

# JavaScript Concepts

```
index.html
85 }
86
87 /**
88 * Fetches post data from my headless cms
89 */
90 function getPersons() {
91   fetch('http://headlesscms.cederdorff.com/wp-json/wp/v2/posts?_embed')
92     .then(function(response) {
93       return response.json();
94     })
95     .then(function(persons) {
96       appendPersons(persons);
97     });
98 }
99 /*
100 Appends json data to the DOM
101 */
102 function appendPersons(persons) {
103   let htmlTemplate = '';
104   for (let person of persons) {
105     console.log();
106     htmlTemplate +=
107       `

108         
109         <h4>${person.title.rendered}</h4>
110         <p>${person.acf.age} years old</p>
111         <p>Hair color: ${person.acf.hairColor}</p>
112         <p>Relation: ${person.acf.relation}</p>
113

`;
114   }
115   document.querySelector("#family-members").innerHTML += htmlTemplate;
116 }
```

# Content

- Variables
- Data Types
- Objects
- Arrays
- Loops
- Conditional Statements - if/else
- Ternary operator
- Array Methods
- Template String
- Functions
- Destructuring
- Spread Operator
- Modules, import & Export

# Programming knowledge

## JavaScript

```
1 "use strict";
2 const doc = document;
3 doc.addEventListener("DOMContentLoaded", function() {
4     const postFetchUrl = "http://api.cederdorff.com/wp-json/wp/v2/posts?_embed",
5         const postFetchUrl = "http://api.cederdorff.com/wp-json/wp/v2/posts?_embed&categories=2";
6
7     fetch(postFetchUrl)
8         .then(function(response) {
9             return response.json();
10        })
11        .then(function(posts) {
12            console.log(posts);
13            appendClients(posts);
14        });
15
16    function appendClients(clients) {
17        let html = "";
18        for (let i = 0; i < clients.length; i++) {
19            let client = clients[i];
20            console.log(client);
21            if (i % 2 == 0) { // index is even
22                html += `<div class="client">
23                    <div class="col-left even">
24                        
25                    </div>
26                    <div class="col-right even">
27                        <div class="container">
28                            <div class="center">
29                                <h3>${client.title.rendered}</h3>
30                                <h4>${client.meta.sub_title[0]}</h4>
31                                ${client.content.rendered}
32                                <div class="client-meta">
33                                    
34                                    <a href="#" target="_blank"><${client.meta.link_text[0]}><i class="ion-ios-arrow-forward"></i><i class="ion-ios-arrow-forward"></i></a>
35                                </div>
36                            </div>
37                        </div>
38                    </div>
39                </article>
40            } else { // index is odd
41                html += `<div class="client">
42                    <div class="col-right">
43                        
44                    </div>
45                </article>
46            };
47        }
48    }
49}
```

# Variables

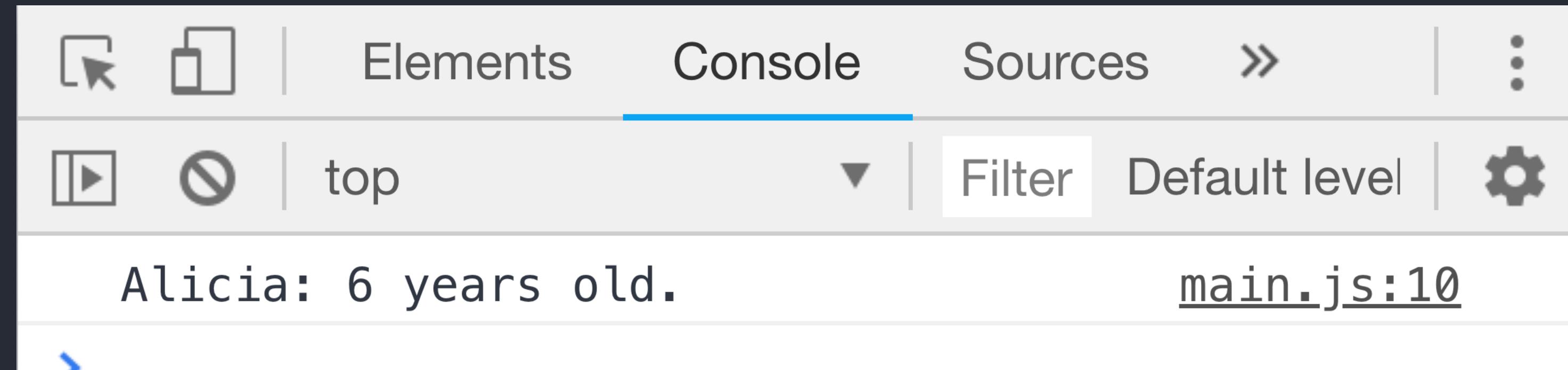
... are used to store data (values, objects, collections) in the memory

