@students.eaaa.dk", "custom_links": []}, {"id": "29923", "name": "Dan Okkels Brendstrup", "created_at": "2021-07-30T00:46:05+02:00", "email": "dob@eaaa.dk", "sis_user_id": null, "short_name": "Dan Okkels Brendstrup (adjunkt – dob@eaaa.dk)", "sortable_name": "Brendstrup, Dan Okkels", "integration_id": null, "login_id": "dob@eaaa.dk", "custom_links": []}, {"id": "14427", "name": "Rasmus Cederdorff", "created_at": "2021-07-30T00:46:05+02:00", "email": "race@eaaa.dk", "sis_user_id": null, "short_name": "Rasmus Cederdorff", "sortable_name": "Cederdorff, Rasmus", "integration_id": null, "login_id": "race@eaaa.dk", "custom_links": []}, {"id": "41", "name": "Jeffrey David Serio", "created_at": "2021-07-30T00:46:05+02:00", "email": "jds@eaaa.dk", "sis_user_id": null, "short_name": "Jeffrey David Serio", "sortable_name": "Serio, Jeffrey David", "integration_id": null, "login_id": "jds@eaaa.dk", "custom_links": []}, {"id": "24043", "name": "Charlotte Meng Emanuel Dyrholm", "created_at": "2021-07-30T00:46:05+02:00", "email": "eaacmed@students.eaaa.dk", "sis_user_id": null, "short_name": "Charlotte Meng Emanuel Dyrholm", "sortable_name": "Dyrholm, Charlotte Meng Emanuel", "integration_id": null, "login_id": "eaacmed@students.eaaa.dk", "custom_links": []}, {"id": "23978", "name": "Jeppe Frik", "created_at": "2020-08-03T10:45:00+02:00", "email": null, "sis_user_id": null, "short_name": "Jeppe Frik", "sortable_name": "Frik, Jeppe", "integration_id": null, "login_id": null, "custom_links": []}, {"id": "23963", "name": "Daniel Tjerrild Gamborg", "created_at": "2021-07-30T00:46:05+02:00", "email": null, "sis_user_id": null, "short_name": "Daniel Tjerrild Gamborg", "sortable_name": "Gamborg, Daniel Tjerrild", "integration_id": null, "login_id": null, "custom_links": []}, {"id": "23992", "name": "Casper Hedegaard Hansen", "created_at": "2021-07-30T00:46:05+02:00", "email": null, "sis_user_id": null, "short_name": "Casper Hedegaard Hansen", "sortable_name": "Hansen, Casper Hedegaard", "integration_id": null, "login_id": null, "custom_links": []}, {"id": "36266", "name": "Morten Gedsted Hansen", "created_at": "2021-07-30T00:46:05+02:00", "email": null, "sis_user_id": null, "short_name": "Morten Gedsted Hansen", "sortable_name": "Hansen, Morten Gedsted", "integration_id": null, "login_id": null, "custom_links": []}, {"id": "23980", "name": "Anders Husted", "created_at": "2020-08-03T10:45:00+02:00", "email": null, "sis_user_id": null, "short_name": "Anders Husted", "sortable_name": "Husted, Anders", "integration_id": null, "login_id": null, "custom_links": []}, {"id": "23531", "name": "Søren Bo Jørgensen", "created_at": "2021-07-30T00:46:05+02:00", "email": null, "sis_user_id": null, "short_name": "Søren Bo Jørgensen", "sortable_name": "Jørgensen, Søren Bo", "integration_id": null, "login_id": null, "custom_links": []}]
```

# 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. At the top, there are two navigation links: "ALLE UDDANNELSER" and "UDDANNELSER UD FRA INTERESSE". Below this, a grid of 12 program cards, each featuring a student's face and the program name. The programs are: AUTOMATIONSTEKNOLOG (Automation Technologist), BYGGEKOORDINATOR (Construction Coordinator), BYGGETEKNIKER (Building Technician), DATAMATIKER (Computer Science), DESIGNTEKNOLOG (Design Technologist), ENTREPRENØRSKAB OG DESIGN (Entrepreneurship and Design), EL-INSTALLATOR (Electrical Installer), ENERGITEKNOLOG (Energy Technologist), IT-TEKNOLOG (IT Technologist), KORT- OG LANDMÅLING (Cartography and Land Surveying), MULTIMEDIEDESIGNER (Multimedia Designer), and PRODUKTIONSTEKNOLOG (Production Technologist). Each card has a small arrow icon pointing to the right.

# Objects with properties in arrays

The screenshot shows a web browser window for the Business Academy Aarhus website ([baaa.dk/programmes/](https://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

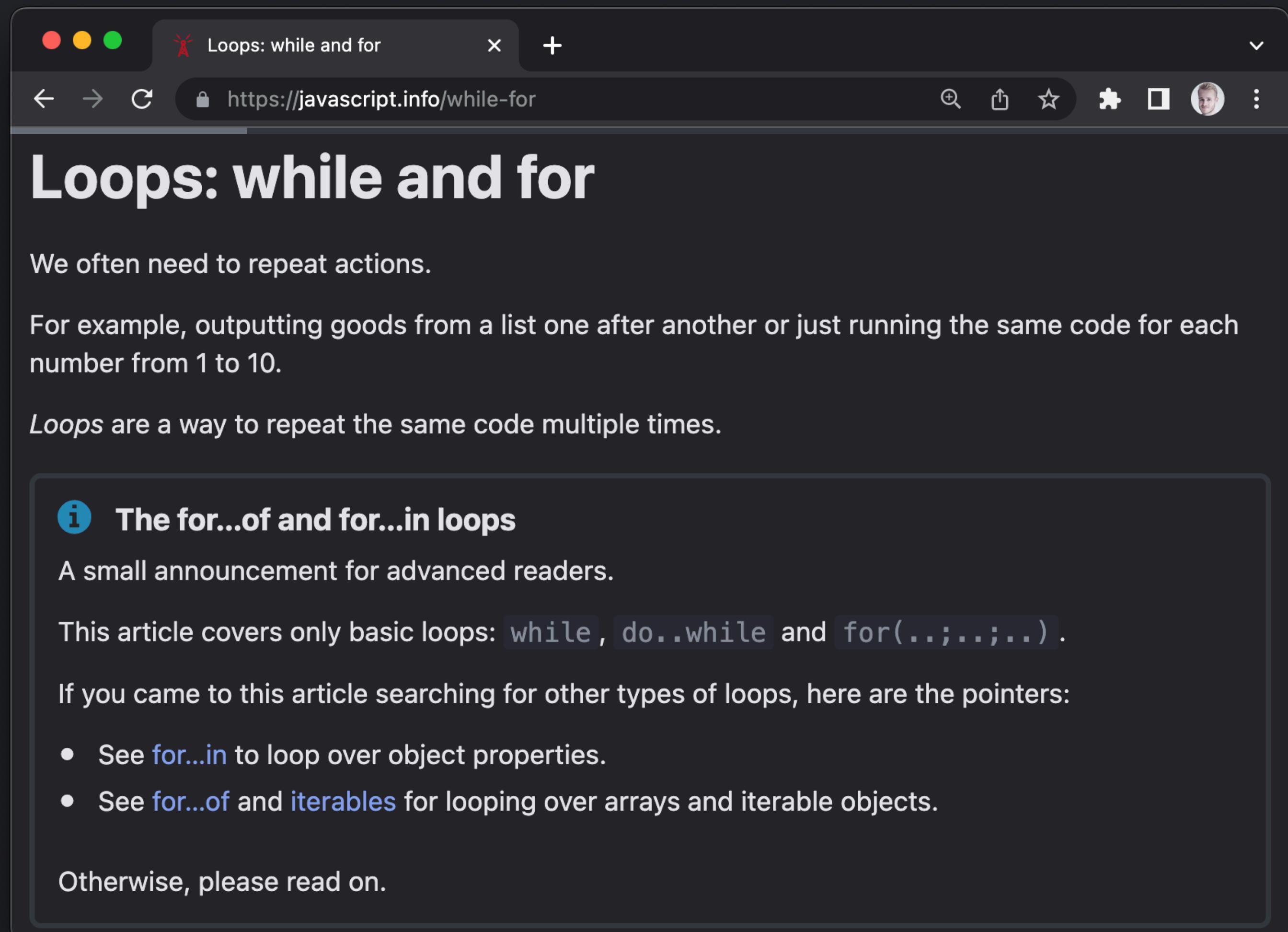
# Arrays

## 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 displays the following text:

## 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://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 }
```



# 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

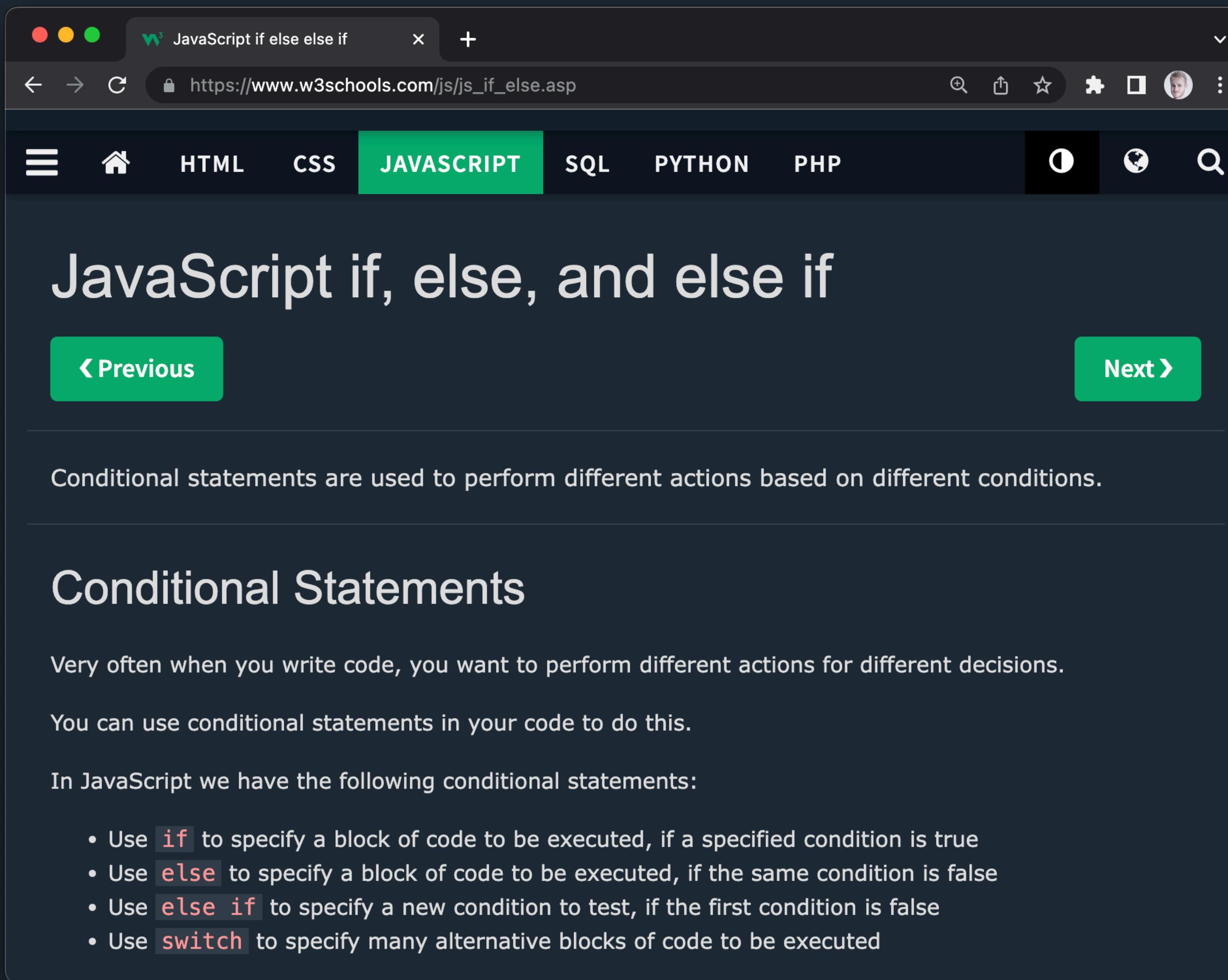
... LOOP THROUGH AN ARRAY AND ADD A CONDITION

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

# Conditional Statements

Perform different action based on  
different conditions

# Conditional Statements



The screenshot shows a web browser window with the title bar "JavaScript if else else if" and the URL "https://www.w3schools.com/js/js\_if\_else.asp". The browser interface includes standard controls like back, forward, and search. A navigation bar at the top has tabs for "HTML", "CSS", "JAVASCRIPT" (which is highlighted in green), "SQL", "PYTHON", and "PHP". Below the navigation bar, the main content area features a large heading "JavaScript if, else, and else if". Underneath the heading are two green buttons: "Previous" on the left and "Next" on the right. The main text content begins with a statement: "Conditional statements are used to perform different actions based on different conditions." This is followed by a section titled "Conditional Statements" with a descriptive paragraph: "Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this. In JavaScript we have the following conditional statements:". A bulleted list then provides four ways to use conditional statements in JavaScript:

- Use `if` to specify a block of code to be executed, if a specified condition is true
- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false
- Use `switch` to specify many alternative blocks of code to be executed

[https://www.w3schools.com/js/js\\_if\\_else.asp](https://www.w3schools.com/js/js_if_else.asp)

**if** to specify a block of code to be executed, if a specified condition is true

```
if (condition) {  
    // block of code to be executed if the condition is true  
}
```

**else** to specify a block of code to be executed, if the same condition is false

```
if (condition) {  
    // block of code to be executed if the condition is true  
} else {  
    // block of code to be executed if the condition is false  
}
```

**else if** to specify a new condition to test, if the first condition is false

```
if (condition1) {  
    // block of code to be executed if condition1 is true  
} else if (condition2) {  
    // block of code to be executed if the condition1 is false  
    // and condition2 is true  
} else {  
    // block of code to be executed if the condition1 is false  
    // and condition2 is false  
}
```

# Ternary Operator

## 1.5. Ternary Operator

What: Simplifies the conditional operator `if` / `else`.

Why: It takes (too) long to write conditionals with `if` / `else`. Ternary Operator is often used in JSX expressions as an efficient implementation of conditional rendering.

Syntax: `condition ? <expression if true> : <expression if false>`

JS: `const result = condition ? value1 : value2;`

```
// condition ? <expression if true> : <expression if false>

const age = 43;

const status = age > 18 ? "adult" : "child";

//same as
let status;
if (age > 18) {
  status = "adult";
} else {
  status = "child";
}
```

# The use of

;  
:  
()  
[]  
{ }  
\$  
"  
'  
. .  
==  
====  
< >  
?

# `==` VS `== =`

`== =` IS MORE STRICT

`== =` compares both types and values  
`==` compares values

```
"32" == 32 is true
"32" == = 32 is false
```

# Array Methods to know

.push (...)

.map (...)

.filter (...)

.find (...)

.sort (...)

Computer science student



Senior developer, 10+ years experience



# 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

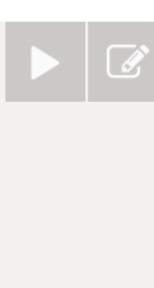
The `find` method looks for a single (first) element that makes the function return `true`. If there may be many, we can use `arr.filter(fn)`.

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>

# Array.map(...)

...iterate over an array and modify each element.

Array.map(...) calls a callback function for each element in the array.

```
const persons = [
  { firstname: "Birgitte", lastname: "Iversen" },
  { firstname: "Lykke", lastname: "Dahlen" },
  { firstname: "Rasmus", lastname: "Cederdorff" }
];

const mapped = persons.map(person => {
  return {
    name: `${person.firstname} ${person.lastname}`
  };
});

console.log(mapped);
```

▼ (3) [{...}, {...}, {...}] *i*

- 0: {name: 'Birgitte Iversen'}
- 1: {name: 'Lykke Dahlen'}
- 2: {name: 'Rasmus Cederdorff'}

length: 3

# From Vanilla JS to React Developer

## 1.4.1. Array.map(...)

What: `.map()` allow us to run a function for each item in an array and transform the items. At the end a new array will be returned.

