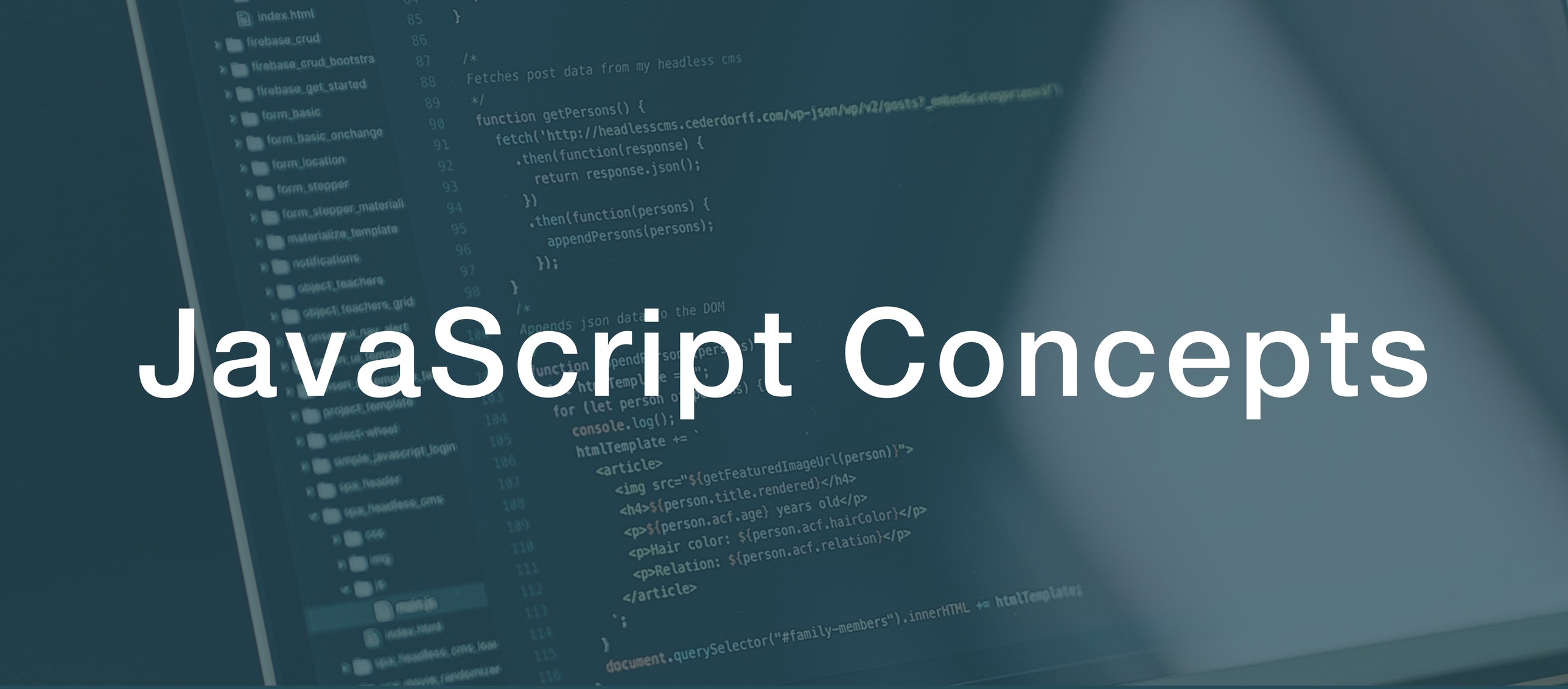


JavaScript Concepts

Web Development

ERHVERVSAKADEMI AARHUS
BUSINESS ACADEMY AARHUS

Content


Programming knowledge
JavaScript

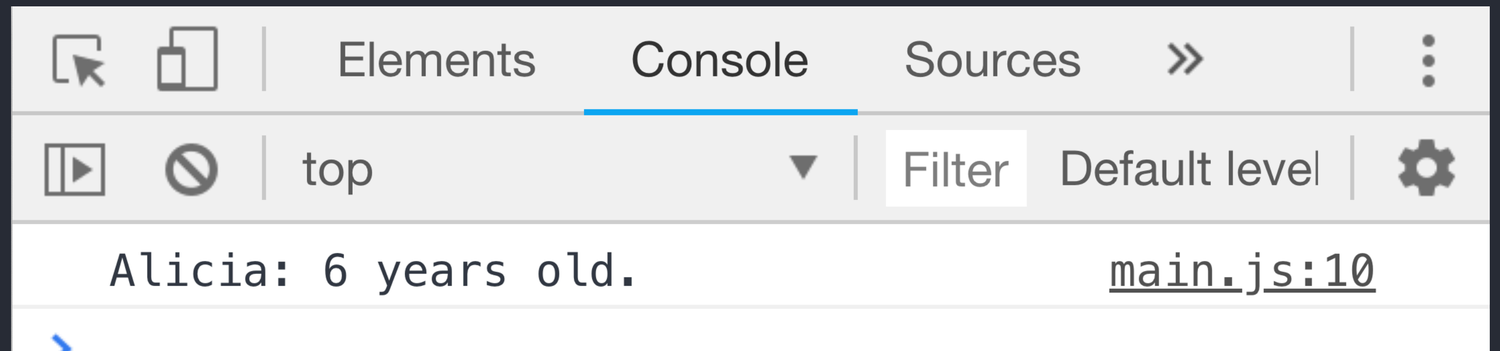
Variables

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

Variables

Store data in the memory

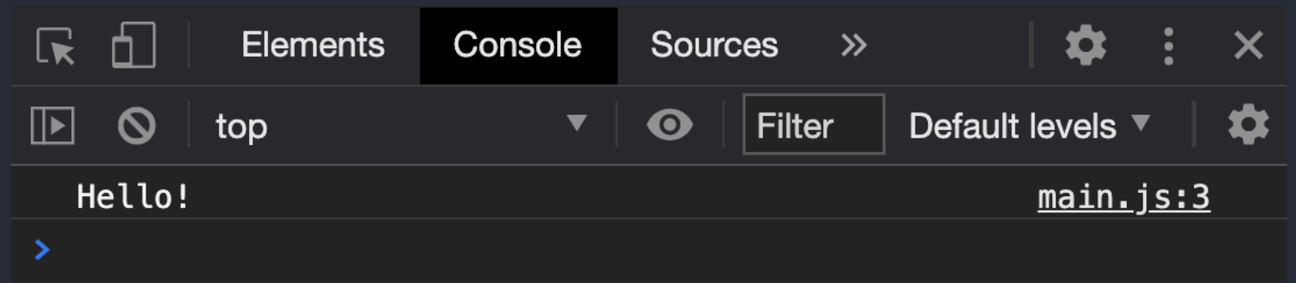
```