# Variables

Store data in the memory

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

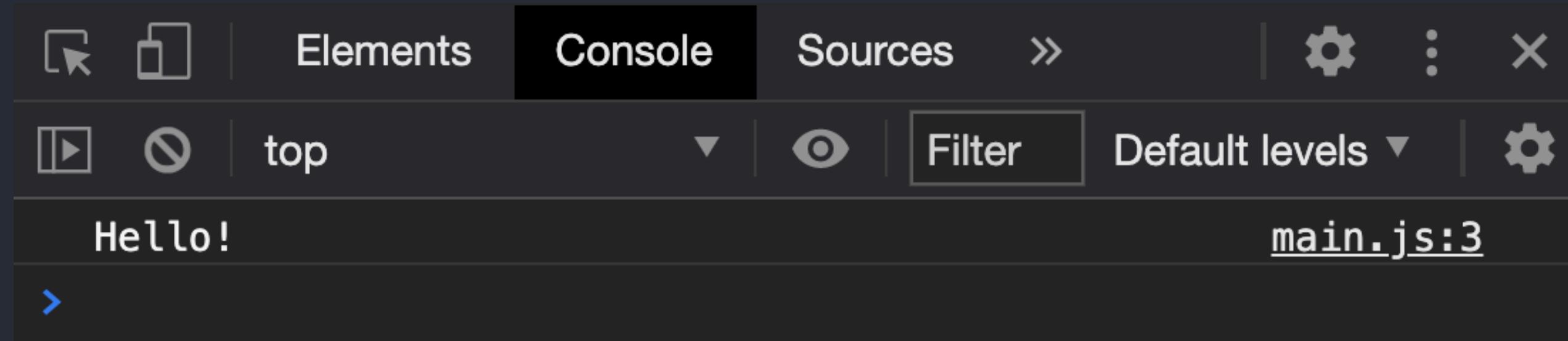
console.log(name + ": " + age + " years old.");
```



# Variables

Store data in the memory

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

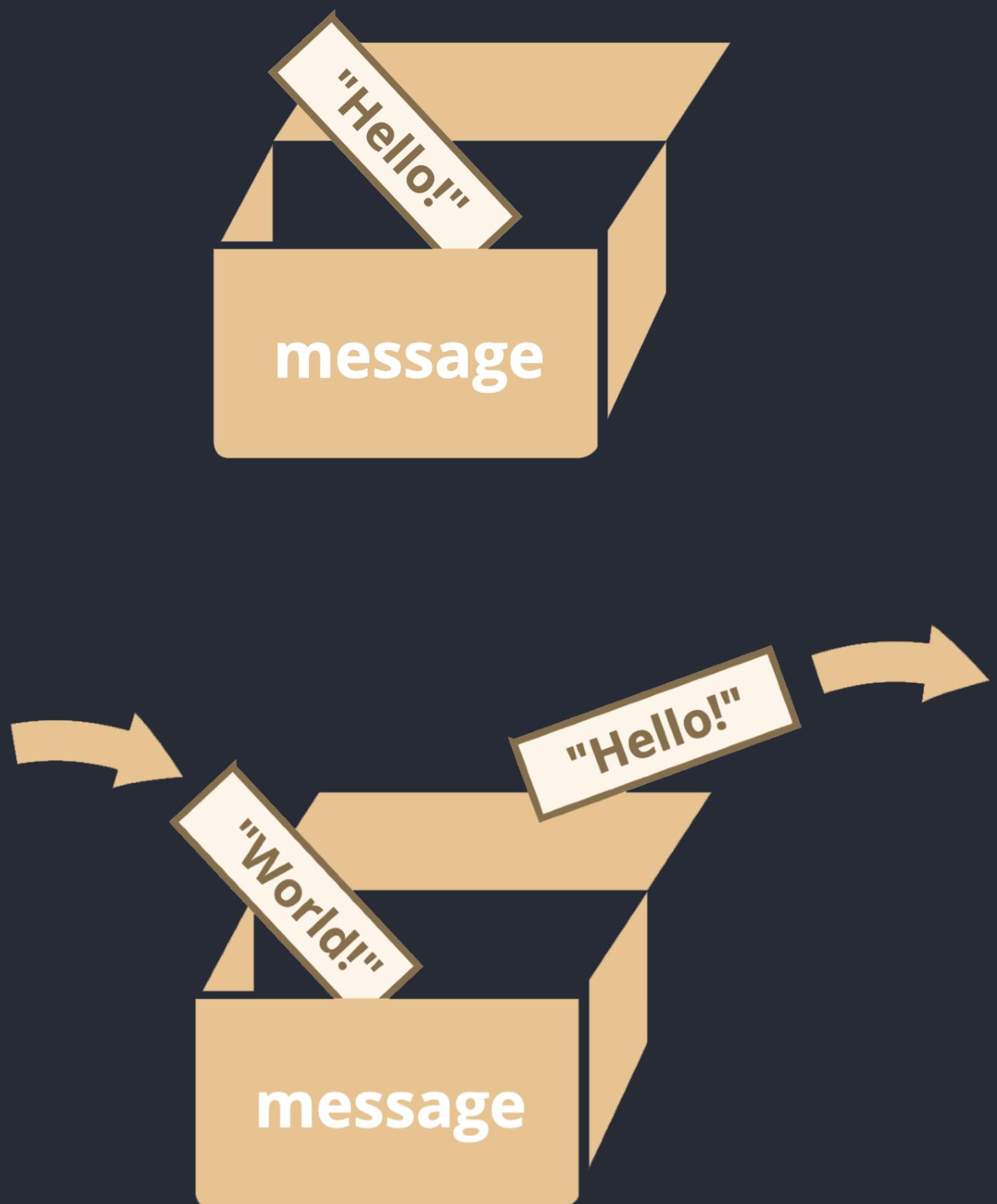
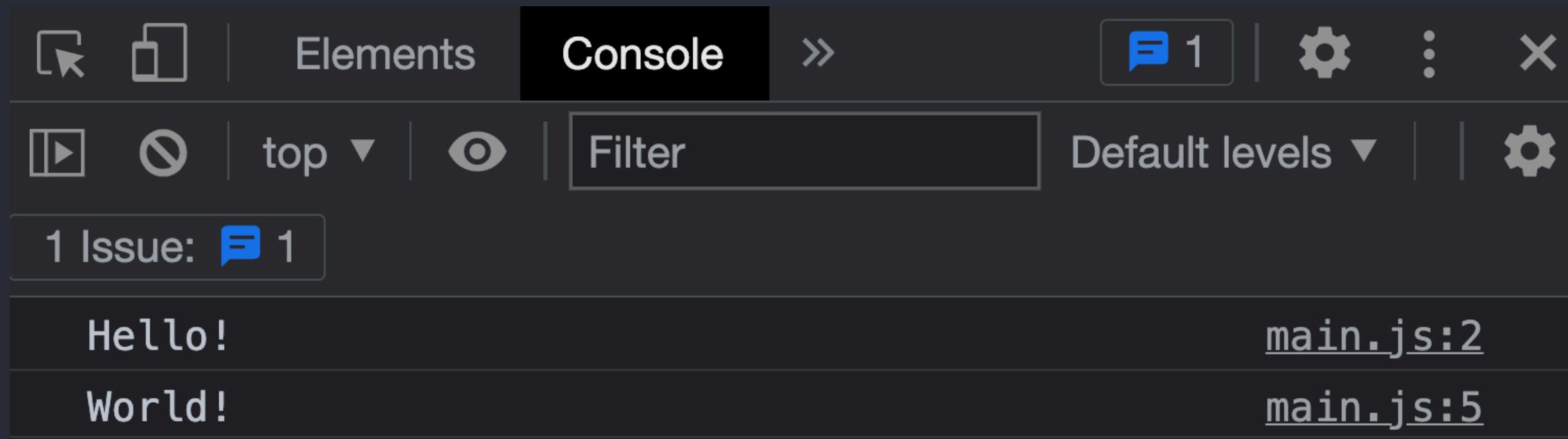


# Variables

Store data in the memory

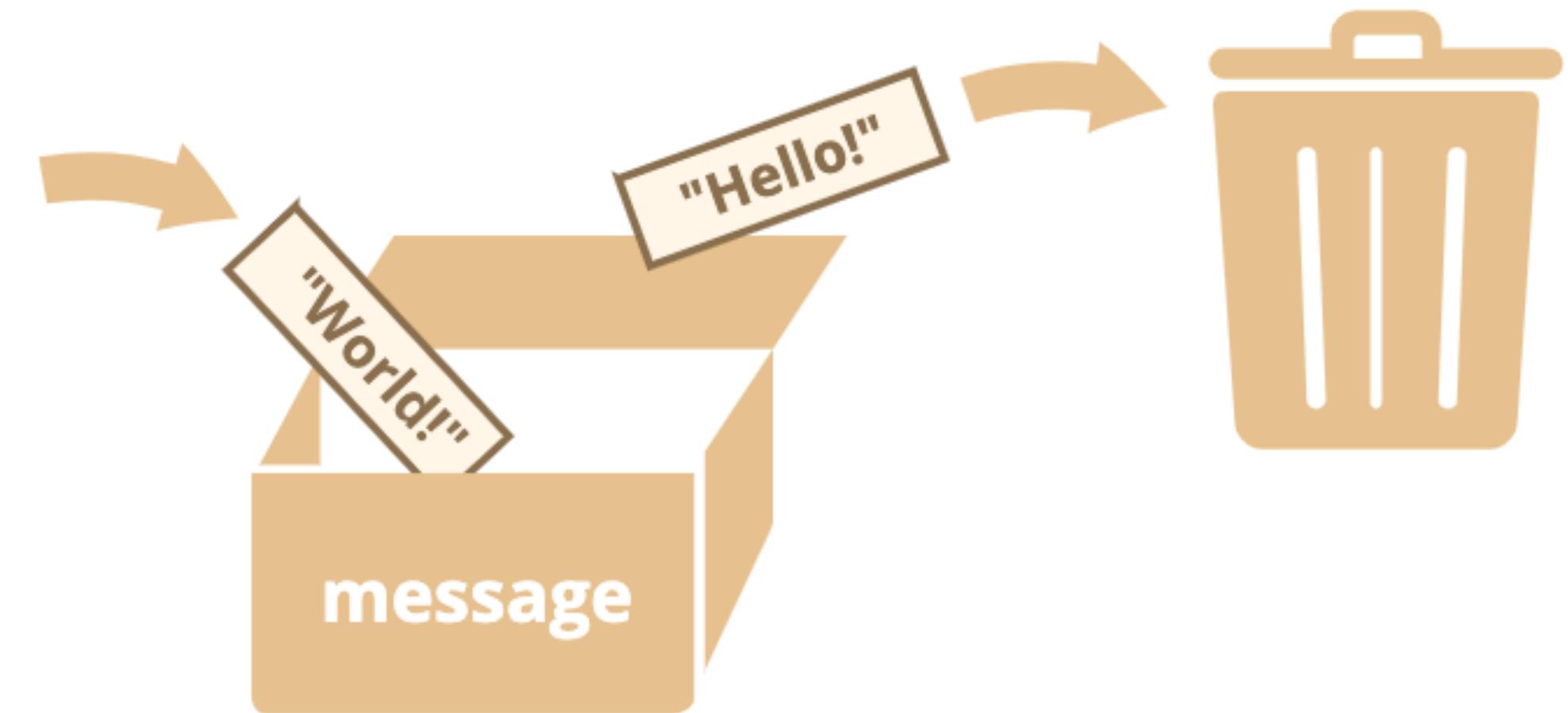
```
let message = "Hello!";
console.log(message);
```

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



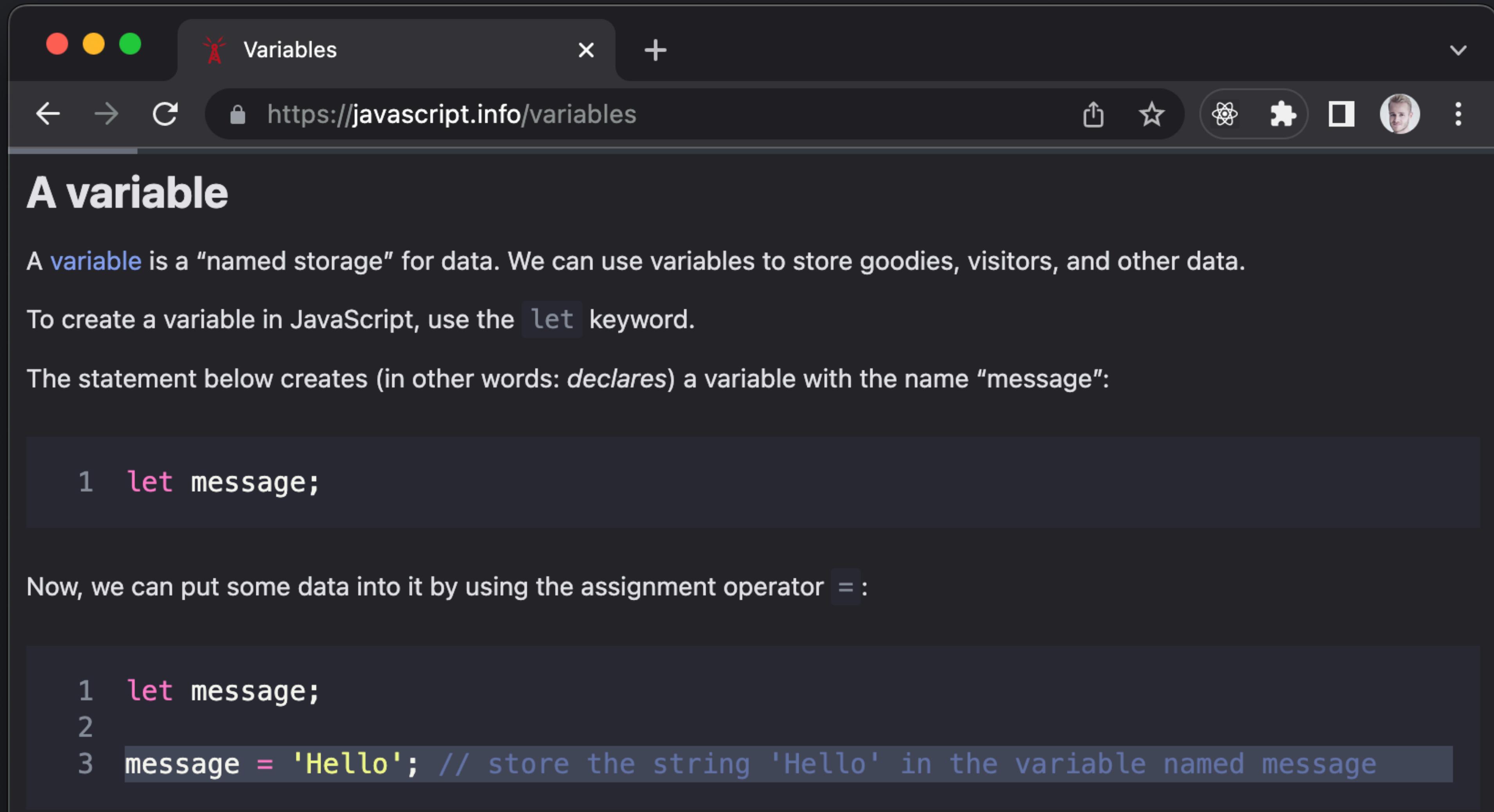
# Variable

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
```