Why: When “thinking in React” one of the most useful is the `.map()` array method. It makes it easy to transform an array of objects into HTML. And in general, to transform and manipulate with objects in an array.

```
const numbers = [2, 4, 6, 8];

const newNumbers = numbers.map(number => number * 2);

//same as
const newNumbers = numbers.map(function(number) {
  return number * 2;
});
```

```
const teachers = ["Rasmus", "Morten", "Dan"];

const result = teachers.map(teacher => <p>{teacher}</p>)
```

```
const persons = [
{
  firstName: "Birgitte",
  lastName: "Iversen"
}]
```

100 seconds of

JS

# ARRAY MAP



# Array.map(...)

- Project template: array-map
- Use .map to map over the persons array and concatenate firstName and lastName to a new property called name.
- console.log() the result
- Add a property called birthYear for each person object. Use map to map over the array and add the property birthYear dynamically based on birthDate (hint: use String.split(...) or .slice(...)).

```
const persons = [  
  {  
    firstName: "Jane",  
    lastName: "Doe",  
    birthDate: "1992-03-04"  
  },  
  {  
    firstName: "Jens",  
    lastName: "Jensen",  
    birthDate: "1992-07-04"  
  },  
  {  
    firstName: "Birgitte",  
    lastName: "Iversen",  
    birthDate: "1990-10-04"  
  },  
  {  
    firstName: "Lykke",  
    lastName: "Dahlen",  
    birthDate: "1987-06-04"  
  },  
  {  
    firstName: "Kasper",  
    lastName: "Topp",  
    birthDate: "1989-03-07"  
  }];  
  
const result = persons.map(person => {  
  console.log(person);  
  // manipulate and return value  
});
```

# 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));
```

# `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`

## 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](mailto:bki@baaa.dk)



Michael Hvidtfeldt

Senior Lecturer  
[mhv@baaa.dk](mailto:mhv@baaa.dk)



Rasmus Cederdorff

Lecturer  
[race@baaa.dk](mailto: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`

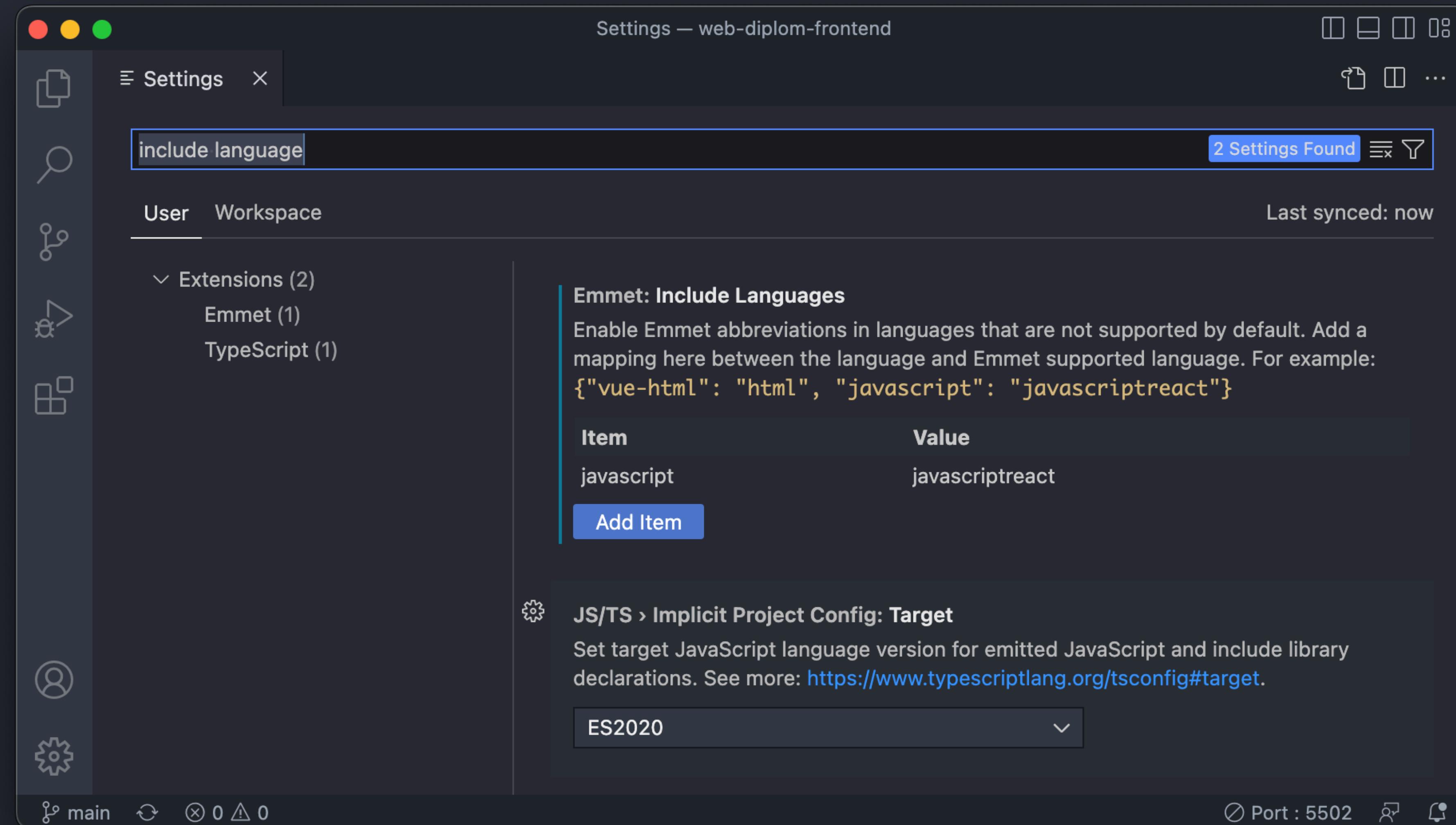
<https://marketplace.visualstudio.com/items?itemName=hjb2012.vscode-es6-string-html>

```
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



# 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: `{}`

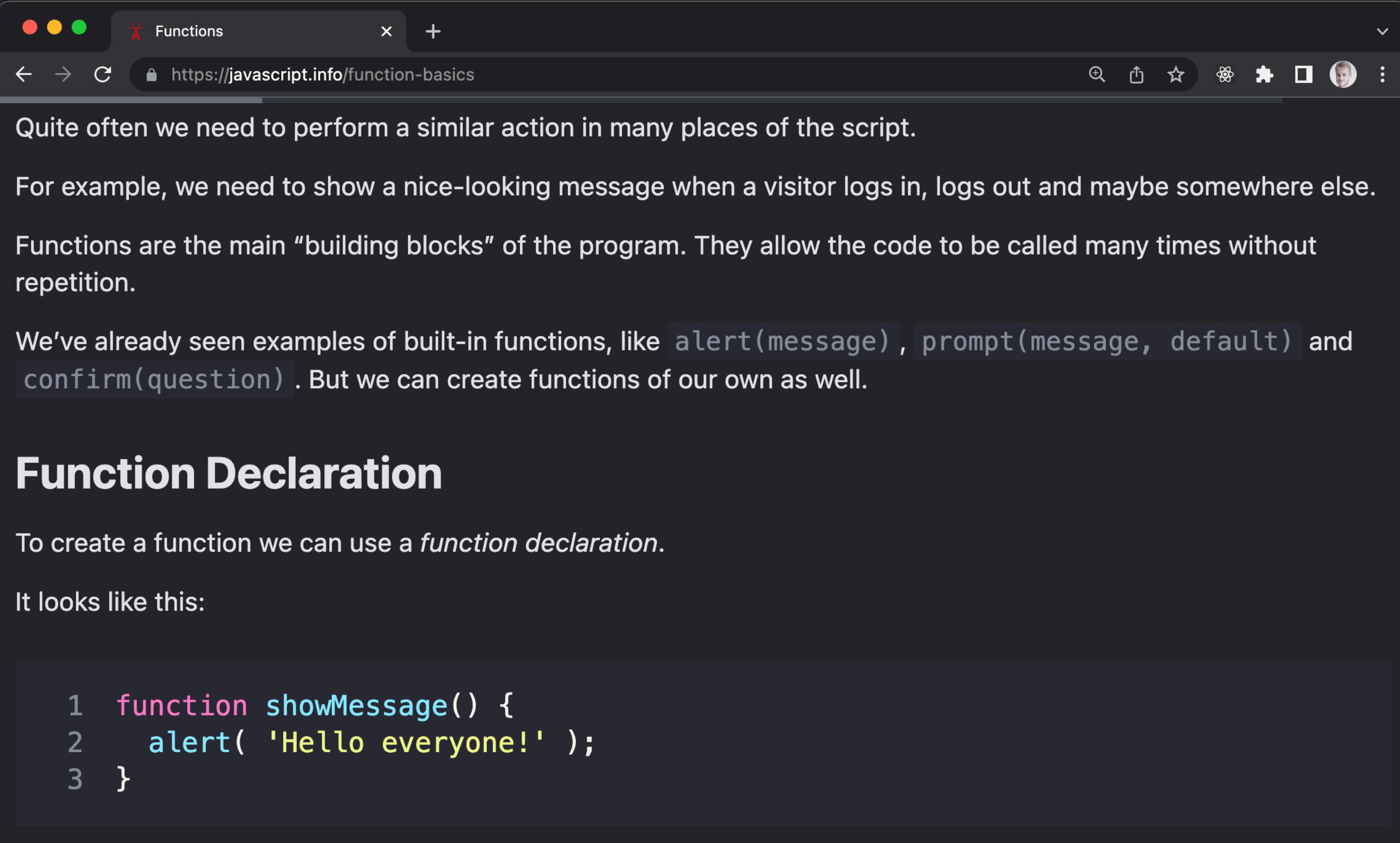
```
function name(parameter1, parameter2, parameter3) {  
    // code to be executed  
}
```

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)

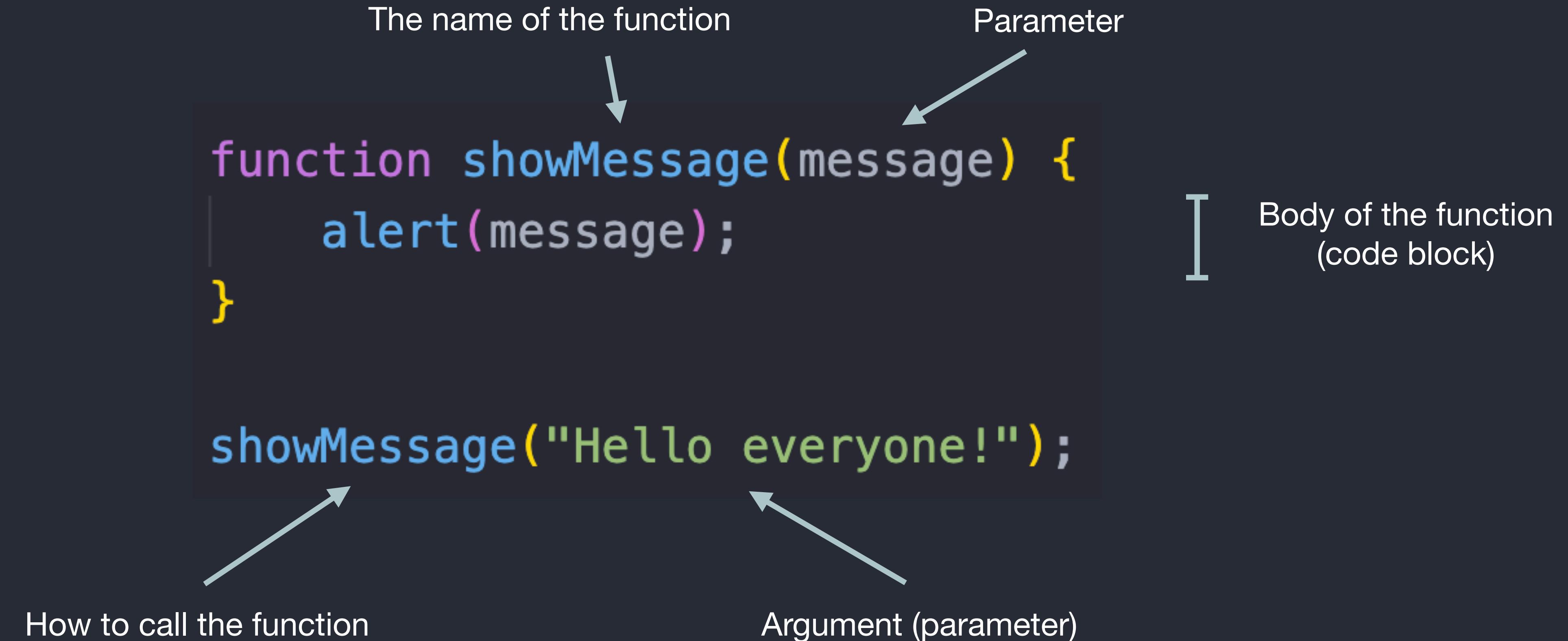
**showMessage();**

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](mailto:bki@baaa.dk)



Michael Hvidtfeldt

Senior Lecturer  
[mhv@baaa.dk](mailto:mhv@baaa.dk)



Rasmus Cederdorff

Lecturer  
[race@baaa.dk](mailto:race@baaa.dk)

# Global Variables

VARIABLES OUTSIDE A FUNCTION (AND SCOPES)  
ARE GLOBAL 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
```

## 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
```

The function has full access to the outer variable. It can modify it as well.

```
let userName = 'John';

function showMessage() {
    userName = "Bob"; // (1) changed the outer variable

    let message = 'Hello, ' + userName;
    alert(message);
}

alert( userName ); // John before the function call

showMessage();

alert( userName ); // Bob, the value was modified by the function
```

## 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.

Global  
Variable

```
let _movies = [];

// fetch all movies from WP
async function getMovies() {
  let response = await fetch("https://movie-api.cederdorff.com/wp-json/wp/v2/posts");
  let data = await response.json();
  console.log(data);
  _movies = data;
  appendMovies(data);
  showLoader(false);
}

getMovies();
```

ARRAY  
movies

```
// append movies to the DOM
function appendMovies(movies) {
  let htmlTemplate = '';
  for (let movie of movies) {
    htmlTemplate += `
      <article>
        <h2>${movie.title.rendered} (${movie.acf.year})</h2>
        
        <p>${movie.acf.description}</p>
        <iframe src="${movie.acf.trailer}"></iframe>
      </article>
    `;
  }
  document.querySelector('#movies-container').innerHTML = htmlTemplate;
}
```

Inline variable

Store the current object  
from the array

function  
argument

variable inside  
the function

```
let _movies = [];

// fetch all movies from WP
async function getMovies() {
  let response = await fetch("https://movie-api.cederdorff.com/wp-json/wp/v2/posts");
  let data = await response.json();
  console.log(data);
  _movies = data;
  appendMovies(data);
  showLoader(false);
}

getMovies();
```

Inline variable

Store the current object  
from the array

```
// append movies to the DOM
function appendMovies(movies) {
  let htmlTemplate = "";
  for (let movie of movies) {
    htmlTemplate += `
      <article>
        <h2>${movie.title.rendered} (${movie.acf.year})</h2>
        
        <p>${movie.excerpt.rendered}</p>
      </article>
    `;
```

function  
argument

variable inside  
the function

```
// ===== Product functionality ===== //
/*
global variables: _products, _selectedProductId
*/
let _products = [];
let _selectedProductId;

/*
Fetches json data from the file products.json
*/
async function fetchData() {
    const response = await fetch('json/products.json');
    const data = await response.json();
    _products = data;
    console.log(_products);
    appendProducts(_products);
    showLoader(false);
}

fetchData();

function appendProducts(products) {
    let htmlTemplate = "";
    for (let product of products) {
        htmlTemplate += /*html*/
            <article class="${product.status}">
                <article onclick="showDetailView(${product.id})">
```