# `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

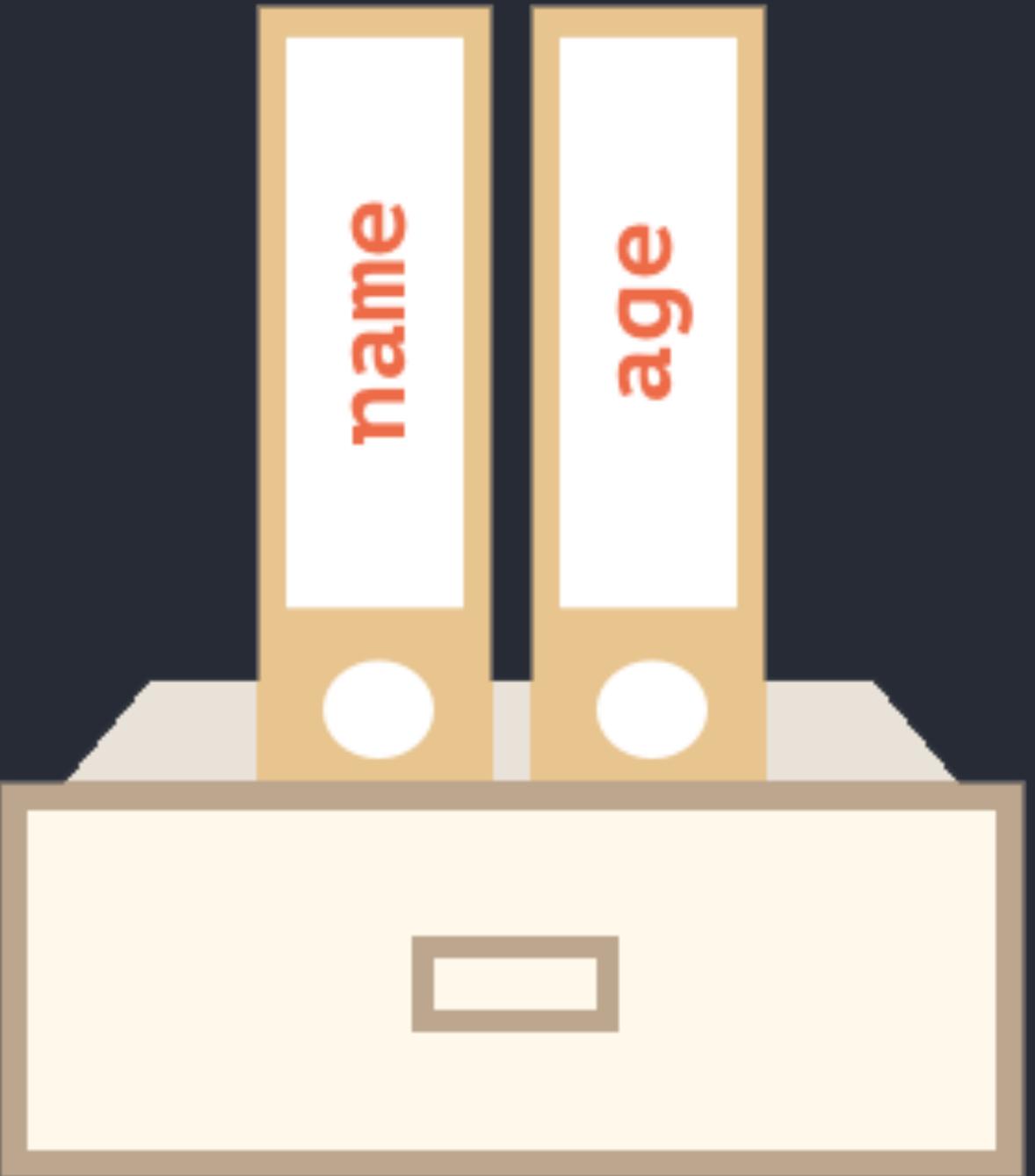
# Objects

A set of named values

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

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

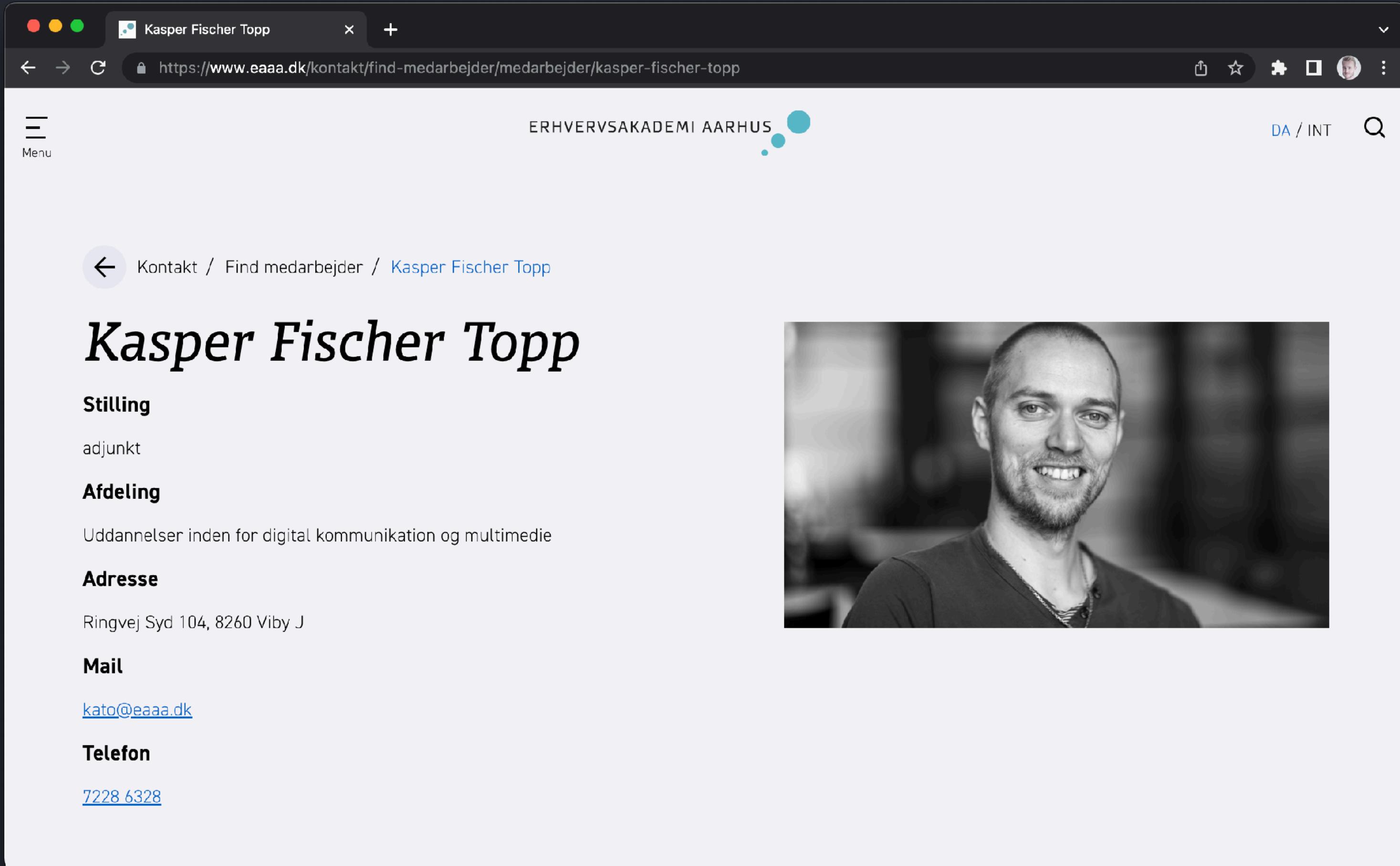
user



Alicia is 6 years old.

main.js:11

# Objects



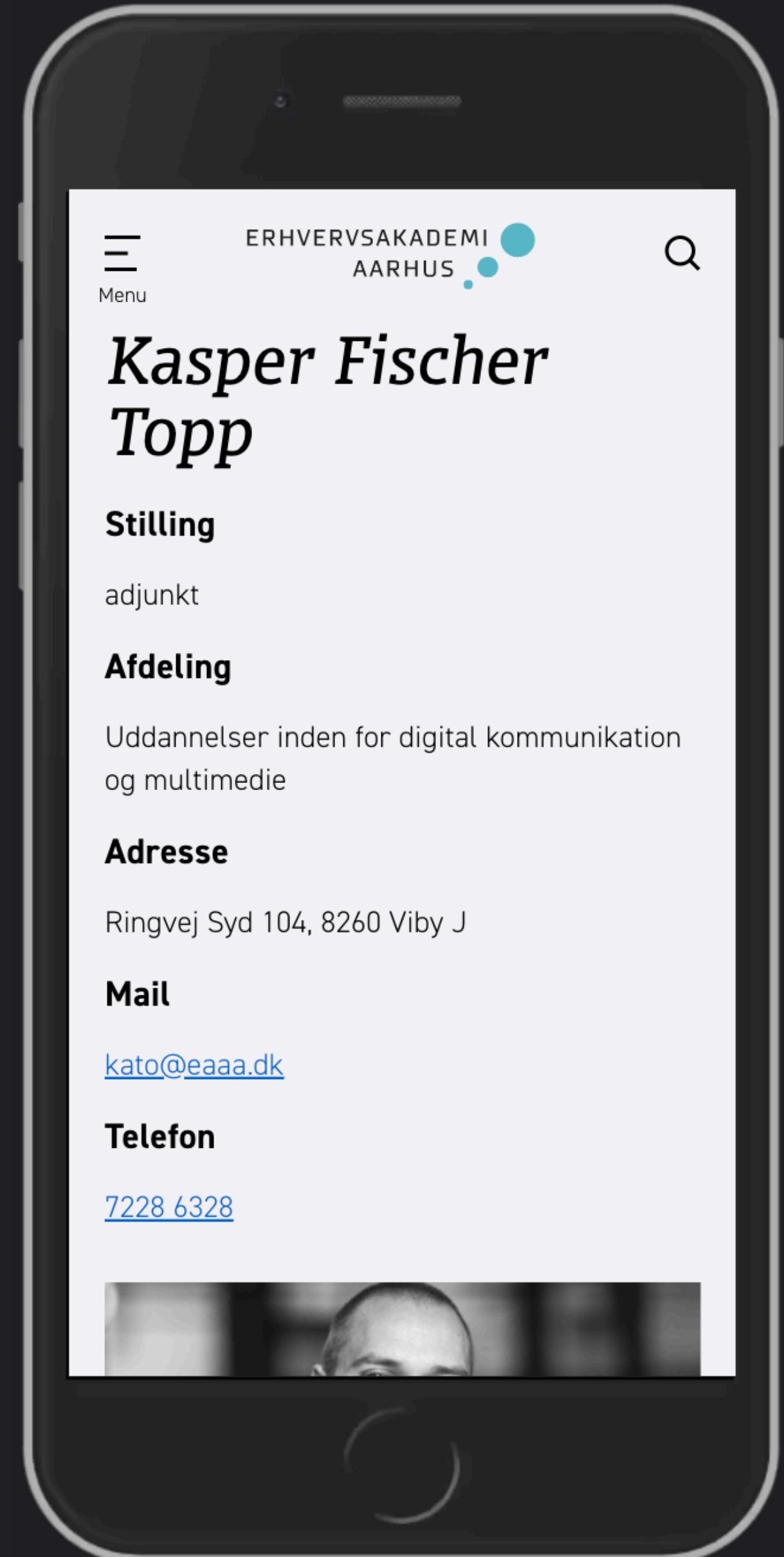
A screenshot of a web browser window displaying a profile page for Kasper Fischer Topp. The browser has a dark theme with red, yellow, and green window control buttons. The title bar shows the page is titled "Kasper Fischer Topp". The address bar contains the URL <https://www.eaaa.dk/kontakt/find-medarbejder/medarbejder/kasper-fischer-topp>. 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

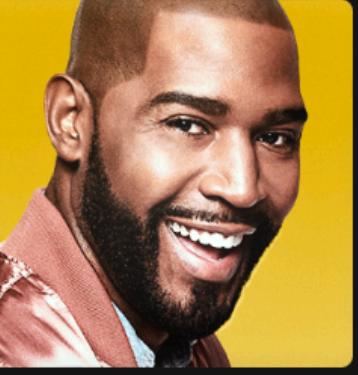
A set of named values

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

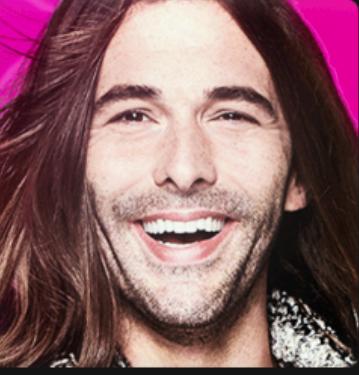


NETFLIX

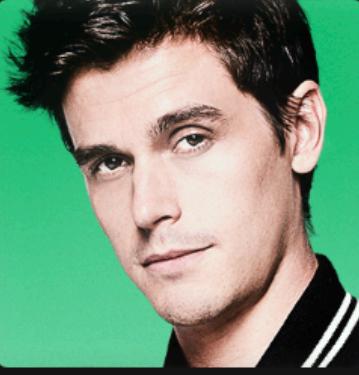
# Hvem ser?