# [JavaScript.info/function-basics#local-variables](https://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
```

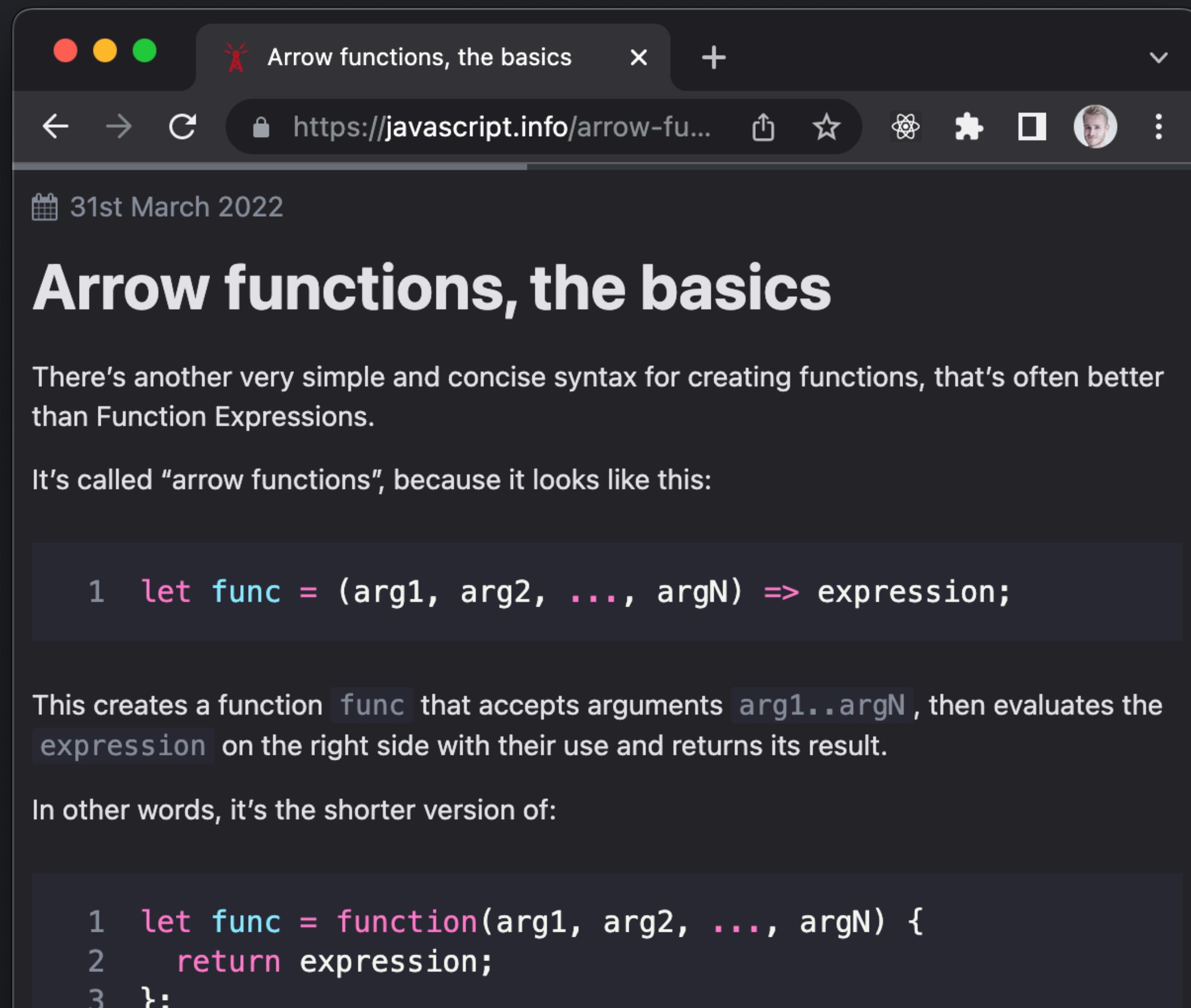
# Function called by another function

Use the name of  
the function to  
call / execute

```
let users = [...  
];  
  
function appendUsers(users) {  
  let htmlTemplate = "";  
  for (const user of users) {  
    console.log(user);  
    htmlTemplate += /*html*/`  
      <article>  
          
        <h2>${user.name}</h2>  
        <a href="mailto:${user.email}">${user.email}</a>  
        <p>Role: ${user.enrollment_type}</p>  
      </article>  
    `;  
  }  
  document.querySelector("#users").innerHTML = htmlTemplate;  
}  
  
function initApp() {  
  appendUsers(users);  
}  
  
initApp();
```

users refers to  
our declared  
variable, users,  
(global variable)

# Javascript.info/Arrow-Functions-Basics



The screenshot shows a dark-themed web browser window. The title bar says "Arrow functions, the basics". The address bar shows the URL "https://javascript.info/arrow-fu...". Below the address bar, there's a date "31st March 2022". The main content area has a large heading "Arrow functions, the basics". Below the heading, a text block says: "There's another very simple and concise syntax for creating functions, that's often better than Function Expressions. It's called "arrow functions", because it looks like this:". A code block shows the syntax: "1 let func = (arg1, arg2, ..., argN) => expression;". Below this, a text block explains: "This creates a function `func` that accepts arguments `arg1..argN`, then evaluates the `expression` on the right side with their use and returns its result. In other words, it's the shorter version of:". Another code block shows the equivalent function expression: "1 let func = function(arg1, arg2, ..., argN) { 2 return expression; 3 };".

```
function orderByBrand() {  
  _products.sort((product1, product2) => {  
    return product1.brand.localeCompare(product2.brand);  
  });  
  appendProducts(_products);  
}
```

```
function orderByModel() {  
  _products.sort((product1, product2) => {  
    return product1.model.localeCompare(product2.model);  
  });  
  appendProducts(_products);  
}
```

```
function orderByPrice() {  
  _products.sort((product1, product2) => {  
    return product1.price - product2.price;  
  });  
  appendProducts(_products);  
}
```

.SORT & ARROW FUNCTIONS  
INSIDE FUNCTIONS DECLARATIONS

```
function search(event) {  
    const searchValue = event.target.value.toLowerCase();  
    const results = [];  
    for (const person of persons) {  
        const name = person.name.toLowerCase();  
        if (name.includes(searchValue)) {  
            results.push(person);  
        }  
    }  
    displayPersons(results);  
}
```

FOR OF LOOP

```
function search(event) {  
    const searchValue = event.target.value.toLowerCase();  
    const results = persons.filter(person => person.name.toLowerCase().includes(searchValue));  
    displayPersons(results);  
}
```

.FILTER & ARROW FUNCTION

# Destructuring

## 1.7. Destructuring

What: Extract what we need from an existing array or object.

Why: Makes it easy to extract only what we need from an existing array or object.

### Objects

```
const teacher = {
  name: "Morten",
  email: "moab@eaaa.dk"
};

//choose name and email
const { name, email } = teacher;
//name: "Morten"
//email: "moab@eaaa.dk"
```

### Arrays

```
const teachers = ["Rasmus", "Morten", "Dan"];

//choose two names
const [mrFrontend, mrWebComponents] = teachers;
//mrFrontend: "Rasmus"
//mrWebComponents: "Morten"
```

# Spread Operator

## 1.6. Spread Operator

What: Allow us to expand and copy existing objects and arrays into another object or array.

Why: Efficient expansion and a simplified syntax to concatenate strings, objects and arrays.

Spread an array:

```
const numbersOne = [1, 2, 3];
const numbersTwo = [4, 5, 6];
const numbersCombined = [...numbersOne, ...numbersTwo];
```

```
const array1 = [1,2,3];
const array2 = [...array1, 4, 5]; // [1,2,3,4,5]
```

Copy arrays without reference using the spread operator:

```
const ar1 = [1,2,3];
const copy = [...ar1]; // [1,2,3]
```

Split strings:

# Modules, Import & Export

## 1.8. Modules, import & export

What: A module is just a script file. One is script or one file is one module. We use the keywords `export` and `import` to tell what we would like to export from one script and import in another script.

Why: Gives structure and easier to maintain a codebase. As our app grows, we want to split our code in multiple files instead of one long script file. Also, it makes it easier to work with components.

Export and import a const:

```
// 📄 user.js
```

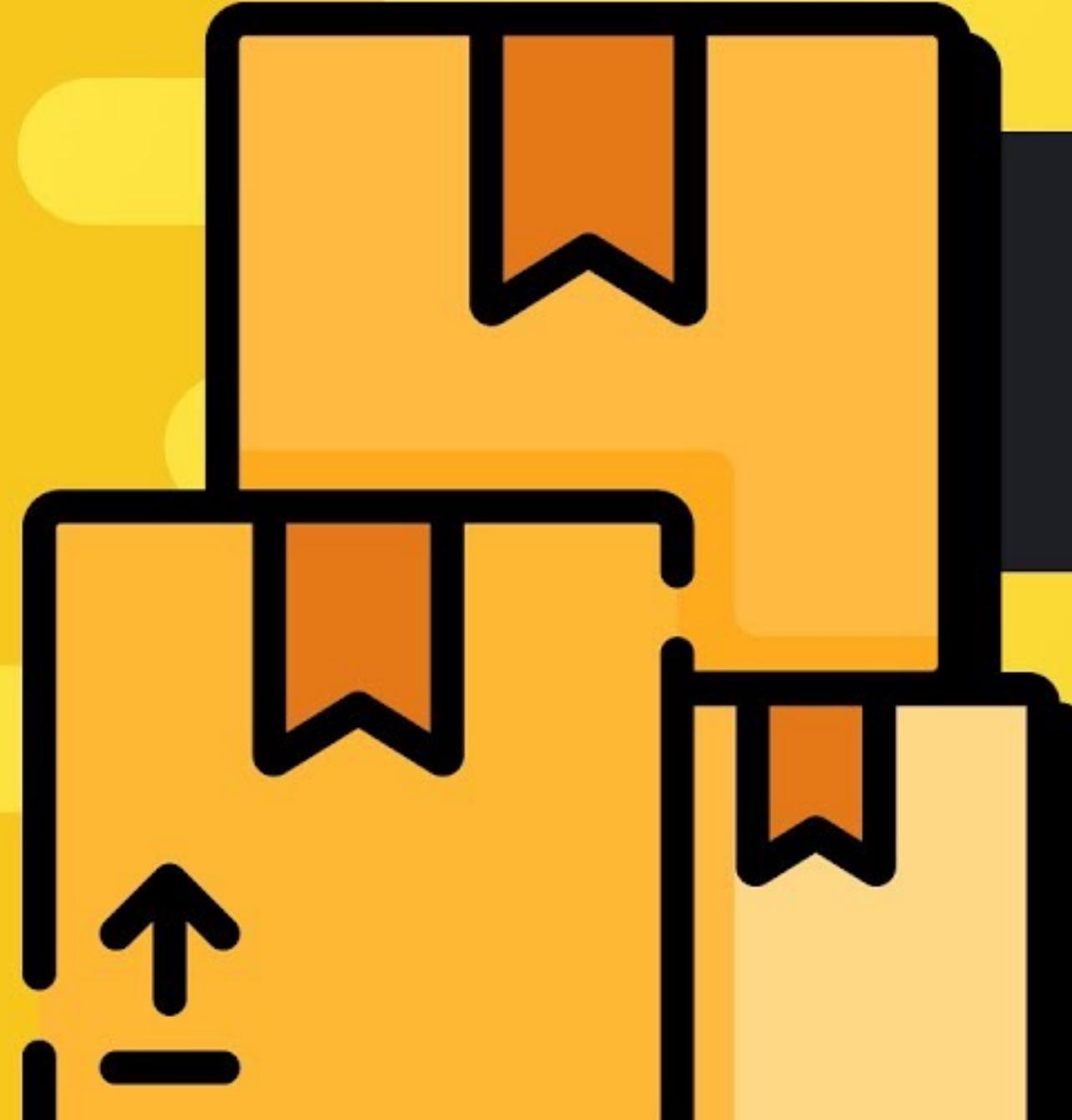
```
export const user = {  
  name: "Jane"  
  age: 29  
};
```

```
// 📄 app.js
```

```
import { user } from "user.js";  
  