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

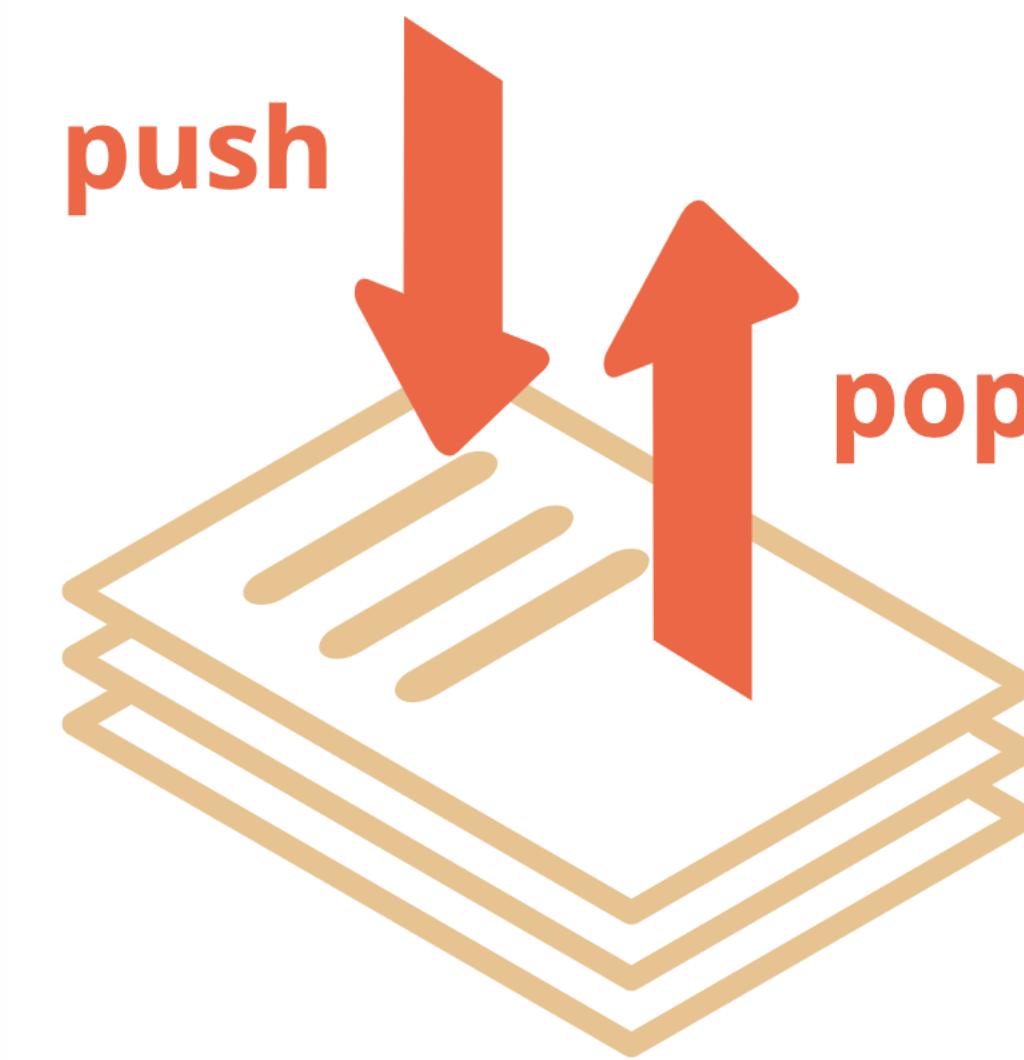
Writers

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

# 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

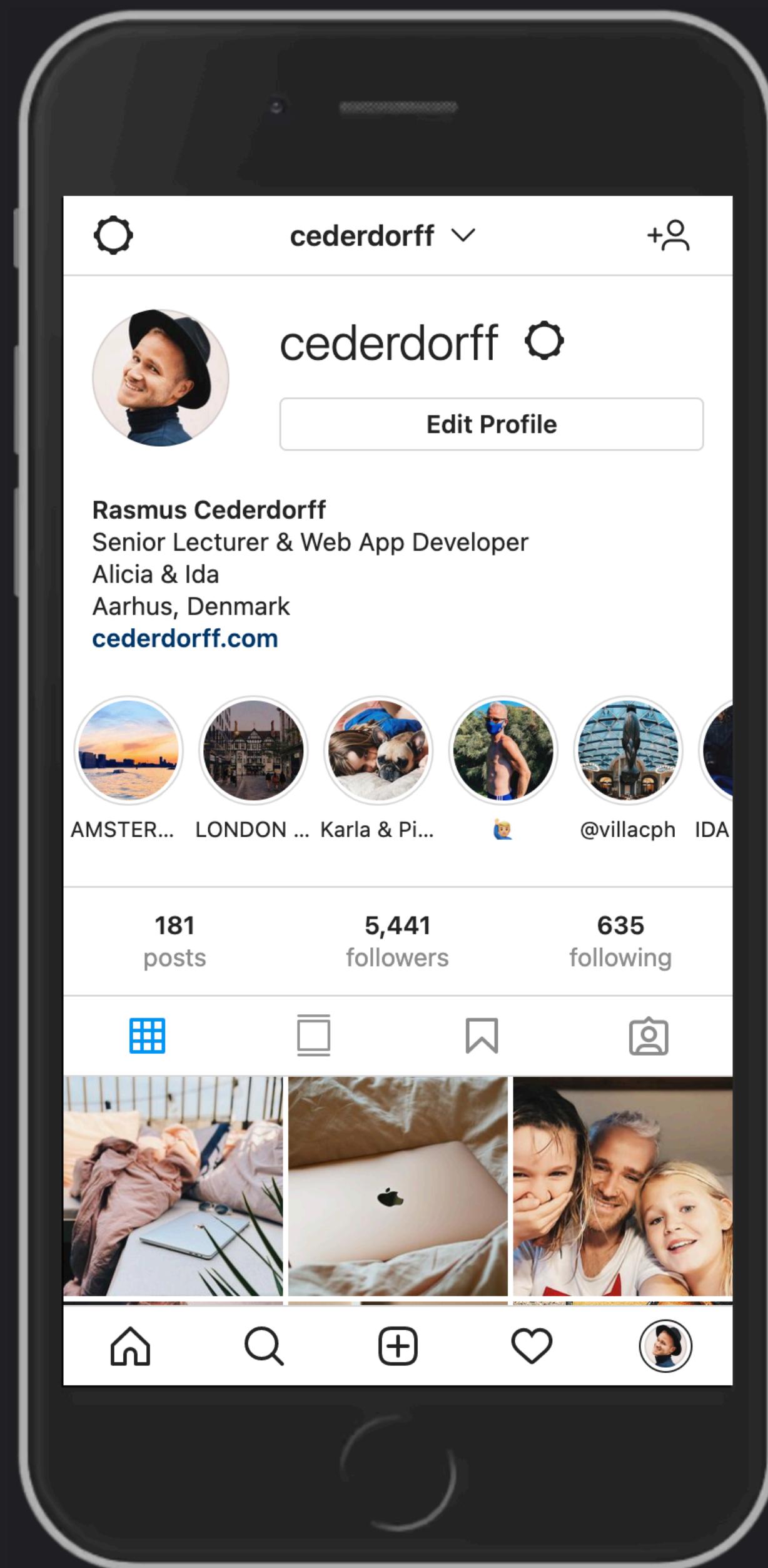
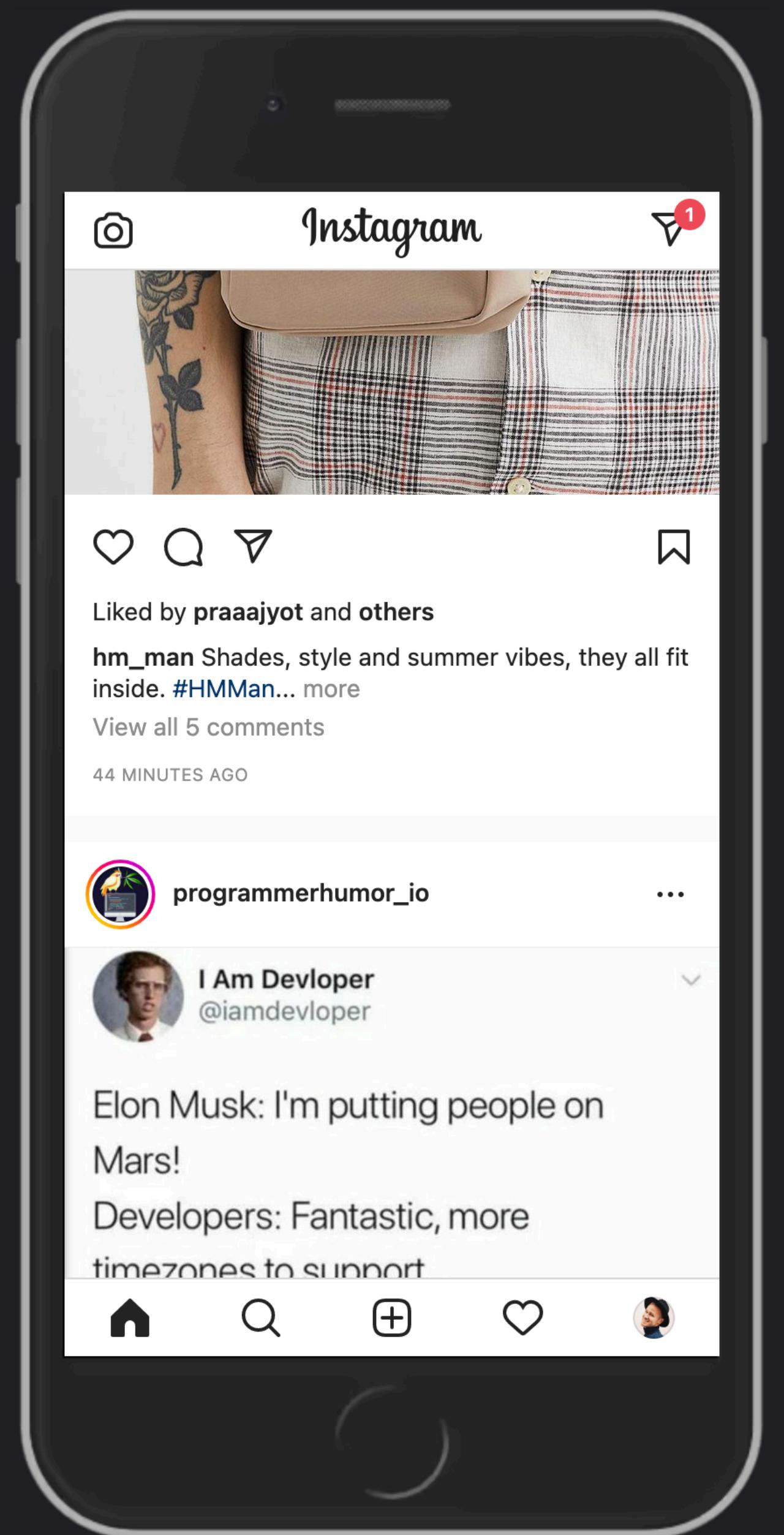
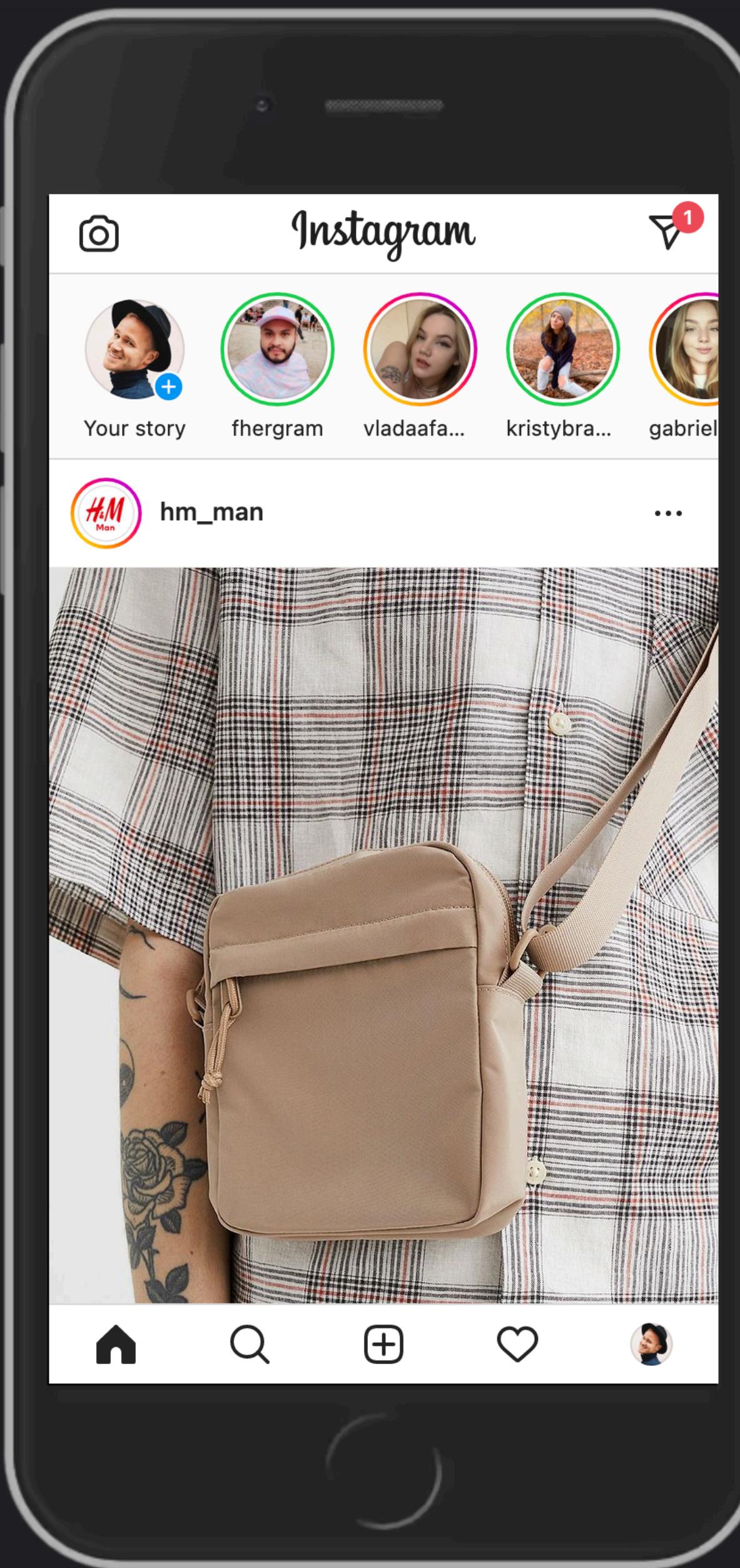
Rasmus Cederdorff	
<b>Position:</b> Lecturer	<i>Michael Hvidtfeldt</i>
<b>Department/</b> Multimedia De Digital Conce	
<b>Address:</b> Ringvej Syd 10	<b>Position:</b> Lecturer <i>Birgitte Kirk Iversen</i>
<b>Mail:</b> <a href="mailto:race@baaa.dk">race@baaa.dk</a>	<b>Department/</b> Multimedia Di
<b>Phone:</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