console.log(user);
```

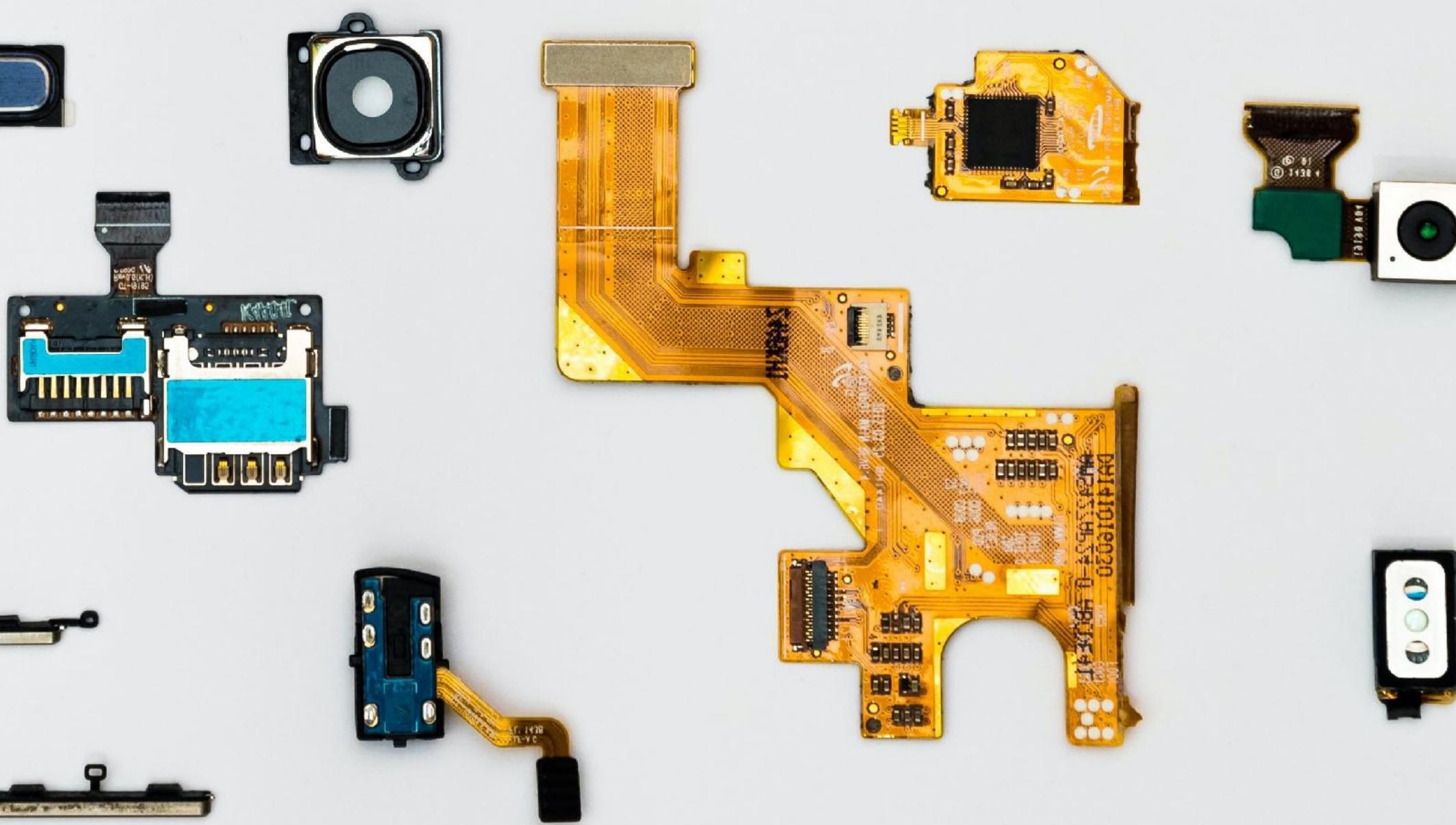
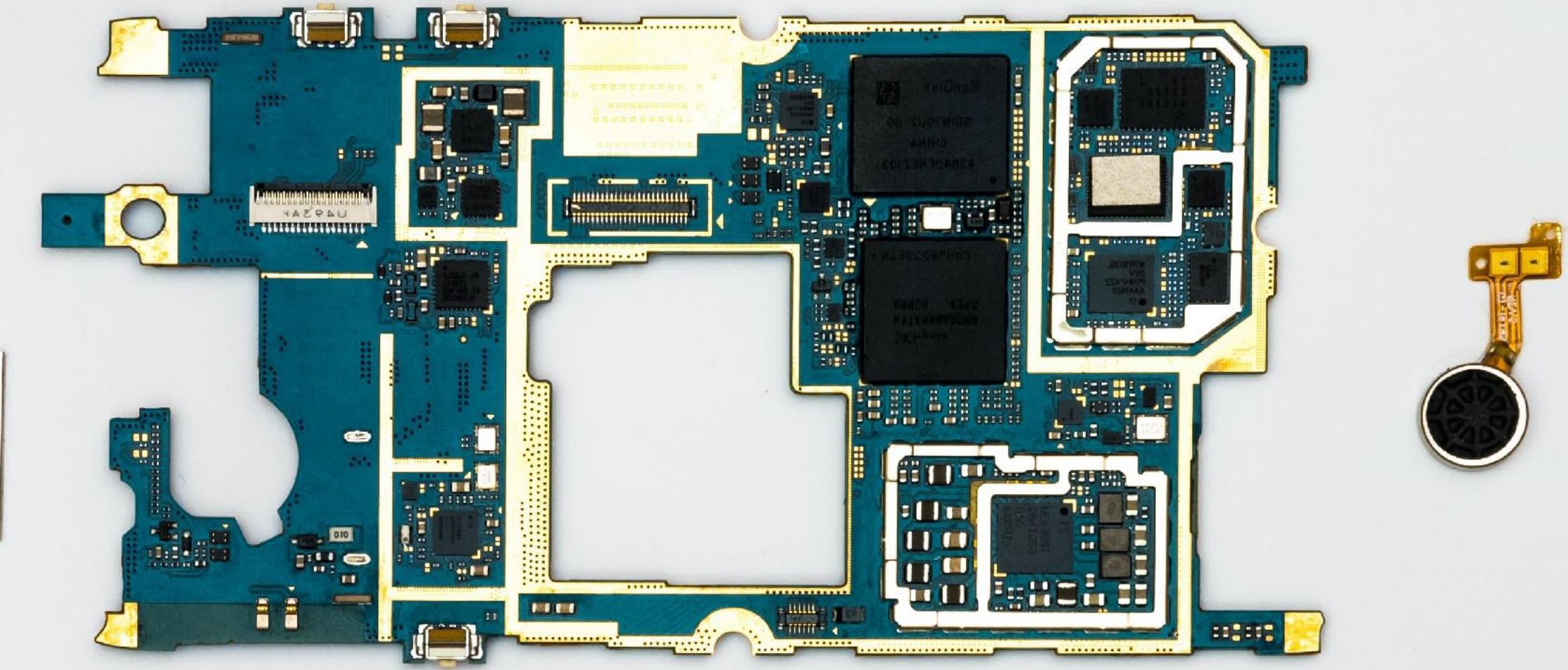
100 seconds of

JS



MODULES

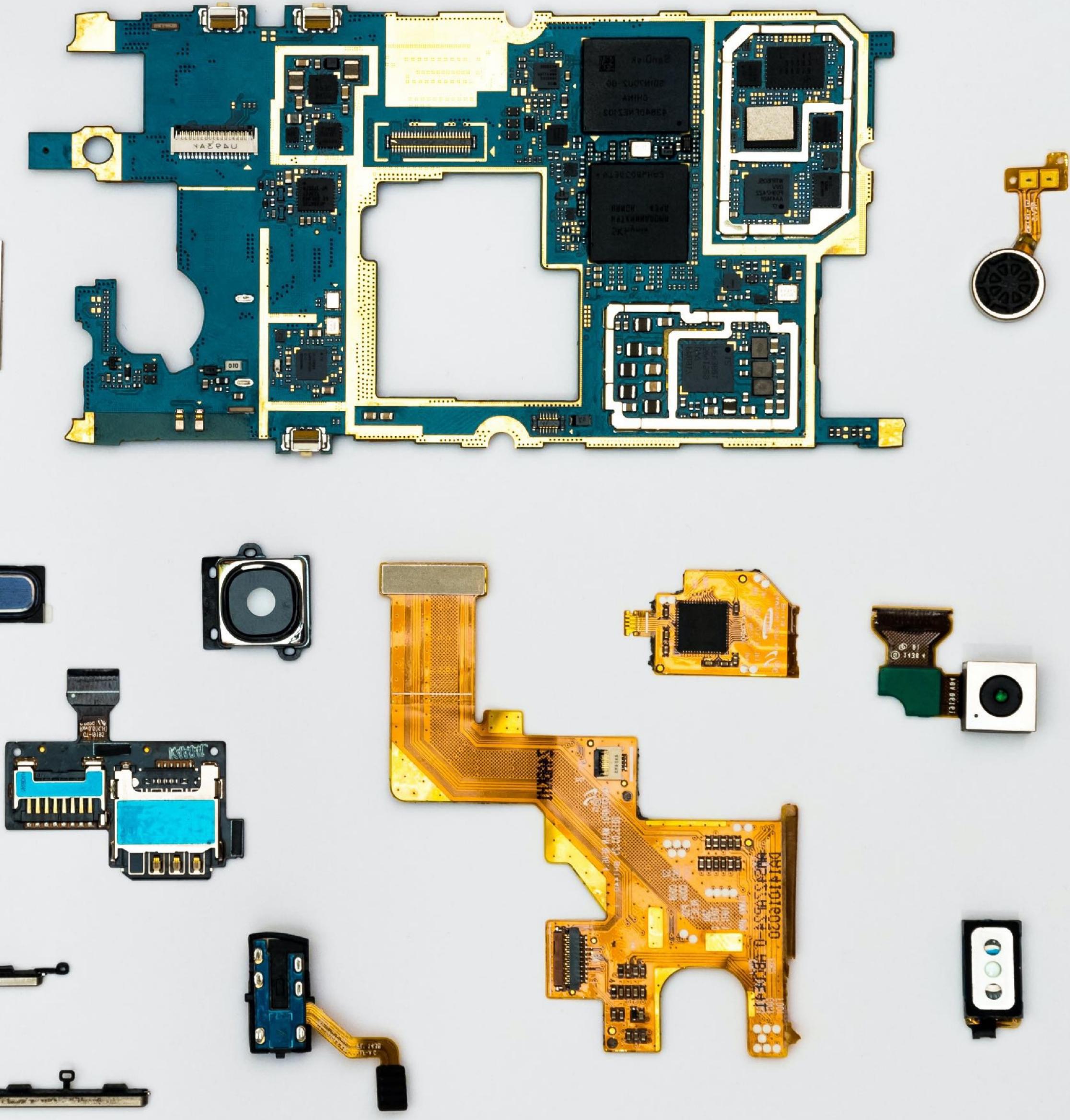
<https://www.youtube.com/watch?v=qgRUr-YUk1Q>



# Modules

A module is just a file  
(or a script).

One script is one  
module.



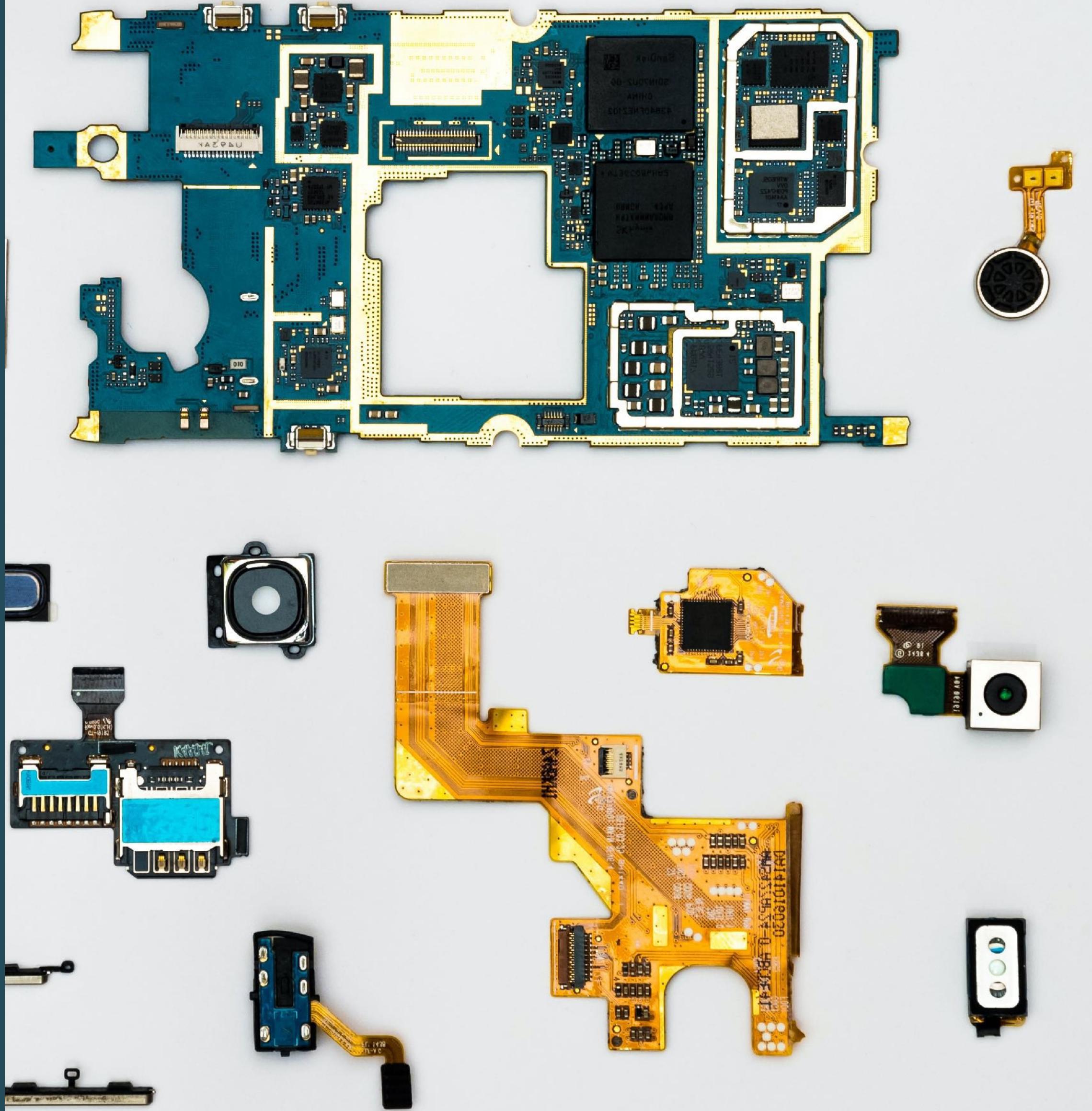
# Modules

As our application grows bigger, we want to split it into multiple files, so-called “modules”.

A module may contain a class or a library of functions for a specific purpose.

# Modules

Modules can load each other and use special directives export and import to interchange functionality, call functions of one module from another one ...



**e x p o r t => i m p o r t**

**export** keyword labels variables and functions that should be accessible from outside the current module.

**import** allows to import functionality from other modules.

```
// └─ sayHi.js
export function sayHi(user) {
  alert(`Hello, ${user}!`);
}
```

A MODULE (SCRIPT)

```
// └─ main.js
import {sayHi} from './sayHi.js';
alert(sayHi); // function...
sayHi('John'); // Hello, John!
```

ANOTHER MODULE

(SCRIPT)

## A MODULE (SCRIPT)

```
// └─ rest-service.js
async function getPosts() {
  const response = await fetch(` ${endpoint}/posts.json`); // fetch request, (G
  const data = await response.json(); // parse JSON to JavaScript
  return data; // convert object of object to array of objects
}

export { getPosts };
```

## ANOTHER MODULE (SCRIPT)

```
// └─ app.js
import { getPosts } from "./rest-service.js";

async function updatePostsGrid() {
  const posts = await getPosts(); // get posts from rest endpoint and save in
  console.log(posts);
}
```

```
// 📁 user.js
class User { // just add "default"
  constructor(name) {
    this.name = name;
  }
}
export default User;
```

```
// 📁 main.js
import User from './user.js'; // not {User}, just User
new User('John');
```

### 3. Function Reference imported

```
app.js — post-app
js > JS app.js > createPostClicked > response
1 import { compareTitle, compareBody } from "./helpers.js";
2 import { getPosts, createPost, getUser, deletePost, updatePost } from "./rest-service.js";
3
4 let posts;
5
6 window.addEventListener("load", initApp);
7
8 function initApp() {
9     updatePostsGrid(); // update the grid of posts - get and show all posts
10    // event listeners
11    document.querySelector("#btn-create-post").addEventListener("click", showCreatePostDialog);
12    document.querySelector("#form-create-post").addEventListener("submit", createPostClicked);
13    document.querySelector("#form-update-post").addEventListener("submit", updatePostClicked);
14    document.querySelector("#form-delete-post").addEventListener("submit", deletePostClicked);
15    document.querySelector("#form-delete-post .btn-cancel").addEventListener("click", deleteCancelClicked);
16    document.querySelector("#select-sort-by").addEventListener("change", sortByChanged);
17    document.querySelector("#input-search").addEventListener("keyup", inputSearchChanged);
18    document.querySelector("#input-search").addEventListener("search", inputSearchChanged);
19}
20
21 // ===== events ===== //
22
23 function showCreatePostDialog() {
24     document.querySelector("#dialog-create-post").showModal(); // show create dialog
25 }
26
27 async function createPostClicked(event) {
28     const form = event.target; // or "this"
29     // extract the values from inputs from the form
30     const title = form.title.value;
31     const body = form.body.value;
32     const image = form.image.value;
33     const response = await createPost(title, body, image); // use values to create a new post
34     // check if response is ok - if the response is successful
35     if (response.ok) {
36         console.log("New post successfully added to Firebase 🎉");
37         form.reset(); // reset the form (clears inputs)
38         updatePostsGrid();
39     }
40 }
41
42 async function updatePostClicked(event) {
43     const form = event.target; // or "this"
44     // extract the values from inputs in the form
45     const title = form.title.value;
46     const body = form.body.value;
47     const image = form.image.value;
48     const id = form.getAttribute("data-id"); // get id of the post to update - saved in data-id
49     const response = await updatePost(id, title, body, image); // call updatePost with arguments
50
51     if (response.ok) {
52         console.log("Post successfully updated in Firebase 🎉");
53     }
54 }
```

### 4. Function is called