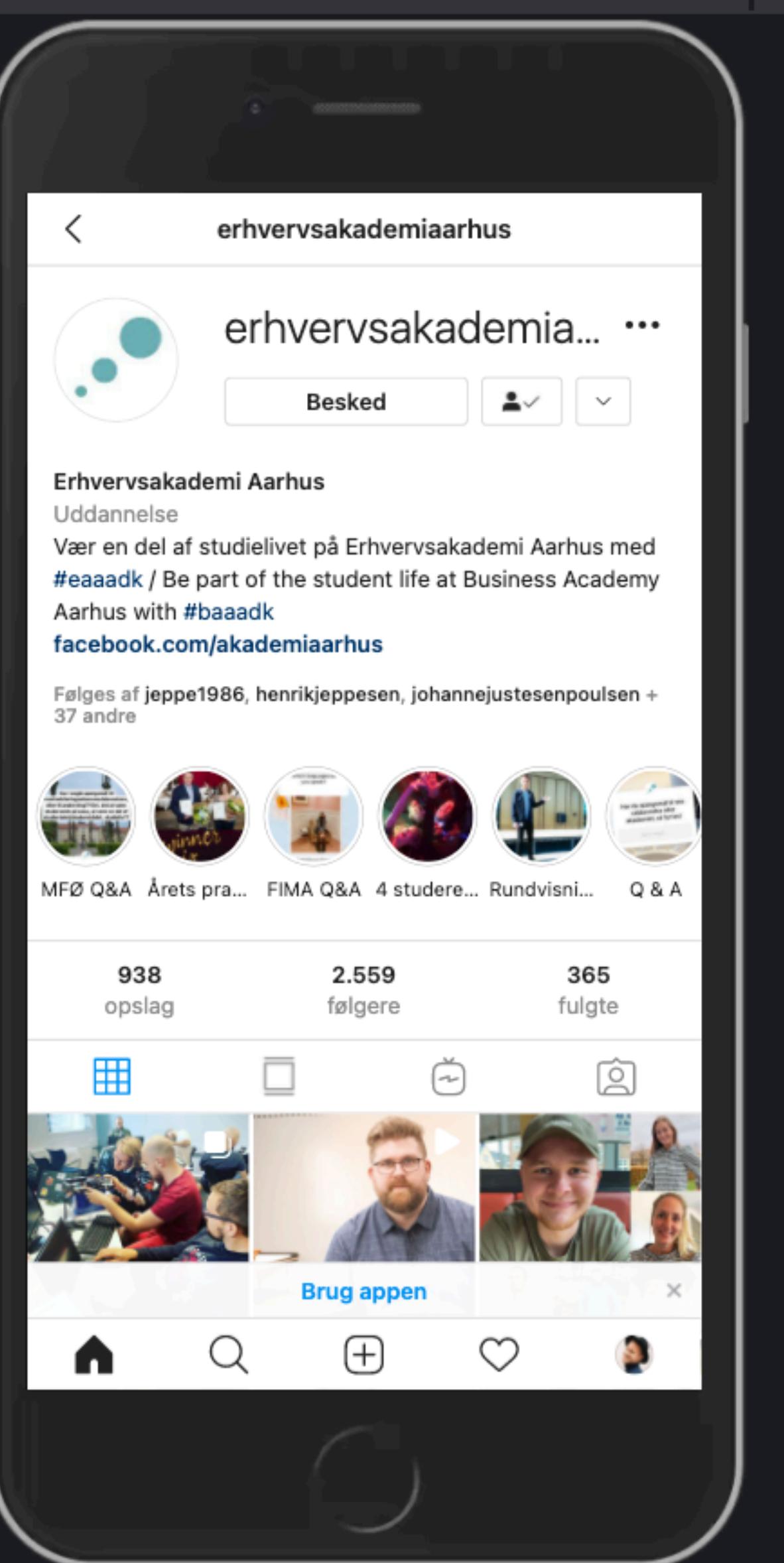


```
main.js:21
▼ (3) [...], [...], [...] ⓘ
▶ 0: {name: "Birgitte Kirk Iversen", mail: "bki@baaa..."}
▶ 1: {name: "Michael Hvidtfeldt", mail: "mhv@baaa.dk..."}
▶ 2: {name: "Rasmus Cederdorff", mail: "race@baaa.dk..."}
  length: 3
▶ __proto__: Array(0)
main.js:22
▶ {name: "Michael Hvidtfeldt", mail: "mhv@baaa.dk"}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





Filter  Hide data URLs

All | Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other  Has blocked cookies  Blocked Requests

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

Name	Headers	Preview	Response	Initiator	Timing	Cookies
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						

▼ {data: {user: {id: "5672009939",...}}, status: "ok"}  
  ▼ data: {user: {id: "5672009939",...}}  
    ▼ user: {id: "5672009939",...}  
      ▼ edge\_web\_feed\_timeline: {page\_info: {has\_next\_page: true,...},...}  
        ▼ edges: [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}]  
          ► 0: {...}  
          ► 1: {...}  
          ► 2: {...}  
          ▼ 3: {...}  
            ▼ node: {\_\_typename: "GraphVideo", id: "2646664183338840310", dimensions: {height: 1080, width: 1080}, attribution: null, clips\_music\_attribution\_info: null, coauthor\_producers: [], comments\_disabled: false, dash\_info: {is\_dash\_eligible: false, video\_dash\_manifest: null, number\_of\_qualifiers: 0}, dimensions: {height: 1080, width: 1080}, display\_resources: [{...}, {...}, {...}], display\_url: "https://scontent-cph2-1.cdninstagram.com/v/t51.2885-15/fr/e15/2646664183338840310/?ccb=1\_5&ccb=1\_5&igshid=1qjwv6y0zg65&utm\_source=copy\_link&utm\_medium=copy\_link", edge\_media\_preview\_comment: {count: 1, page\_info: {has\_next\_page: false, end\_cursor: "CgQAAQ"}, edges: [{node: {id: "553627065",...}}]}, edge\_media\_preview\_like: {count: -1, edges: [{node: {id: "553627065",...}}]}, edge\_media\_to\_caption: {edges: [{node: {...}}]}  
            ▼ edges: [{node: {...}}]  
              ▼ 0: {node: {...}}  
                ▼ node: {...}  
                  text: "Priden er netop slut. I ti dage har vi debatteret, festet og v...

► edge\_media\_to\_sponsor\_user: {edges: []}  
► edge\_media\_to\_tagged\_user: {edges: []}  
  fact\_check\_information: null  
  fact\_check\_overall\_rating: null  
  follow\_hashtag\_info: null

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

property value