```
app.js — post-app
js > JS rest-service.js > ...
1 import { postData } from "./helpers.js";
2 const endpoint = "https://post-rest-api-default.firebaseio.com";
3
4 // Get all posts - HTTP Method: GET
5 async function getPosts() {
6     const response = await fetch(`${endpoint}/posts.json`); // fetch request, (GET)
7     const data = await response.json(); // parse JSON to JavaScript
8     return postData(data); // convert object of object to array of objects
9 }
10
11 // Create a new post - HTTP Method: POST
12 async function createPost(title, body, image) {
13     const newPost = { title, body, image, uid: "fTs84KR0Yw5pRZEWQq2Z" }; // create new post object
14     const json = JSON.stringify(newPost); // convert the JS object to JSON string
15     // POST fetch request with JSON in the body
16     const response = await fetch(`${endpoint}/posts.json`, {
17         method: "POST",
18         body: json
19     });
20     return response;
21 }
22
23 // Update an existing post - HTTP Method: DELETE
24 async function deletePost(id) {
25     const response = await fetch(`${endpoint}/posts/${id}.json`, {
26         method: "DELETE"
27     });
28     return response;
29 }
30
31 // Delete an existing post - HTTP Method: PUT
32 async function updatePost(id, title, body, image) {
33     const postToUpdate = { title, body, image }; // post update to update
34     const json = JSON.stringify(postToUpdate); // convert the JS object to JSON string
35     // PUT fetch request with JSON in the body. Calls the specific element in resource
36     const response = await fetch(`${endpoint}/posts/${id}.json`, {
37         method: "PATCH",
38         body: json
39     });
40     return response;
41 }
42
43 // get user data by given id
44 async function getUser(id) {
45     const response = await fetch(`${endpoint}/users/${id}.json`);
46     const user = await response.json();
47     return user;
48 }
49
50 export { getPosts, createPost, updatePost, deletePost, getUser };
51
```

### 1. Function Declaration

### 2. Function Reference exported

```
// 📁 user.js
export default class User { // just add "default"
  constructor(name) {
    this.name = name;
  }
}
```

```
// 📁 main.js
import User from './user.js'; // not {User}, just User
new User('John');
```

With export default you can skip {...} when you import

```
JS main.js
1 import User from "./user.js";
2
3 const users = [
4   new User("Birgitte", "bki@mail.dk", "1966-01-14", "https://www.eaaa")
5   new User("Martin", "mnor@mail.dk", "1989-05-02", "https://media-ex")
6   new User("Rasmus", "race@mail.dk", "1990-09-15", "https://www.eaaa")
7 ];
8
9 console.log(users);
10
11 for (const user of users) {
12   document.querySelector("#content").innerHTML += user.getHtmlTemplate();
13 }
```

```
JS user.js
1 export default class User {
2   constructor(name, mail, birthDate, img) {
3     this.name = name;
4     this.mail = mail;
5     this.birthDate = birthDate;
6     this.img = img;
7   }
8
9   log() {
10     console.log(`Name: ${this.name}, Mail: ${this.mail}, Birth date: ${this.birthDate}, Image Url: ${this.img}`);
11   }
12
13   getAge() {
14     const birthDate = new Date(this.birthDate);
15     const today = new Date();
16     const diff = new Date(today - birthDate);
17     return diff.getFullYear() - 1970;
18   }
19
20   getHtmlTemplate() {
21     const template = /*html*/
22       <article>
23         
24         <h2>${this.name}</h2>
25         <a href="mailto:${this.mail}">${this.mail}</a>
26         <p>Birth date: ${this.birthDate}</p>
27         <p>Age: ${this.getAge()} years old</p>
28       </article>
29   }
30
31 }
```

## Reference to a variable is exported and imported elsewhere

```
app.js — web-diplom-frontend
```

```
JS app.js  X
modules-array-persons > JS app.js > ...
1 import { persons } from "./data.js";
2
3 console.log(persons);
4
5 export function displayPersons(listOfPersons) {
6     let html = "";
7     //loop through all persons and create an article with content fo
8     for (const person of listOfPersons) {
9         html += /*html* `

10            <article>
11                
12                <h2>${person.name}</h2>
13                <p>${person.title}</p>
14                <a href="mailto:${person.mail}">${person.mail}</a>
15            </article>
16        `;
17    }
18    document.querySelector("#content").innerHTML = html; // set grid
19 }
20
21 displayPersons(persons);
22
23 function savePerson(event) {
24     event.preventDefault(); // prevent form refreshing page
25     const form = event.target; // save reference to form in varaiabl
26
27     // create new person object based on input values
28     const newPerson = {
29         name: form.name.value,
30         mail: form.mail.value,
31         title: form.title.value,
32         img: form.url.value
33     };
34 }
```

```
JS data.js  X
modules-array-persons > JS data.js > ...
1 export const persons = [
2     {
3         name: "Birgitte Kirk Iversen",
4         mail: "bki@mail.dk",
5         title: "Senior Lecturer",
6         img: "https://www.eaaa.dk/media/u4gorzsd/birgitte-kirk-ivers
7     },
8     {
9         name: "Martin Aagaard Nøhr",
10        mail: "mnor@mail.dk",
11        title: "Lecturer",
12        img: "https://www.eaaa.dk/media/oayjq02h/martin-n%C3%B8hr.jp
13    },
14    {
15        name: "Rasmus Cederdorff",
16        mail: "race@mail.dk",
17        title: "Senior Lecturer",
18        img: "https://www.eaaa.dk/media/devlvgj/rasmus-cederdorff.j
19    },
20    {
21        name: "Dan Okkels Brendstrup",
22        mail: "dob@mail.dk",
23        title: "Lecturer",
24        img: "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstr
25    },
26    {
27        name: "Anne Kirketerp",
28        mail: "anki@mail.dk",
29        title: "Head of Department",
30        img: "https://www.baaa.dk/media/5buh1xeo/anne-kirketerp.jpg?
31    }
32];
33 ];
```

The diagram illustrates the flow of code between three files: `app.js`, `data.js`, and `helpers.js`.

**app.js:**

```
// ===== imports ===== //
import { persons } from './data.js';
import { search, sortPersons, showLecturers } from "./helpers.js";

console.log(persons);

export function displayPersons(listOfPersons) {
    let html = "";
    //loop through all persons and create an article with content for each
    for (const person of listOfPersons) {
        html += /*html*/
            `


                <h2>${person.name}</h2>
                <p>${person.title}</p>
                <a href="mailto:${person.mail}">${person.mail}</a>
            </article>
    `;
    }
    document.querySelector("#content").innerHTML = html; // set grid
}

displayPersons(persons);

function savePerson(event) {
    event.preventDefault(); // prevent form refreshing page
    const form = event.target; // save reference to form in variable

    // create new person object based on input values
    const newPerson = {
        name: form.name.value,
        mail: form.mail.value,
        title: form.title.value,
        img: form.url.value
    }
}


```

**data.js:**

```
// ===== imports ===== //
import { persons } from "./data.js";
import { displayPersons } from "./app.js";

// ===== helper functions ===== //
export function search(event) {
    const searchValue = event.target.value.toLowerCase();
    const results = persons.filter(person => person.name.toLowerCase().includes(searchValue));
    displayPersons(results);
}

export function sortPersons(event) {
    const option = event.target.value;
    if (option === "name") {
        console.log("sort by name");
        persons.sort((person1, person2) => person1.name.localeCompare(person2.name));
    } else if (option === "title") {
        console.log("sort by title");
        persons.sort((person1, person2) => person1.title.localeCompare(person2.title));
    }
    displayPersons(persons);
}

export function showLecturers(event) {
    const showLecturers = event.target.checked;
    console.log(showLecturers);

    if (showLecturers) {
        displayPersons(persons);
    } else {
        const results = persons.filter(person => person.title === "S");
        displayPersons(results);
    }
}
```

**helpers.js:**

```
// ===== imports ===== //
import { persons } from "./data.js";
import { displayPersons } from "./app.js";

// ===== helper functions ===== //
export function search(event) {
    const searchValue = event.target.value.toLowerCase();
    const results = persons.filter(person => person.name.toLowerCase().includes(searchValue));
    displayPersons(results);
}

export function sortPersons(event) {
    const option = event.target.value;
    if (option === "name") {
        console.log("sort by name");
        persons.sort((person1, person2) => person1.name.localeCompare(person2.name));
    } else if (option === "title") {
        console.log("sort by title");
        persons.sort((person1, person2) => person1.title.localeCompare(person2.title));
    }
    displayPersons(persons);
}

export function showLecturers(event) {
    const showLecturers = event.target.checked;
    console.log(showLecturers);

    if (showLecturers) {
        displayPersons(persons);
    } else {
        const results = persons.filter(person => person.title === "S");
        displayPersons(results);
    }
}
```

Annotations in the code:

- Arrows point from the `persons` import in `app.js` to the `persons` export in `data.js`.
- Arrows point from the `displayPersons` import in `app.js` to the `displayPersons` export in both `data.js` and `helpers.js`.
- An arrow points from the `search` import in `app.js` to the `search` export in `data.js`.
- An arrow points from the `sortPersons` import in `app.js` to the `sortPersons` export in `data.js`.
- An arrow points from the `showLecturers` import in `app.js` to the `showLecturers` export in `data.js`.

# Tell the browser it's a module

```
<script src="js/main.js" type="module"></script>
```

# Forms

Collect user inputs

In HTML we use the `<form>` element  
with `<input>` fields to create forms

The screenshot shows a web browser window with the title "HTML Forms" from w3schools.com. The page content includes a brief description of what an HTML form is used for, followed by an "Example" section. The example shows a simple form with two input fields: "First name:" containing "John" and "Last name:" containing "Doe". Below the fields is a "Submit" button. A green "Try it Yourself »" button is also present. Further down, there's a section titled "The <form> Element" with a code snippet illustrating its structure.

An HTML form is used to collect user input. The user input is most often sent to a server for processing.

**Example**

First name:  
John

Last name:  
Doe

Submit

Try it Yourself »

**The `<form>` Element**

The HTML `<form>` element is used to create an HTML form for user input:

```
<form>
  .
  form elements
  .
</form>
```

# Inputs

Mostly used inside the  
`<form>` element

`<input>` fields can be displayed in  
different ways with different types.

See all types: [HTML Input Types](#)

The `<input>` Element

The HTML `<input>` element is the most used form element.

An `<input>` element can be displayed in many ways, depending on the `type` attribute.

Here are some examples:

Type	Description
<code>&lt;input type="text"&gt;</code>	Displays a single-line text input field
<code>&lt;input type="radio"&gt;</code>	Displays a radio button (for selecting one of many choices)
<code>&lt;input type="checkbox"&gt;</code>	Displays a checkbox (for selecting zero or more of many choices)
<code>&lt;input type="submit"&gt;</code>	Displays a submit button (for submitting the form)
<code>&lt;input type="button"&gt;</code>	Displays a clickable button

All the different input types are covered in this chapter: [HTML Input Types](#).

```
<form>
  <input type="text" name="name" placeholder="Type new name">
  <input type="email" name="mail" placeholder="Type new mail">
  <input type="text" name="title" placeholder="Type new title">
  <input type="url" name="url" placeholder="Paste url to image">
  <button>Save</button>
</form>
```

# Forms & Inputs

```
<form onsubmit="saveUser(event)">
  <h2>Create new user</h2>
  <input type="text" name="name" placeholder="Type new name">
  <input type="email" name="mail" placeholder="Type new mail">
  <input type="text" name="title" placeholder="Type new title">
  <input type="url" name="url" placeholder="Paste url to image">
  <button>Save</button>
</form>
```

```
function saveUser(event) {
  event.preventDefault(); // prevent form refreshing page
  const form = event.target; // save reference to form in variable

  // create new person object based on input values
  const newPerson = {
    name: form.name.value,
    mail: form.mail.value,
    title: form.title.value,
    img: form.url.value
  };

  persons.push(newPerson); // add new peron object to array (persons)
  displayPersons(); // make sure to reload all persons
  form.reset(); // reset (clear) input fields
}
```

HTML

JavaScript

# Forms & Inputs

```
<form onsubmit="saveUser(event)">  
  <h2>Create new user</h2>  
  <input type="text" name="name" placeholder="Type new name">  
  <input type="email" name="mail" placeholder="Type new mail">  
  <input type="text" name="title" placeholder="Type new title">  
  <input type="url" name="url" placeholder="Paste url to image">  
  <button>Save</button>  
</form>
```

```
function saveUser(event) {  
  event.preventDefault(); // prevent form refreshing page  
  const form = event.target; // save reference to form in variable  
  
  // create new person object based on input values  
  const newPerson = {  
    name: form.name.value,  
    mail: form.mail.value,  
    title: form.title.value,  
    img: form.url.value  
  };  
  
  persons.push(newPerson); // add new peron object to array (persons)  
  displayPersons(); // make sure to reload all persons  
  form.reset(); // reset (clear) input fields  
}
```

HTML

JavaScript

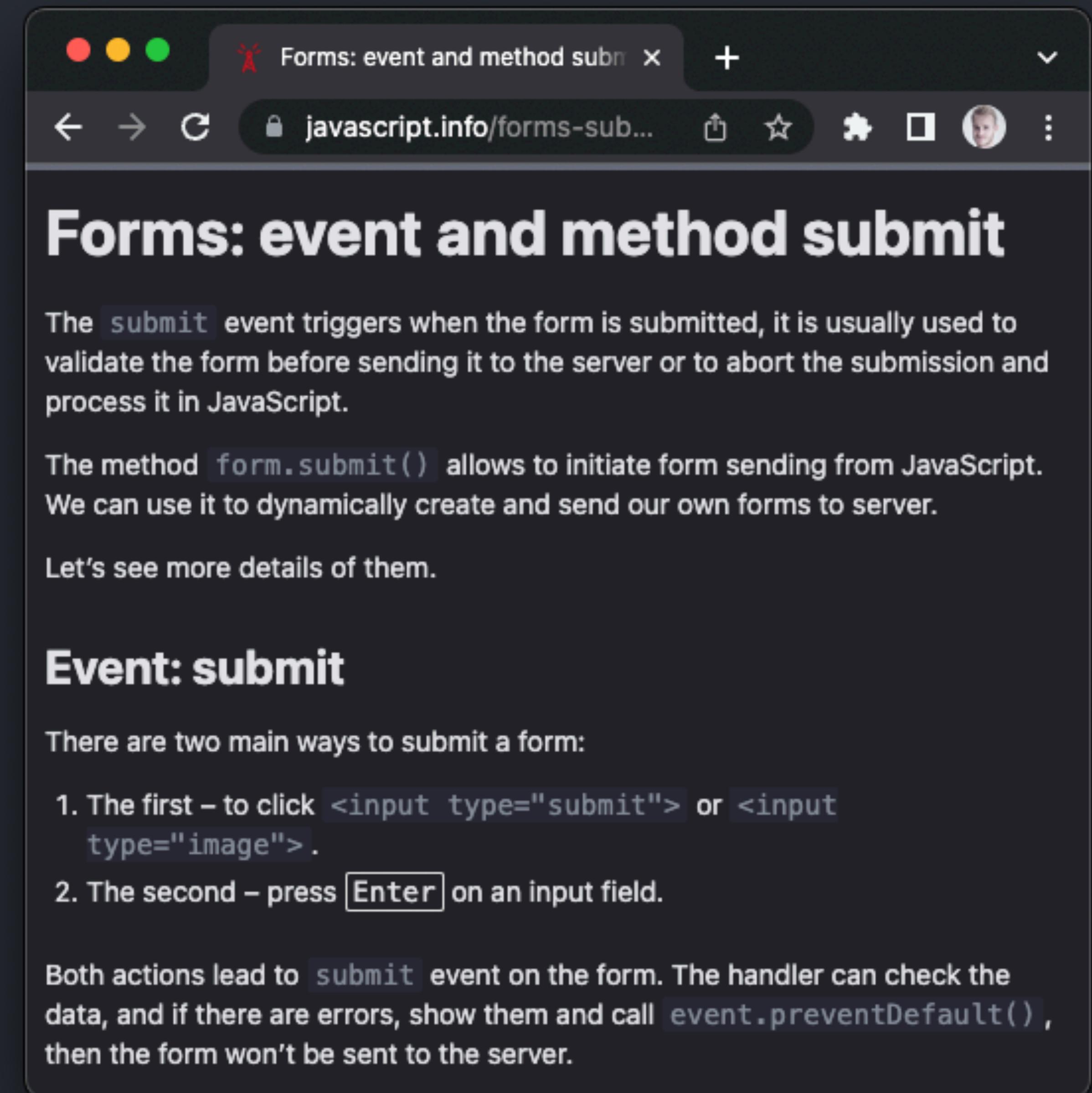
# Submit event