```
[{"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", "unread_count": 0, "group_categories": []}, {"id": "36267", "name": "Martin Rieper Boesen", "created_at": "2020-08-10T10:45:00+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", "unread_count": 1, "group_categories": []}, {"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", "unread_count": 0, "group_categories": []}, {"id": "14427", "name": "Rasmus Cederdorff", "created_at": "2020-08-10T10:45:00+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", "unread_count": 0, "group_categories": []}, {"id": "41", "name": "Jeffrey David Serio", "created_at": "2020-08-10T10:45:00+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", "unread_count": 0, "group_categories": []}, {"id": "24043", "name": "Charlotte Meng Emanuel Dyrholm", "created_at": "2020-08-10T10:45:00+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", "unread_count": 0, "group_categories": []}, {"id": "23978", "name": "Jeppe Frik", "created_at": "2020-08-07T10:45:00+02:00", "email": null, "sis_user_id": null, "short_name": "Jeppe Frik", "sortable_name": "Frik, Jeppe", "integration_id": null, "login_id": null, "unread_count": 0, "group_categories": []}, {"id": "23963", "name": "Daniel Tjerrild Gamborg", "created_at": "2020-08-07T10:45:00+02:00", "email": null, "sis_user_id": null, "short_name": "Daniel Tjerrild Gamborg", "sortable_name": "Gamborg, Daniel Tjerrild", "integration_id": null, "login_id": null, "unread_count": 0, "group_categories": []}, {"id": "23992", "name": "Casper Hedegaard Hansen", "created_at": "2020-08-07T10:45:00+02:00", "email": null, "sis_user_id": null, "short_name": "Casper Hedegaard Hansen", "sortable_name": "Hansen, Casper Hedegaard", "integration_id": null, "login_id": null, "unread_count": 0, "group_categories": []}, {"id": "36266", "name": "Morten Gedsted Hansen", "created_at": "2020-08-07T10:45:00+02:00", "email": null, "sis_user_id": null, "short_name": "Morten Gedsted Hansen", "sortable_name": "Hansen, Morten Gedsted", "integration_id": null, "login_id": null, "unread_count": 0, "group_categories": []}, {"id": "23980", "name": "Anders Husted", "created_at": "2020-08-07T10:45:00+02:00", "email": null, "sis_user_id": null, "short_name": "Anders Husted", "sortable_name": "Husted, Anders", "integration_id": null, "login_id": null, "unread_count": 0, "group_categories": []}, {"id": "23531", "name": "Søren Bo Jørgensen", "created_at": "2020-08-07T10:45:00+02:00", "email": null, "sis_user_id": null, "short_name": "Søren Bo Jørgensen", "sortable_name": "Jørgensen, Søren Bo", "integration_id": null, "login_id": null, "unread_count": 0, "group_categories": []}]
```

6 / 157 requests

# Objects? Arrays?

The screenshot shows the DR Nyheder - Breaking - TV website. At the top, there are six thumbnail images with titles: DR1: Løvens Hule, DR3: Nationens stærkste, P1: LSD kælderen, DR LYD: Annas Margrethe, DR3: Du fucker med de forkerte, and A Very British Scandal. Below this is a section titled "Seneste nyt" (Latest news) with three 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 graphic of a face wearing a mask, with various items like a test kit, a bottle, and a hand sanitizer placed around it. Below the graphic, a headline reads "15 lande bakker Danmark op: Danske soldater skal blive i Mali". At the bottom, a red banner says "Liveblog" with three dots.

The screenshot shows the Erhvervsakademietuddannelse website. The header includes the logo "ERHVERVSAKADEMI AARHUS" and language options "DA / INT". The page displays eight vocational training programs in a grid format:

- Byggekoordinator (lukket)**: 2-årig erhvervsakademiuddannelse. Uddannelsen optager ikke nye studerende fra og med 2022.
- Datamatiker**: 2-årig erhvervsakademiuddannelse. Få en bred viden inden for systemudvikling og programmering, og bliv klar til et job med udvikling, implementering og drift af IT-systemer i virksomheden.
- Financial controller**: 2-årig erhvervsakademiuddannelse. Bliv klar til et job i en økonomiafdeling eller som revisor. Du gør karriere inden for fx regnskab, likviditetsstyring, årsrapporter, skat, moms og husholdningen.
- Finansøkonom**: 2-årig erhvervsakademiuddannelse. Bliv rustet til en karriere i investerings-, forsikrings- eller ejendomsbranchen - eller i kreditforsørger. Du bliver en god rådgiver, der løser kundebehov med din virksomheds ressourcer.
- It-teknolog**: 2-årig erhvervsakademiuddannelse. Brænder du for at følge med i udviklingen af den nyeste teknologi? Så gør karriere inden for computer-, server- og netværksteknologi eller elektronik.
- Jordbrugsteknolog**: 2-årig erhvervsakademiuddannelse. Bliv specialist inden for miljø og natur, jordbrugsekonomi og driftsledelse, husdyrproduktion, landskab og anlæg og planteproduktion.
- Laborant**: 2½-årig erhvervsakademiuddannelse. Få job i et kontrol-, forsknings- eller udviklingslaboratorium og arbejd inden for fx fødevare sikkerhed, miljøbeskyttelse, medicinalindustrien eller bioteknologi.
- Markedsføringsøkonom**: 2-årig erhvervsakademiuddannelse. For dig, der vil arbejde med markedsføring, salg og kommunikation. Gør karriere som fx indkøber, sæger, marketingkoordinator eller projektleder. Vi tilbyder specialiseringer inden for helsekønns design.
- Miljøteknolog**: 2-årig erhvervsakademiuddannelse. For dig, der vil arbejde med kvalitetskontrol og forbedring af virksomhedens miljøindsats. Som miljøteknolog sikrer du, at virksomheden er beredvilligt mindsker forurenningen om overhinder.
- Multimediedesigner**: 2-årig erhvervsakademiuddannelse. Få viden om user interface design (UI), user experience design (UX), programmering og formændsforståelse. Bliv klar til job som fx frontendumulværker UX-designer, minid manager eller

# 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

# Computer science student



## Senior developer, 10+ years experience



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

# Arrays

## Loops

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

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

# Loops

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

**Loops: while and for**

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

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

A small announcement for advanced readers.

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

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

- See `for...in` to loop over object properties.
- See `for...of` and `iterables` for looping over arrays and iterable objects.

Otherwise, please read on.

# For of loop

iterate over arrays or other iterable objects

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

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

# Loops

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

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

[https://www.w3schools.com/js/js\\_loop\\_for.asp](https://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

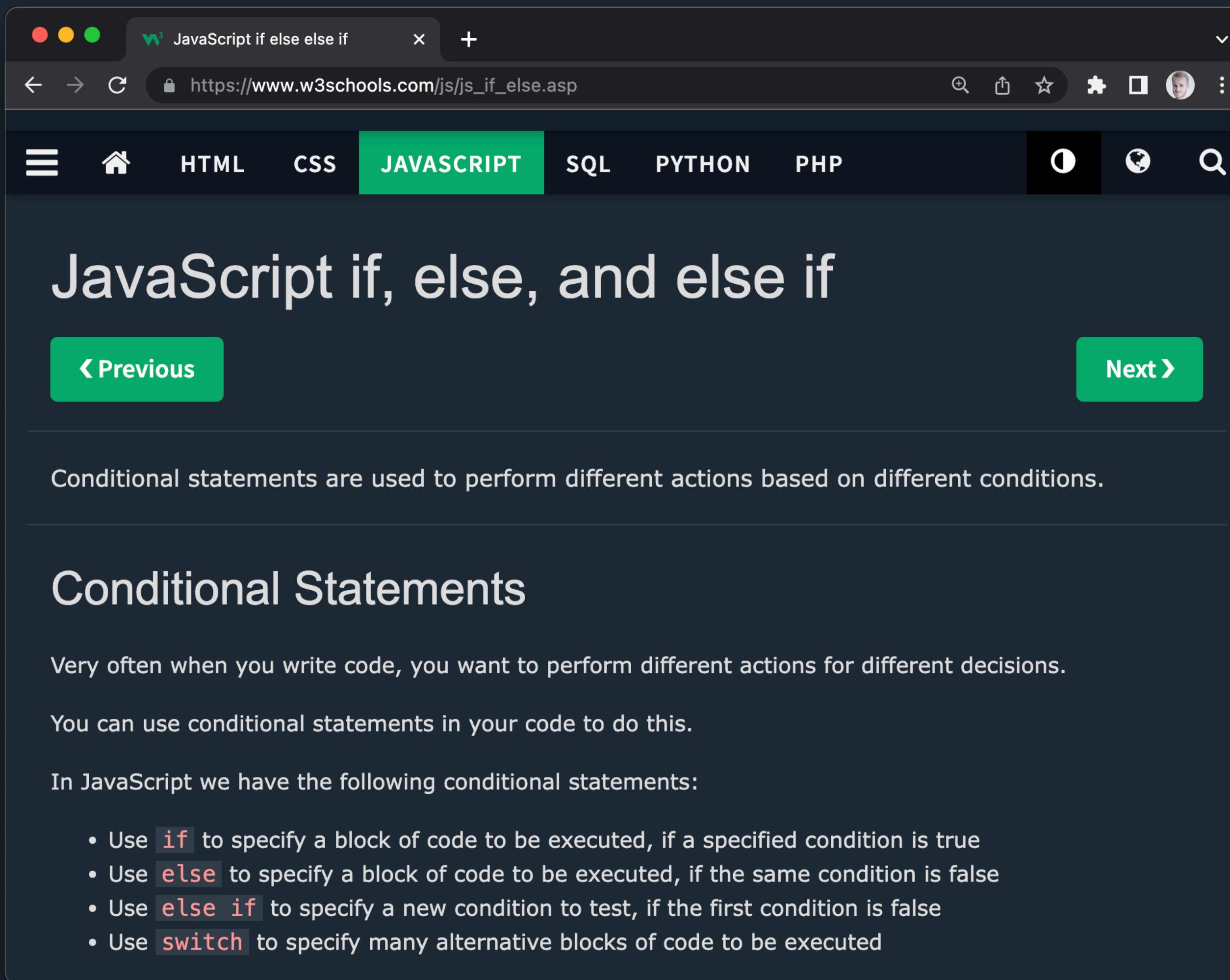
main.js:20

# LOOPS

... LOOP THROUGH AN ARRAY AND ADD A CONDITION

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

# Conditional Statements



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)

# `==` VS `== =`

`== =` IS MORE STRICT

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

```
"32" == 32 is true
"32" == = 32 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";
}
```

# Array Methods to know

.push (...)

.map (...)

.filter (...)

.find (...)

.sort (...)

Edit src/App.js and save to reload  
Learn React

# 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

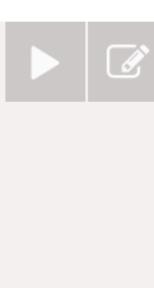
The `find` method looks for a single (first) element that makes the function return `true`. If there may be many, we can use `arr.filter(fn)`.

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
```

# `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>;
  }
}
```

# 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

## 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: `{}`

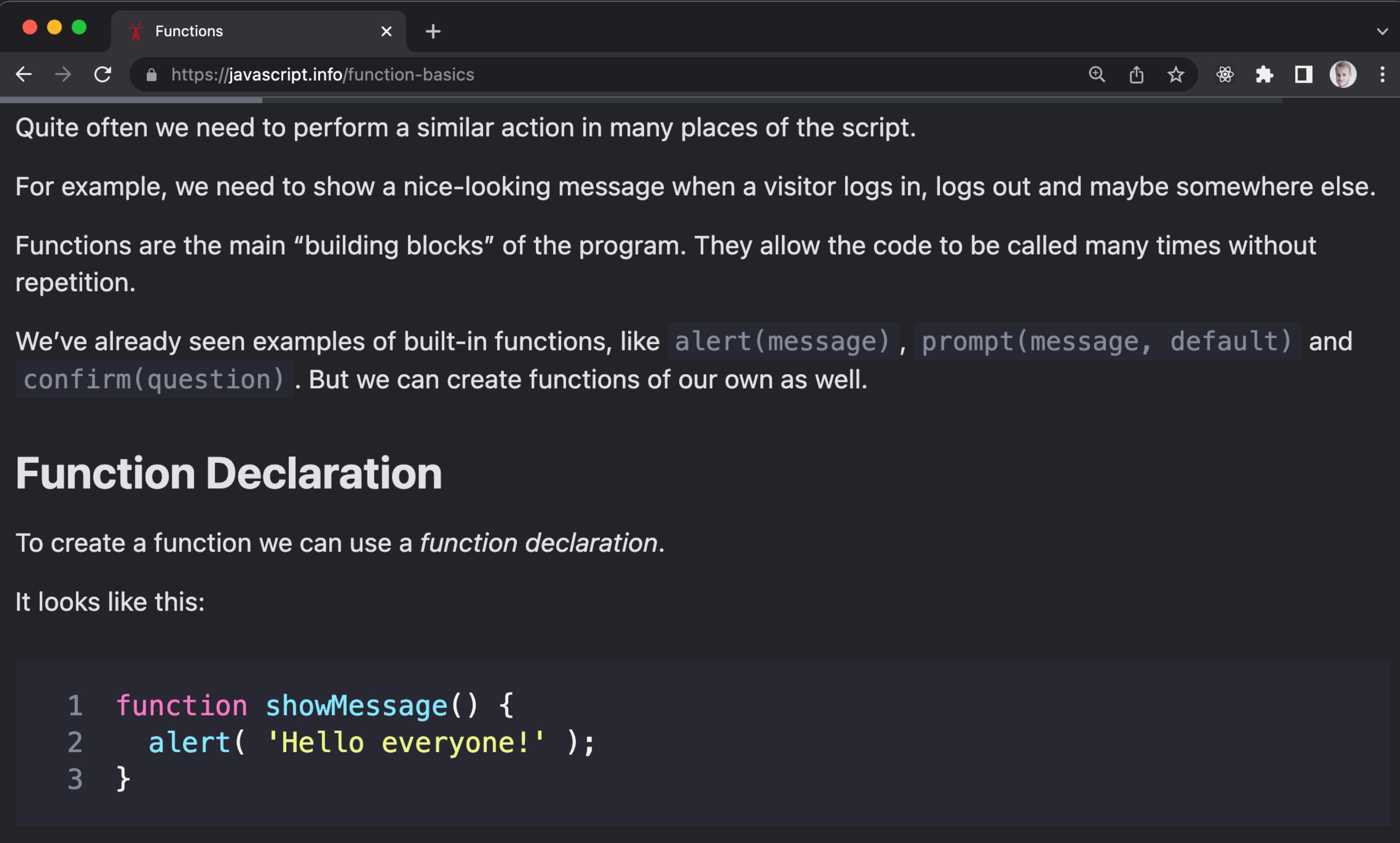
```
function name(parameter1, parameter2, parameter3) {  
    // code to be executed  
}
```

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

```
function append(htmlTemplate, idOfElement) {  
    console.log(htmlTemplate);  
    document.getElementById(idOfElement).innerHTML += htmlTemplate;  
    alert("Yaaaaah, you did it!");  
}  
  
append("<h2>Hi Frontenders!</h2>", "content");
```

The name of the function

Parameters

Body of the function  
(code block)

How to call the function

The diagram illustrates the structure of a JavaScript function declaration. It features a central code block with arrows pointing to its various components. An arrow from the text 'The name of the function' points to the word 'append'. Another arrow from 'Parameters' points to the two parameters 'htmlTemplate' and 'idOfElement' listed in the parentheses. A large bracket on the right side of the code block is labeled 'Body of the function (code block)'. An arrow from the text 'How to call the function' points to the line of code 'append("...")'.

# 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)

# Functions

3 different types

```
function logPersons(persons) {  
  for (var i = 0; i < persons.length; i++) {  
    console.log(persons[i]);  
  }  
}
```

FUNCTION DECLARATION

```
const logPersons = function(persons) {  
  for (var i = 0; i < persons.length; i++) {  
    console.log(persons[i]);  
  }  
}
```

FUNCTION EXPRESSION

```
const logPersons = (persons) => {  
  for (var i = 0; i < persons.length; i++) {  
    console.log(persons[i]);  
  }  
}
```

ARROW FUNCTION

# 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? (**)
```

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

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
```

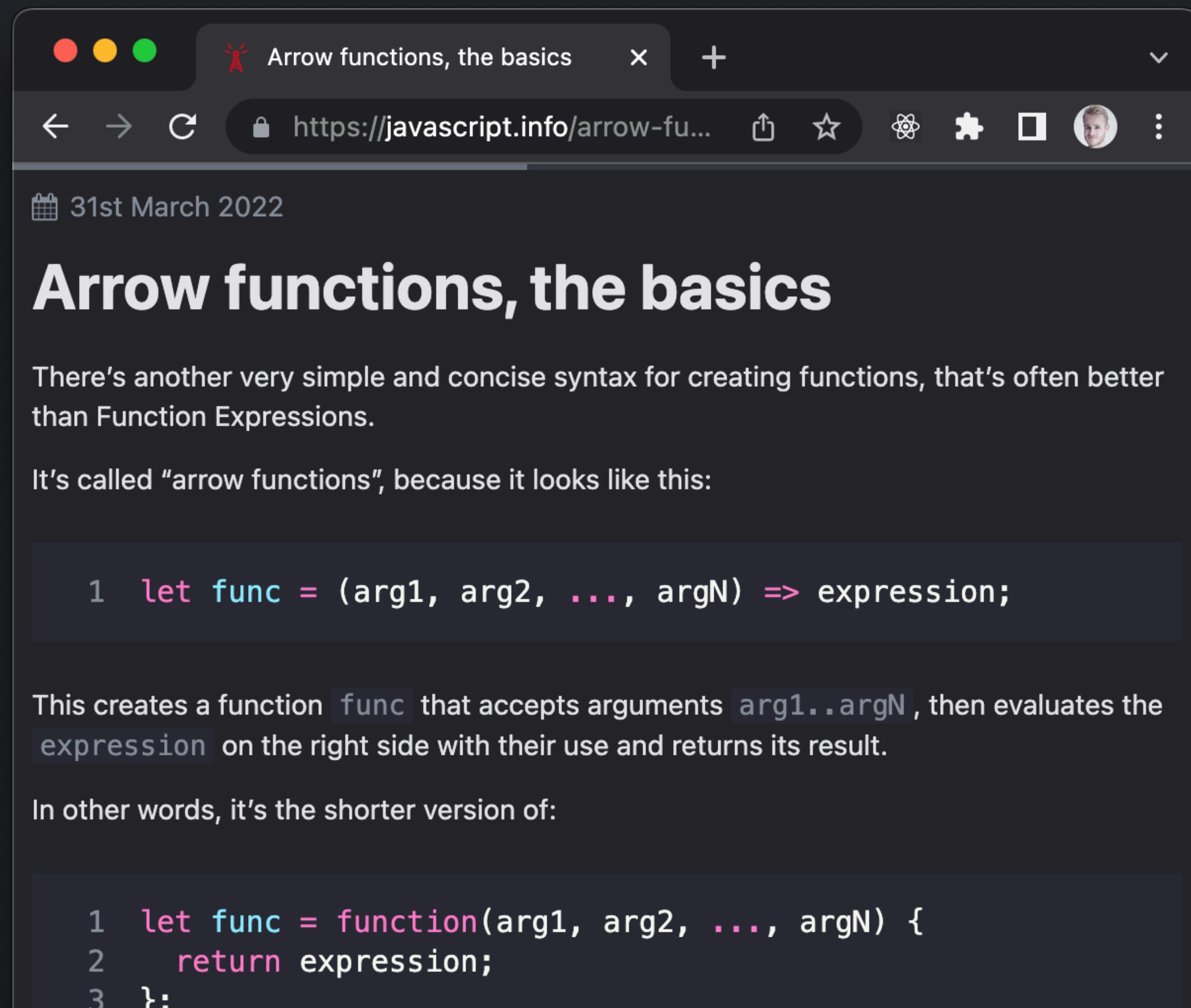
# Function called by another function

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(value) {  
    value = value.toLowerCase();  
    let filteredTeachers = [];  
    for (let teacher of teachers) {  
        let name = teacher.name.toLowerCase();  
        if (name.includes(value)) {  
            filteredTeachers.push(teacher);  
        }  
    }  
    appendTeachers(filteredTeachers);  
}
```

FOR OF LOOP

## .FILTER & ARROW FUNCTION

```
function search(value) {  
    let searchValue = value.toLowerCase();  
    let filteredTeachers = _teachers.filter(teacher => teacher.title.rendered.toLowerCase().includes(searchValue));  
    appendTeachers(filteredTeachers);  
}
```

# 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.

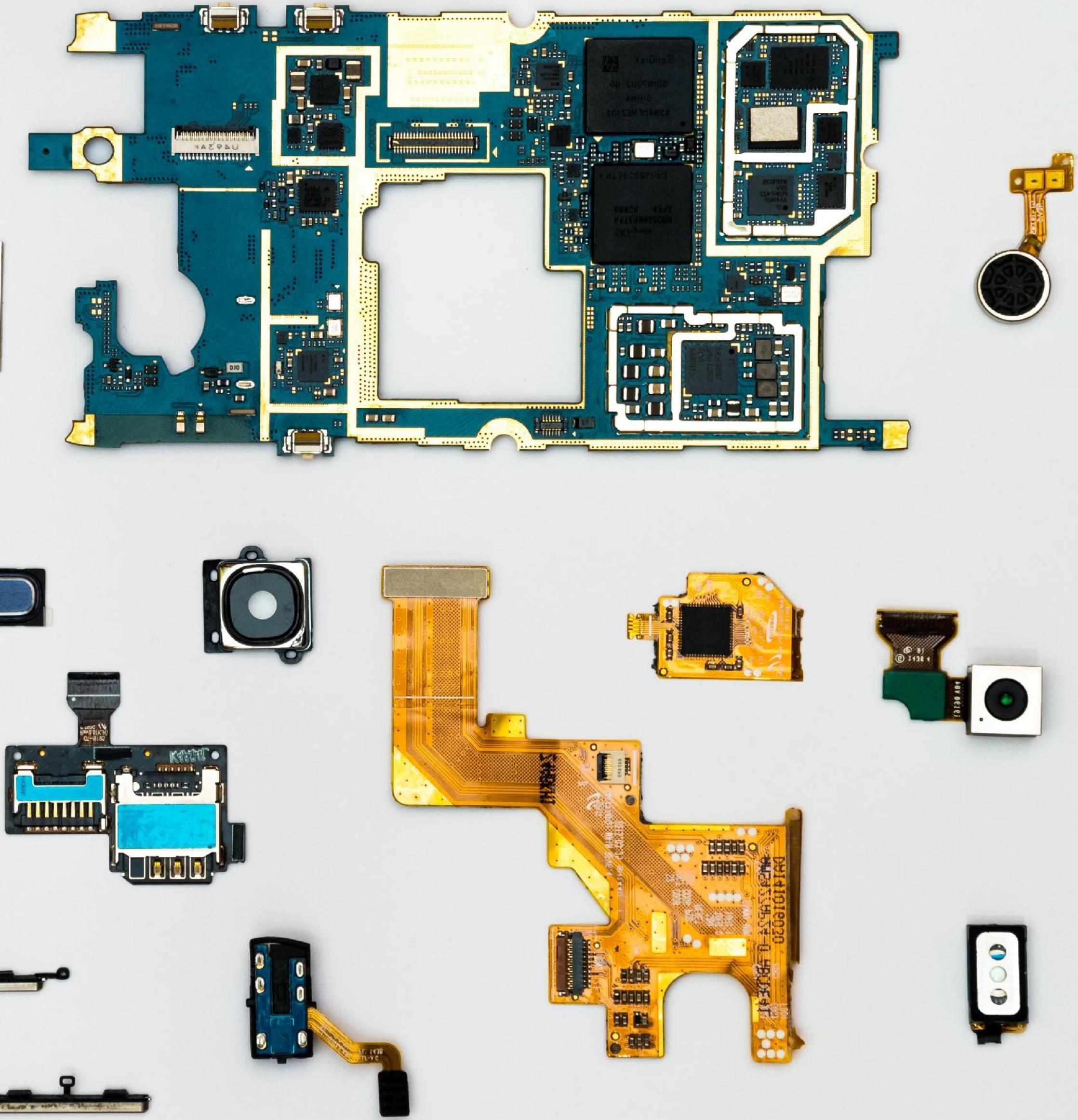
Export and import a const:

```
// 📄 user.js
```

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

```
// 📄 app.js
```

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



# Modules

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

One script is one  
module.

```
// └─ 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)

# 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 ...

# EXPORT => IMPORT

**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)

```
// 📁 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');
```

```
// 📁 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');
```

```
JS main.js ... JS user.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-expo"),
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 }
32
33 
```

# TELL THE BROWSER IT'S A MODULE

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



Code  
Every  
Day