```
<form onsubmit="saveUser(event)">
  <h2>Create new user</h2>
  <input type="text" name="name" placeholder="Type new name">
  <input type="email" name="mail" placeholder="Type new mail">
  <input type="text" name="title" placeholder="Type new title">
  <input type="url" name="url" placeholder="Paste url to image">
  <button>Save</button>
</form>
```

```
function saveUser(event) {
  event.preventDefault(); // prevent form refreshing page
  const form = event.target; // save reference to form in variable

  // create new person object based on input values
  const newPerson = {
    name: form.name.value,
    mail: form.mail.value,
    title: form.title.value,
    img: form.url.value
  };

  persons.push(newPerson); // add new peron object to array (persons)
  displayPersons(); // make sure to reload all persons
  form.reset(); // reset (clear) input fields
}
```



The screenshot shows a web browser window with the title "Forms: event and method submit". The URL in the address bar is "javascript.info/forms-submit". The main content of the page is titled "Forms: event and method submit" and contains the following text:

The `submit` event triggers when the form is submitted, it is usually used to validate the form before sending it to the server or to abort the submission and process it in JavaScript.

The method `form.submit()` allows to initiate form sending from JavaScript. We can use it to dynamically create and send our own forms to server.

Let's see more details of them.

## Event: submit

There are two main ways to submit a form:

1. The first – to click `<input type="submit">` or `<input type="image">`.
2. The second – press `Enter` on an input field.

Both actions lead to `submit` event on the form. The handler can check the data, and if there are errors, show them and call `event.preventDefault()`, then the form won't be sent to the server.

# Prevent Default

```
<form onsubmit="saveUser(event)">
  <h2>Create new user</h2>
  <input type="text" name="name" placeholder="Type new name">
  <input type="email" name="mail" placeholder="Type new mail">
  <input type="text" name="title" placeholder="Type new title">
  <input type="url" name="url" placeholder="Paste url to image">
  <button>Save</button>
</form>

function saveUser(event) {
  event.preventDefault(); // prevent form refreshing page
  const form = event.target; // save reference to form in variable

  // create new person object based on input values
  const newPerson = {
    name: form.name.value,
    mail: form.mail.value,
    title: form.title.value,
    img: form.url.value
  };

  persons.push(newPerson); // add new peron object to array (persons)
  displayPersons(); // make sure to reload all persons
  form.reset(); // reset (clear) input fields
}
```

The screenshot shows a web browser window with the title "preventDefault() Event Method". The URL in the address bar is "w3schools.com/jsref/event\_preventdefault.asp". The browser interface includes standard controls like back, forward, and search, along with navigation links for "HTML" and "CSS".

## Definition and Usage

The preventDefault() method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur.

For example, this can be useful when:

- Clicking on a "Submit" button, prevent it from submitting a form
- Clicking on a link, prevent the link from following the URL

**Note:** Not all events are cancelable. Use the cancelable property to find out if an event is cancelable.

**Note:** The preventDefault() method does not prevent further propagation of an event through the DOM. Use the stopPropagation() method to handle this.

[https://www.w3schools.com/jsref/event\\_preventdefault.asp](https://www.w3schools.com/jsref/event_preventdefault.asp)

# Events & EventListener

Assign an onclick event to a button element:

```
<button onclick="displayDate()">Try it</button>
```

Assign an onclick event to a button element:

```
document.getElementById("myBtn").onclick = displayDate;
```

```
function displayDate() {  
    document.getElementById("demo").innerHTML = Date();  
}
```

Add an event listener that fires when a user clicks a button:

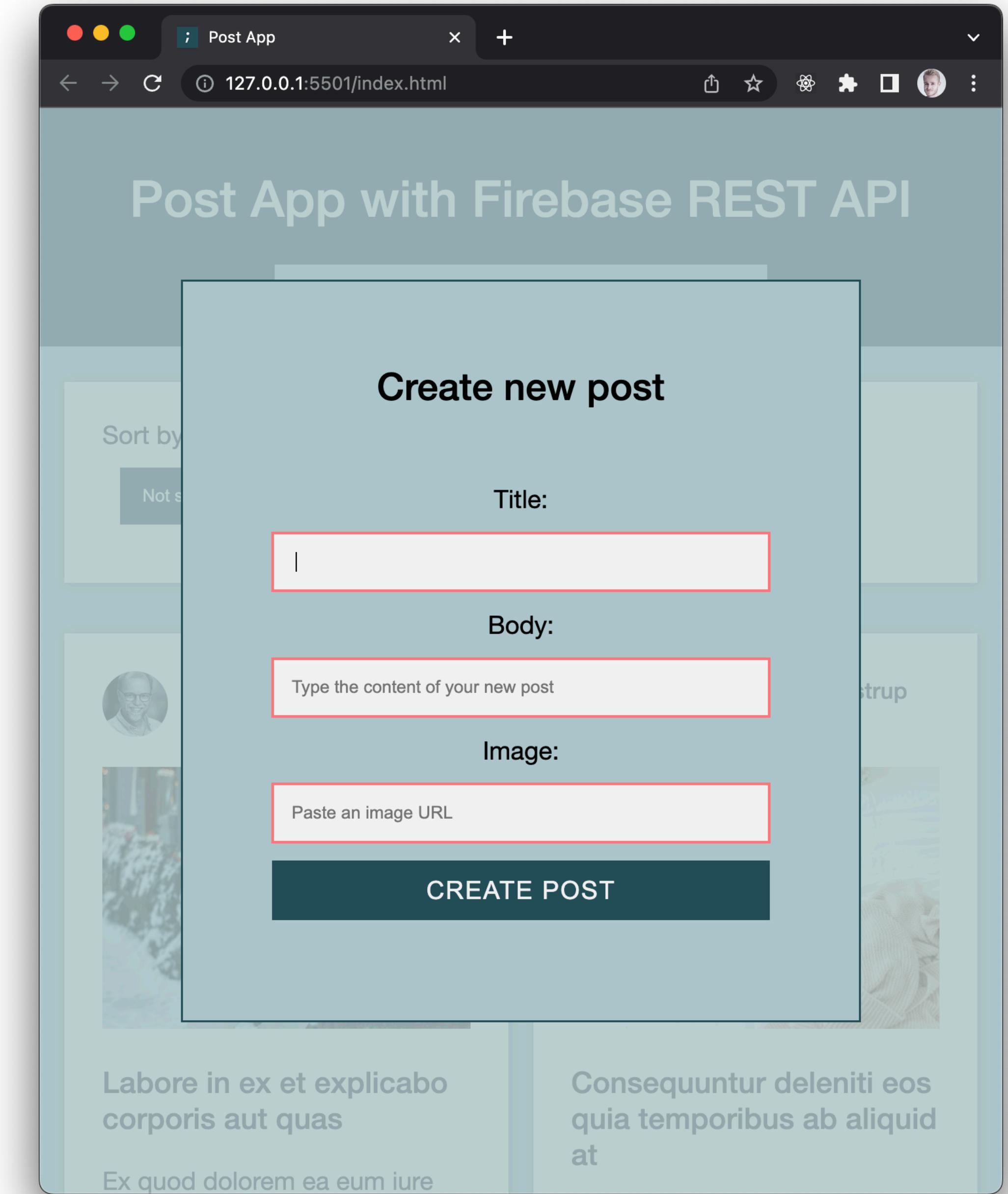
```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

The screenshot shows a web browser window with a dark theme. The title bar reads "HTML DOM Event Object" and the address bar shows "w3schools.com/jsref/dom\_obj\_event.asp". The main content area has a header "HTML DOM Events" with navigation buttons "Previous" and "Next". Below the header, a text block states: "HTML DOM events allow JavaScript to register different event handlers on elements in an HTML document. Events are normally used in combination with functions, and the function will not be executed before the event occurs (such as when a user clicks a button). For a tutorial about Events, read our [JavaScript Events Tutorial](#)." A table lists various HTML events with their descriptions and categories:

Event	Description	Belongs To
<a href="#">abort</a>	The event occurs when the loading of a media is aborted	<a href="#">UiEvent</a> , <a href="#">Event</a>
<a href="#">afterprint</a>	The event occurs when a page has started printing, or if the print dialogue box has been closed	<a href="#">Event</a>
<a href="#">animationend</a>	The event occurs when a CSS animation has completed	<a href="#">AnimationEvent</a>
<a href="#">animationiteration</a>	The event occurs when a CSS animation is repeated	<a href="#">AnimationEvent</a>
<a href="#">animationstart</a>	The event occurs when a CSS animation has started	<a href="#">AnimationEvent</a>
<a href="#">beforeprint</a>	The event occurs when a page is about to be printed	<a href="#">Event</a>
<a href="#">beforeunload</a>	The event occurs before the document is about to be unloaded	<a href="#">UiEvent</a> , <a href="#">Event</a>
<a href="#">blur</a>	The event occurs when an element loses focus	<a href="#">FocusEvent</a>
<a href="#">canplay</a>	The event occurs when the browser can start playing the media (when it has buffered enough to begin)	<a href="#">Event</a>
<a href="#">canplaythrough</a>	The event occurs when the browser can play through the media without stopping for buffering	<a href="#">Event</a>
<a href="#">change</a>	The event occurs when the content of a form element, the selection, or the checked state have changed (for <input>, <select>, and <textarea>)	<a href="#">Event</a>
<a href="#">click</a>	The event occurs when the user clicks on an element	<a href="#">MouseEvent</a>

# Validation

Make sure the user input is clean, correct, useful and ready to send to the server!



# Validation

There are different approaches.

**Server-side:** performed by a web server, after input has been sent to the server.

**Client-side:** performed by a web browser, before input is sent to a web server.

## Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

- has the user filled in all required fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input.

Validation can be defined by many different methods, and deployed in many different ways.

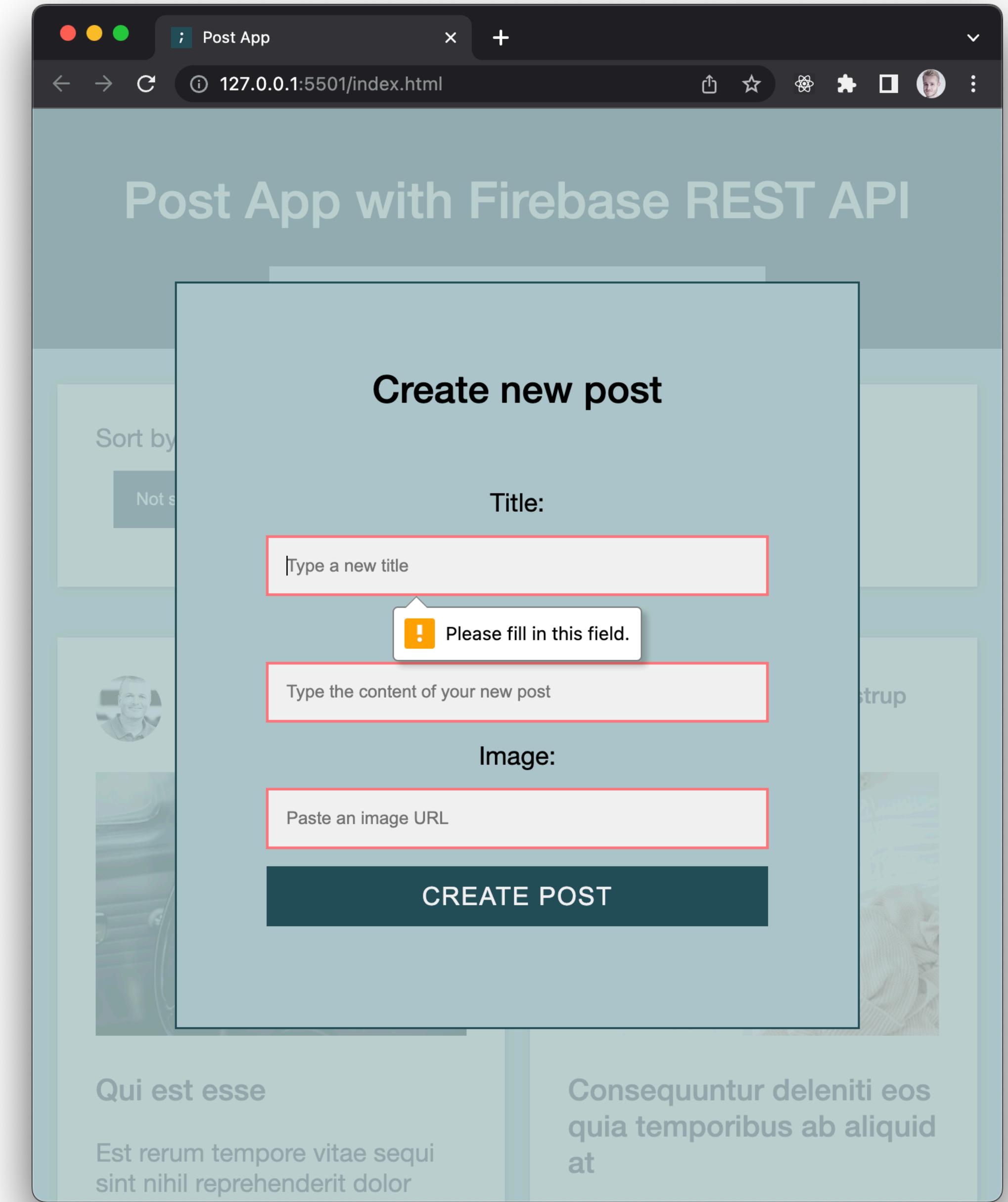
**Server side validation** is performed by a web server, after input has been sent to the server.

**Client side validation** is performed by a web browser, before input is sent to a web server.

# Client-side

Different types of client-side validation:

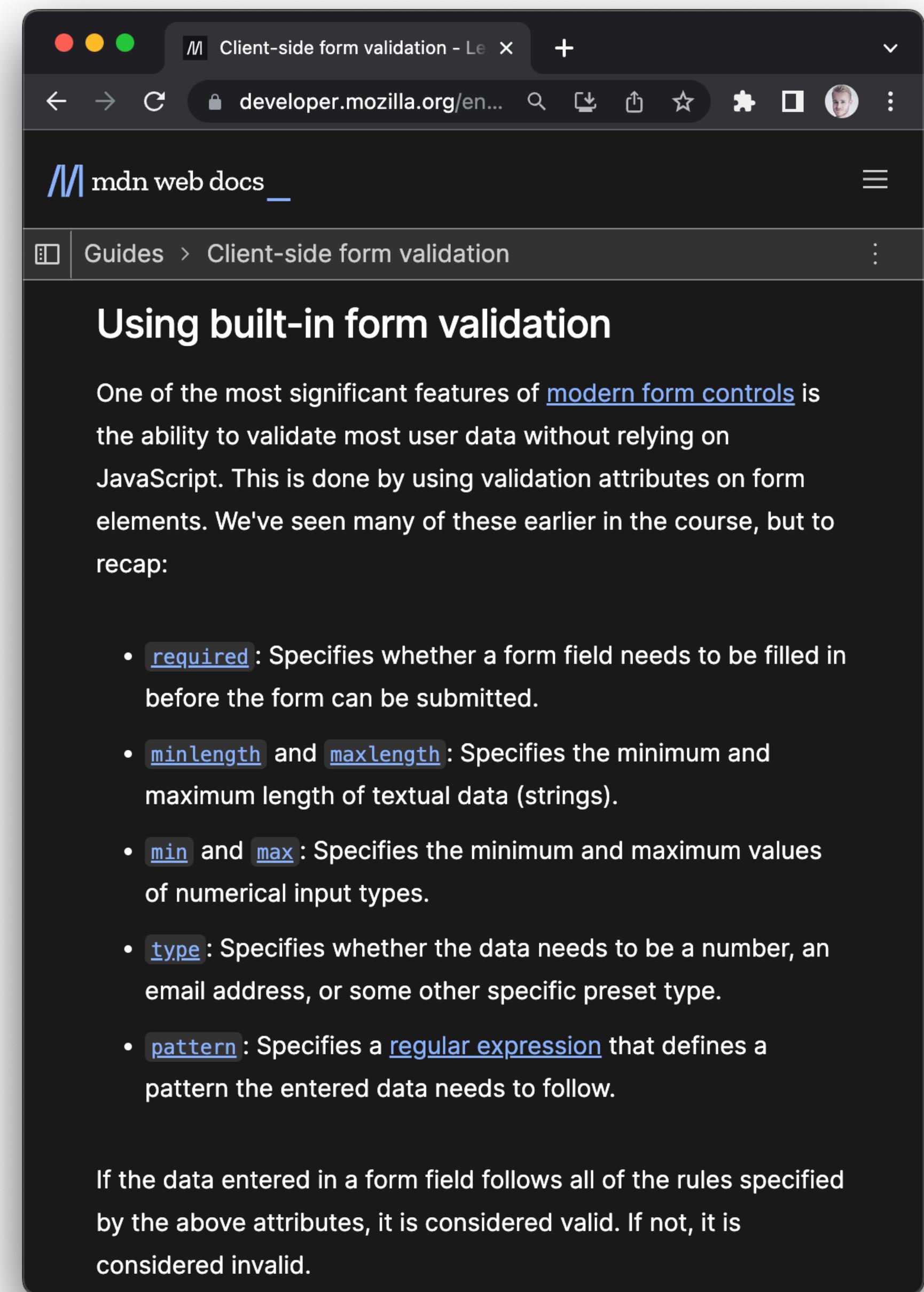
- Built-in form validation in HTML inputs
- JavaScript validation, coded and customisable in JavaScript.



# Built-in form validation in HTML inputs

You can use the built-in validation attributes to define validation criteria.

If the input follows the specified rules, it is considered valid. If not, it is considered invalid.



The screenshot shows a dark-themed web browser window. The address bar displays "developer.mozilla.org/en...". The page title is "Client-side form validation - MDN". The main content area has a heading "Using built-in form validation". Below the heading, there is a paragraph of text followed by a bulleted list of validation attributes. At the bottom of the page, there is a note about the validity of form fields based on the specified rules.

One of the most significant features of [modern form controls](#) is the ability to validate most user data without relying on JavaScript. This is done by using validation attributes on form elements. We've seen many of these earlier in the course, but to recap:

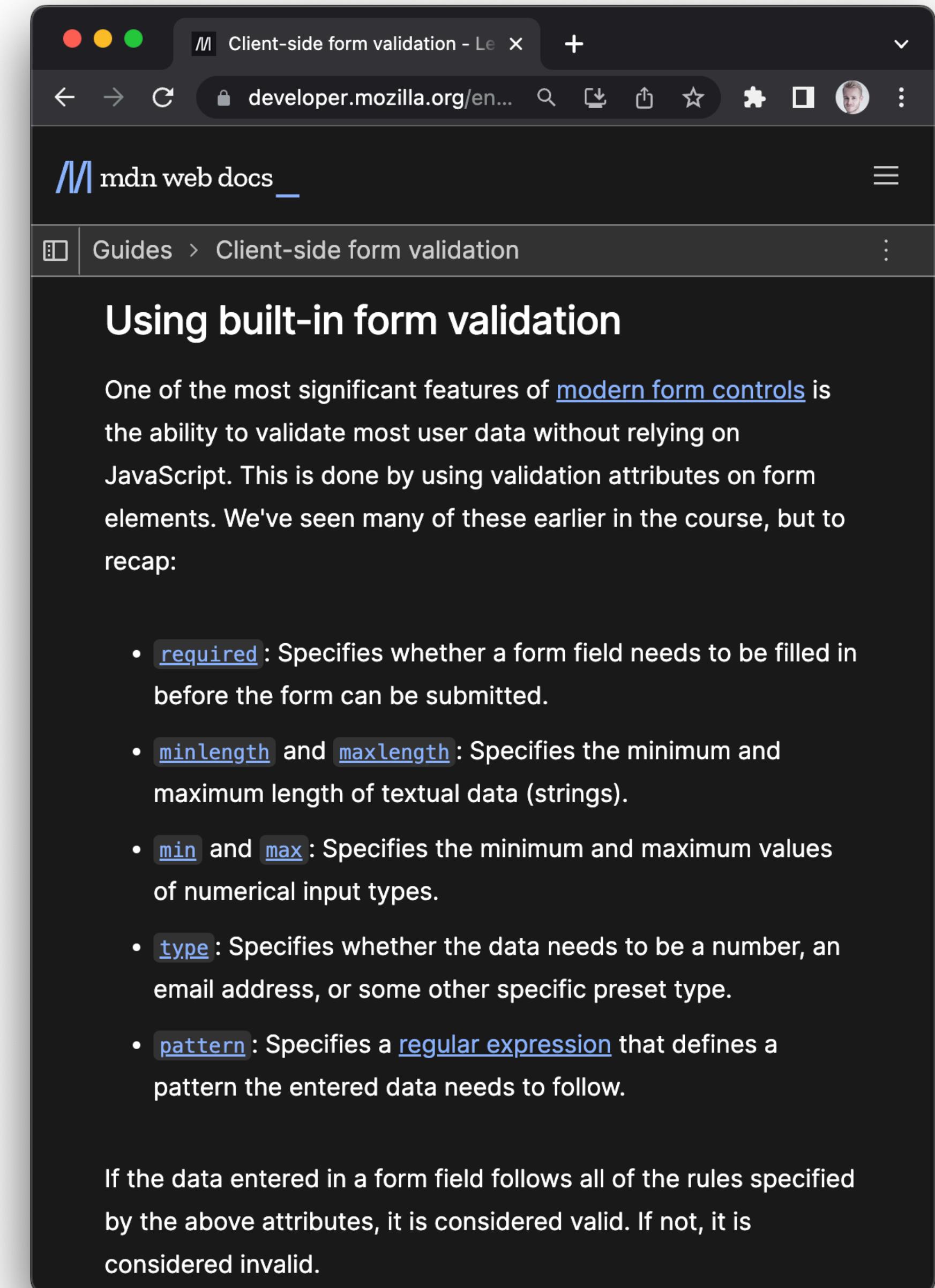
- `required`: Specifies whether a form field needs to be filled in before the form can be submitted.
- `minlength` and `maxlength`: Specifies the minimum and maximum length of textual data (strings).
- `min` and `max`: Specifies the minimum and maximum values of numerical input types.
- `type`: Specifies whether the data needs to be a number, an email address, or some other specific preset type.
- `pattern`: Specifies a [regular expression](#) that defines a pattern the entered data needs to follow.

If the data entered in a form field follows all of the rules specified by the above attributes, it is considered valid. If not, it is considered invalid.

[https://developer.mozilla.org/en-US/docs/Learn/Forms/  
Form\\_validation#using\\_builtin\\_form\\_validation](https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation#using_builtin_form_validation)

# Built-in form validation in HTML inputs

- Use input types (text, number, url, file, etc..)
- Use required, only if required
- Use minlength and maxlength - min and max if number
- If the above doesn't fit your criteria, use pattern and regex



The screenshot shows a dark-themed web browser window. The address bar displays "developer.mozilla.org/en...". The main content area is titled "Using built-in form validation". It contains text about the ability to validate user data using validation attributes like `required`, `minlength`, `maxlength`, `min`, `max`, `type`, and `pattern`. Below this, a bulleted list details these attributes. At the bottom, a note states that if all rules are followed, the data is valid; otherwise, it is invalid.

One of the most significant features of [modern form controls](#) is the ability to validate most user data without relying on JavaScript. This is done by using validation attributes on form elements. We've seen many of these earlier in the course, but to recap:

- `required`: Specifies whether a form field needs to be filled in before the form can be submitted.
- `minlength` and `maxlength`: Specifies the minimum and maximum length of textual data (strings).
- `min` and `max`: Specifies the minimum and maximum values of numerical input types.
- `type`: Specifies whether the data needs to be a number, an email address, or some other specific preset type.
- `pattern`: Specifies a [regular expression](#) that defines a pattern the entered data needs to follow.

If the data entered in a form field follows all of the rules specified by the above attributes, it is considered valid. If not, it is considered invalid.

[https://developer.mozilla.org/en-US/docs/Learn/Forms/  
Form\\_validation#using\\_builtin\\_form\\_validation](https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation#using_builtin_form_validation)

```

<!-- dialog - create new post -->
<dialog id="dialog-create-post">
  <h2>Create new post</h2>
  <form id="form-create-post" method="dialog">
    <label for="title">Title:</label>
    <input type="text" id="title" name="title" placeholder="Type a new title"
      minlength="3"
      maxlength="100"
      required>
    <label for="body">Body:</label>
    <input type="text" id="body" name="body" placeholder="Type the content of your post"
      minlength="5"
      required>
    <label for="image">Image:</label>
    <input type="text" id="image" name="image" placeholder="Paste an image URL"
      pattern="^https?://[^\s]+(\.[^\s]+)*$"
      required>
    <button>Create Post</button>
  </form>
</dialog>

```

```

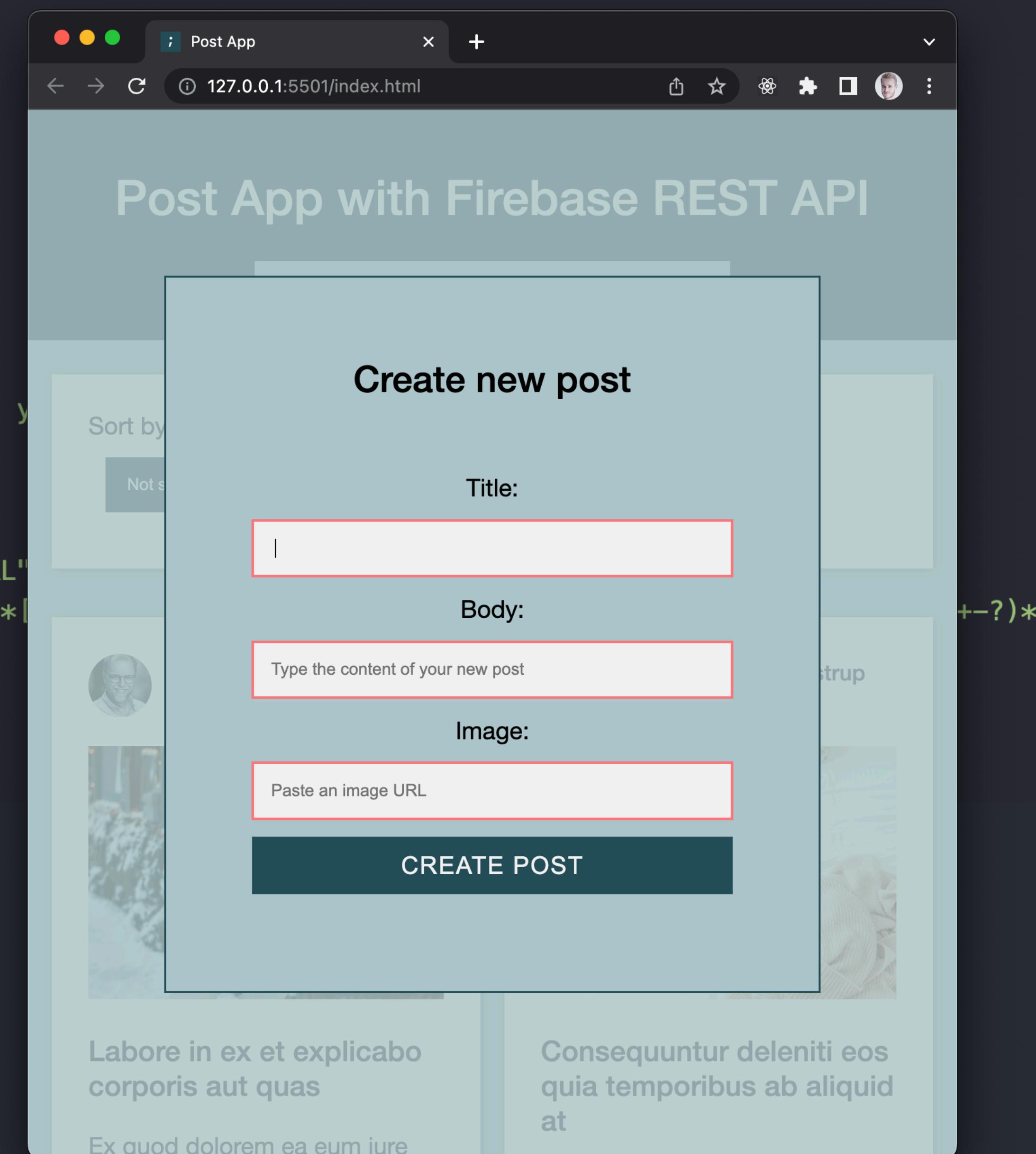
input:invalid {
  border: 2px solid red;
}

```

```

input:focus,
select:focus,
textarea:focus,
button:focus {
  outline: none;
}

```



```

<!-- dialog - create new post -->
<dialog id="dialog-create-post">
  <h2>Create new post</h2>
  <form id="form-create-post" method="dialog">
    <label for="title">Title:</label>
    <input type="text" id="title" name="title" placeholder="Type a new title"
      minlength="3"
      maxlength="100"
      required>
    <label for="body">Body:</label>
    <input type="text" id="body" name="body" placeholder="Type the content of your new post"
      minlength="5"
      required>
    <label for="image">Image:</label>
    <input type="text" id="image" name="image" placeholder="Paste an image URL"
      pattern="[Hh][Tt][Tt][Pp][Ss]?:\/\/(?:[a-zA-Z\u00a1-\uffff0-9]+-?)*[a-zA-Z\u00a1-\uffff0-9]+(\.[a-zA-Z\u00a1-\uffff0-9]+)*"
      required>
    <button>Create Post</button>
  </form>
</dialog>

```

```

input:invalid {
  border: 2px solid red;
}

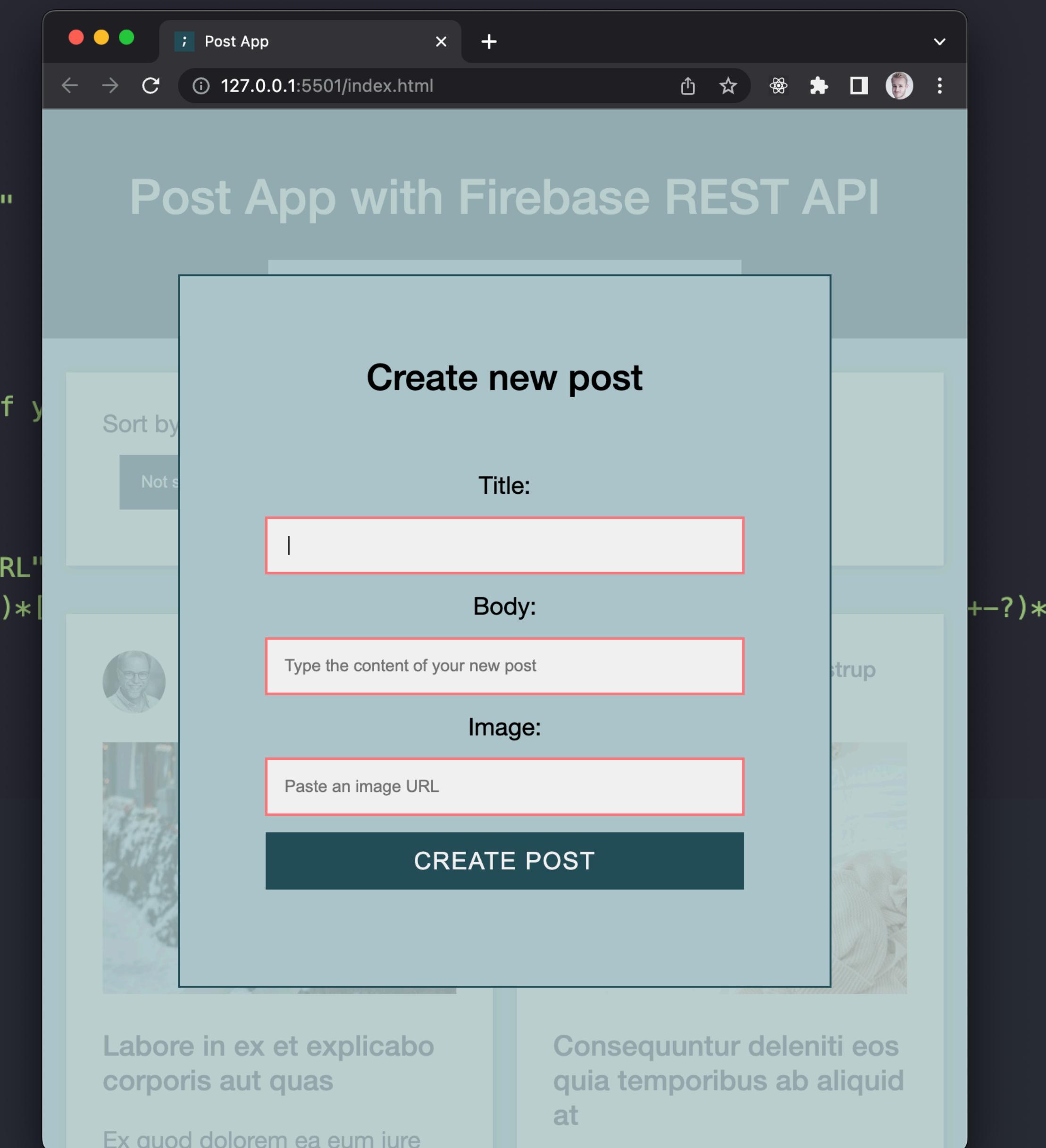
```

```

input:focus,
select:focus,
textarea:focus,
button:focus {
  outline: none;
}

```

Change to correct type



[https://www.w3schools.com/js/js\\_validation.asp](https://www.w3schools.com/js/js_validation.asp)  
[https://developer.mozilla.org/en-US/docs/Learn/Forms/Form\\_validation](https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation)

```

<!-- dialog - create new post -->
<dialog id="dialog-create-post">
  <h2>Create new post</h2>
  <form id="form-create-post" method="dialog">
    <label for="title">Title:</label>
    <input type="text" id="title" name="title" placeholder="Type a new title"
      minlength="3"
      maxlength="100"
      required>
    <label for="body">Body:</label>
    <input type="text" id="body" name="body" placeholder="Type the content of your new post"
      minlength="5"
      required>
    <label for="image">Image:</label>
    <input type="url" id="image" name="image" placeholder="Paste an image URL"
      required>
    <button>Create Post</button>
  </form>
</dialog>

```

```

input:invalid {
  border: 2px solid red;
}

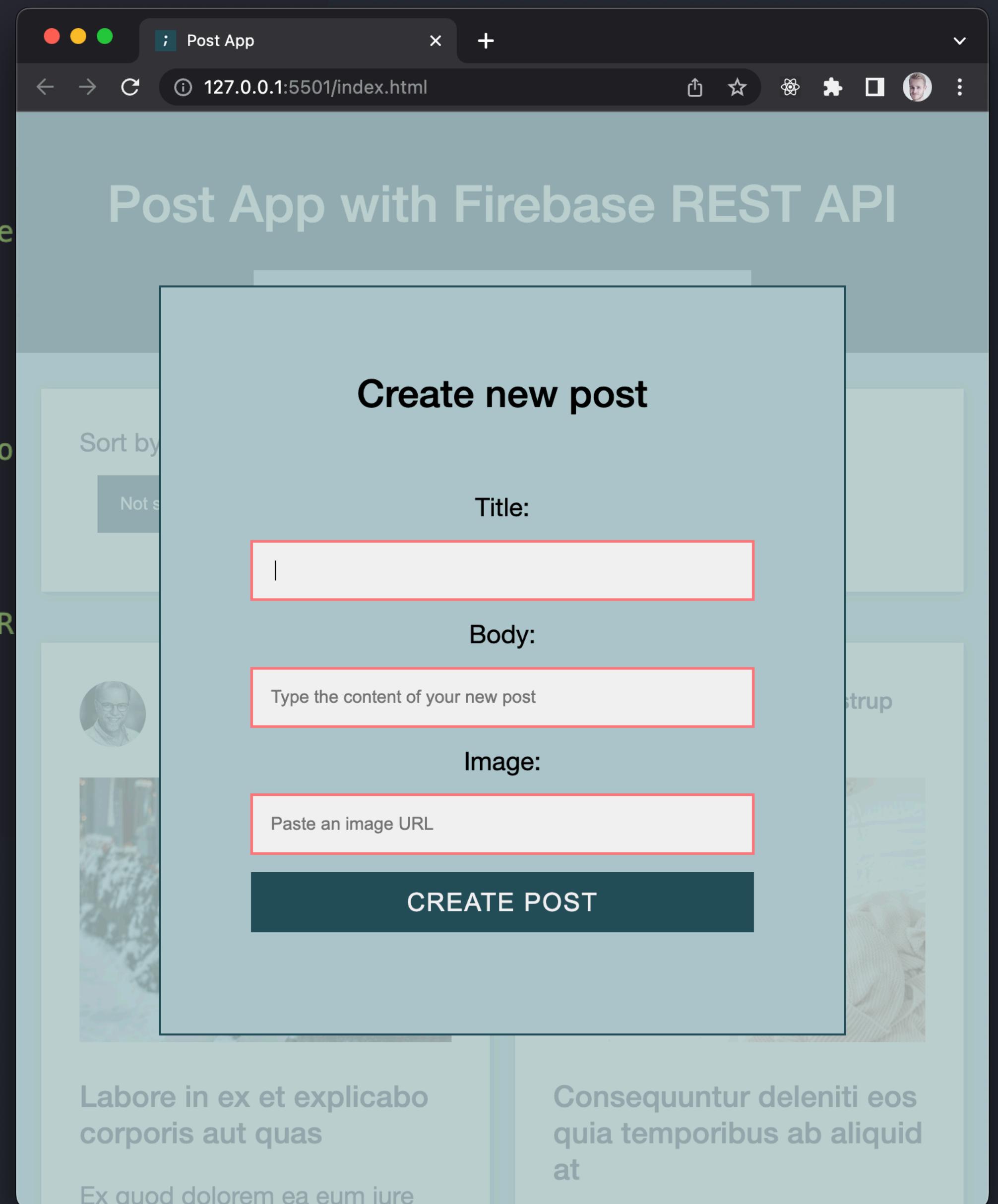
```

```

input:focus,
select:focus,
textarea:focus,
button:focus {
  outline: none;
}

```

And remove pattern



[https://www.w3schools.com/js/js\\_validation.asp](https://www.w3schools.com/js/js_validation.asp)

[https://developer.mozilla.org/en-US/docs/Learn/Forms/Form\\_validation](https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation)

# Exercises



# OBJECT TEACHERS EXERCISE

The screenshot shows a web browser window with the title "TEACHERS". The address bar indicates the URL is 127.0.0.1:5500/object-teachers/index.html. The browser is set to an iPhone 6/7/8 view with a width of 375 and a height of 667, at 98% zoom, and is connected to an online network.

The main content area displays a mobile application interface. At the top, it says "TEACHERS". Below that is a portrait of a woman with blonde hair, identified as Birgitte Kirk Iversen, a Senior Lecturer, with an email link: [bki@baaa.dk](mailto:bki@baaa.dk). Below her profile is another partial portrait of a man.

The browser's developer tools are open on the right side. The "Console" tab is active, showing the following log entries:

- ▶ Object main.js:52
- ▶ Object main.js:53
- ▶ Object main.js:54
- ▶ Object main.js:55
- Live reload enabled. index.html:49

# OBJECT TEACHERS

Create a grid view with your new teachers.

1. Make a copy of the project-template folder and save it in your own dev library. Rename the project to object-techers-grid.
2. Create two variables, teacher1 & teacher2, containing a teacher object with properties (name, position, department, address, mail & phone):
  - 2.1. BKI: <https://www.baaa.dk/contact/find-employee/employee/birgitte-kirk-iversen>
  - 2.2. RACE: <https://www.baaa.dk/contact/find-employee/employee/rasmus-cederdorff>
3. Append the data from the teacher objects to the DOM (display them in the HTML page).
4. Add at least two new objects. The objects must consist of data about teachers:
  - 4.1. PETJ: <https://www.baaa.dk/contact/find-employee/employee/per-thykjaer-jensen>
  - 4.2. JDS: <https://www.baaa.dk/contact/find-employee/employee/jeffrey-david-serio>
5. Append the data from the teacher objects to the DOM (display them in the HTML page).
6. Make sure to display all the properties: position, department, address, mail & phone
7. Add styling and display the teachers nicely in a grid.
8. Add an image property to every teacher object and display it in the DOM.

# ARRAY TEACHERS

- Make a copy of your local project object-teachers and rename to array-teachers
- Modify the structure of your script (your JavaScript), in order to use an array and a loop to display your teachers:
  - Define an array and add the object to the array.
  - Test and debug the array using `console.log`
  - Use a `for` of loop to loop through the array and display the teachers (append to the DOM).
  - Add another teacher object and test.

# FAMILY MEMBERS

- Use the project-template. Copy the project into your own dev folder and rename family-members.
- If needed, get inspired by the project template called array-family-members-template
- Declare a new array called familyMembers.
- Use push (...) to add at least four person objects (family members) to the array.
- The person object must consist of the properties name, age & relation. Add more properties if you like.
- Make use of .length & pop () and tryout other array methods.
- Loop through familyMembers and append the data to the DOM (your HTML page).
- Loop through familyMembers and log persons older than 32 years (add a condition).

# FAMILY MEMBERS

- Extra:
  - Implement search functionality.
  - Implement a form and a function to add new family members to the array.
  - Make sure you display the new added family member.

# FAMILY MEMBERS 2

- Make use of your new knowledge about backtick strings, arrays and functions.
- Customise the previously exercise about family members.
- Get inspired by the project template called `array-family-members-template`
- Create a function called `appendFamilyMembers()` looping through `familyMembers` and appending the properties from each object to the DOM (your HTML page). Make use of the backtick string to create your HTML template. Remember to call your function.
- Create another function that loops through `familyMembers` and log persons older than 32 years (add a condition). Remember to call your function.
- Extra: Create a button to handle the above filter function and append the new filtered `familyMembers` to the DOM.

# ARRAY TEACHERS 2

- Make use of your new knowledge about backtick strings, arrays and functions.
- Customise the exercise about teachers by using:
  - an array to define and hold the teacher objects
  - a function to loop through the array of teachers and append the teachers to the DOM (your HTML page).
  - the backtick string to create your HTML template defining the HTML of the teacher.
- Create another function that loops through the teachers array and logs all teachers with the position === "Senior Lecturer"
- Extra: Create buttons to handle *show* and *hide* Lectures and Senior Lecturers (filter by teacher.position).

# EXERCISE

## TEACHER AND/OR FAMILY MEMBER

- Make it possible to add a new teacher or family member (a new object).
- Get inspired by the code examples in the slides.
- Add a form with input fields and a save button.
- Implement a function called `createTeacher()` or `createFamilyMember()`.
- Make use of `push()` inside the function to add a new object to the array.
- Add an `eventListener` (`onclick event`) to the save button - must call `createTeacher()` or `createFamilyMember()`.
- Make sure the create function appends the new object data to the DOM (your HTML file).

# EXERCISE

## TEACHER AND/OR FAMILY MEMBER

- Improvements:
  - Add search functionality: Add an input field to your `index.html` and implement a search function in your script.
  - Add edit functionality
  - Add delete functionality
- JSON & Fetch (+++):
  - Create a JSON file consisting of an array of `teachers` or `familyMembers`.
  - Make use of `fetch (...)` to fetch the data from the JSON file and append it to the DOM.
- JSON: [https://www.w3schools.com/js/js\\_json\\_intro.asp](https://www.w3schools.com/js/js_json_intro.asp)
- FETCH: <https://javascript.info/fetch> or <https://scrimba.com/scrim/cVye2ztW>

# JSON & FETCH FAMILY MEMBERS

- With the exercise about family members on your mind:
- Explore the `fetch-family-members-template` and use it as your code base.
- Define your own family members as json in `json/persons.json`
- Add at least 3 JSON objects with at least 5 properties (`name`, `age`, `hairColor`, `relation`, `img`)
- Make use of `fetch()` to get the data from the JSON file in to your JavaScript.
- Append the data from the JSON file to your HTML file by using JavaScript.
- Add more properties like `birthDate`, `height`, `email`, etc. and make sure to display them in your HTML page.
- Extra: Use `async` & `await` to implement `fetch()`

SINGLE PAGE WEB APP TEMP | Products | +

127.0.0.1:5501/spa-products-fetch-json-enhanced/index.html

# PRODUCTS

Order by: Choose here ▾

Show out of stock

Search



**MacBook Pro 13"**

Apple

Price: 11799 kr.

Status: outOfStock

[EDIT](#) [DELETE](#)



**MacBook Pro 15"**

Apple

Price: 21499 kr.

Status: inStock

[EDIT](#) [DELETE](#)



**Zenbook 14"**

ASUS

Price: 8099 kr.

Status: outOfStock

[EDIT](#) [DELETE](#)



**Unknown laptop**

ASUS

Price: 8099 kr.

Status: inStock

[EDIT](#) [DELETE](#)



**ASUS ZenBook**

ASUS

Price: 8099 kr.

Status: inStock

[EDIT](#) [DELETE](#)



**Unknown tablet**

ASUS

Price: 8099 kr.

Status: inStock

[EDIT](#) [DELETE](#)

PRODUCTS ADD PRODUCT

# PRODUCT WEB APP

- Make a copy of the project template *spa-products-template* and paste it into your own code library.
- Declare an array with at least three product objects with the following properties: model, brand, price & img
- Implement a function that appends the products to the DOM
- Implement a form and a function to create new products
- Implement search functionality

# PRODUCT WEB APP (EXTRA)

- Project template: spa-products-template
- Use your knowledge about JavaScript and SPA to build a web app.
- Declare a new array with at least 4 products. The product objects must have the following properties: model, brand, price, img and status
- Two of your products must have the status `outOfStock` and two `inStock`.
- The img should just be an URL to an image like: [https://store.storeimages.cdn-apple.com/4668/as-images.apple.com/is\\_mbp13touch-space-select-201807?wid=904&hei=840&fmt=jpeg&qlt=80&op\\_usm=0.5,0.5&.v=1529520060550](https://store.storeimages.cdn-apple.com/4668/as-images.apple.com/is_mbp13touch-space-select-201807?wid=904&hei=840&fmt=jpeg&qlt=80&op_usm=0.5,0.5&.v=1529520060550)
- Display all products by appending the products to the PRODUCTS page.
- Another tab/page with a form to add new products.
- Customise and style your app.
- Add functionality to filter and/or sort by the status. You could add an input checkbox to show and hide `outOfStock` products.
- Add functionality to handle edit and delete.

# JSON & FETCH PRODUCTS

- With the exercises about family members and products on your mind:
- Copy and use the `spa-products-fetch-json-template` as your code base.
- Define products as json in `json/products.json`
- Add at least 4 JSON objects with the following properties: `model`, `brand`, `price`, `status` and `img`
- Make sure the products have a property called `status`. Two of your products must have the `status` “`outOfStock`” and two “`inStock`”.
- Use `fetch()` to get the data from the JSON file and append the products to your HTML page.
- Implement the add functionality.
- Implement the search functionality.
- Extra / advanced:
  - Add functionality to filter and/or sort by the `status`. You could add an input checkbox to show only products “`inStock`”. Make use of your knowledge about functions, loops, `filter`, etc.
  - Add functionality to handle edit and delete.
  - Customise and style your web app.



Code  
Every  
Day