let name = "Alicia";
let age = 6;

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



Variables

Store data in the memory

```
let message = "Hello!";

console.log(message);
```



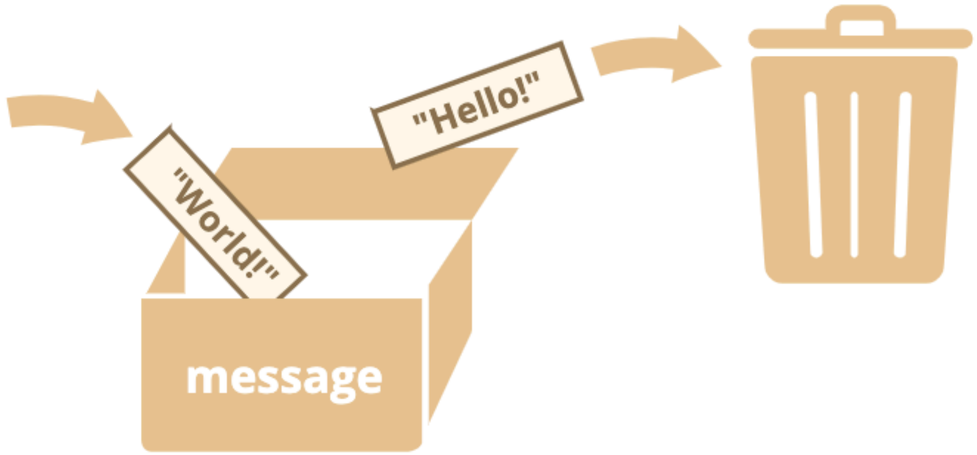
Variables

Store data in the memory

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

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

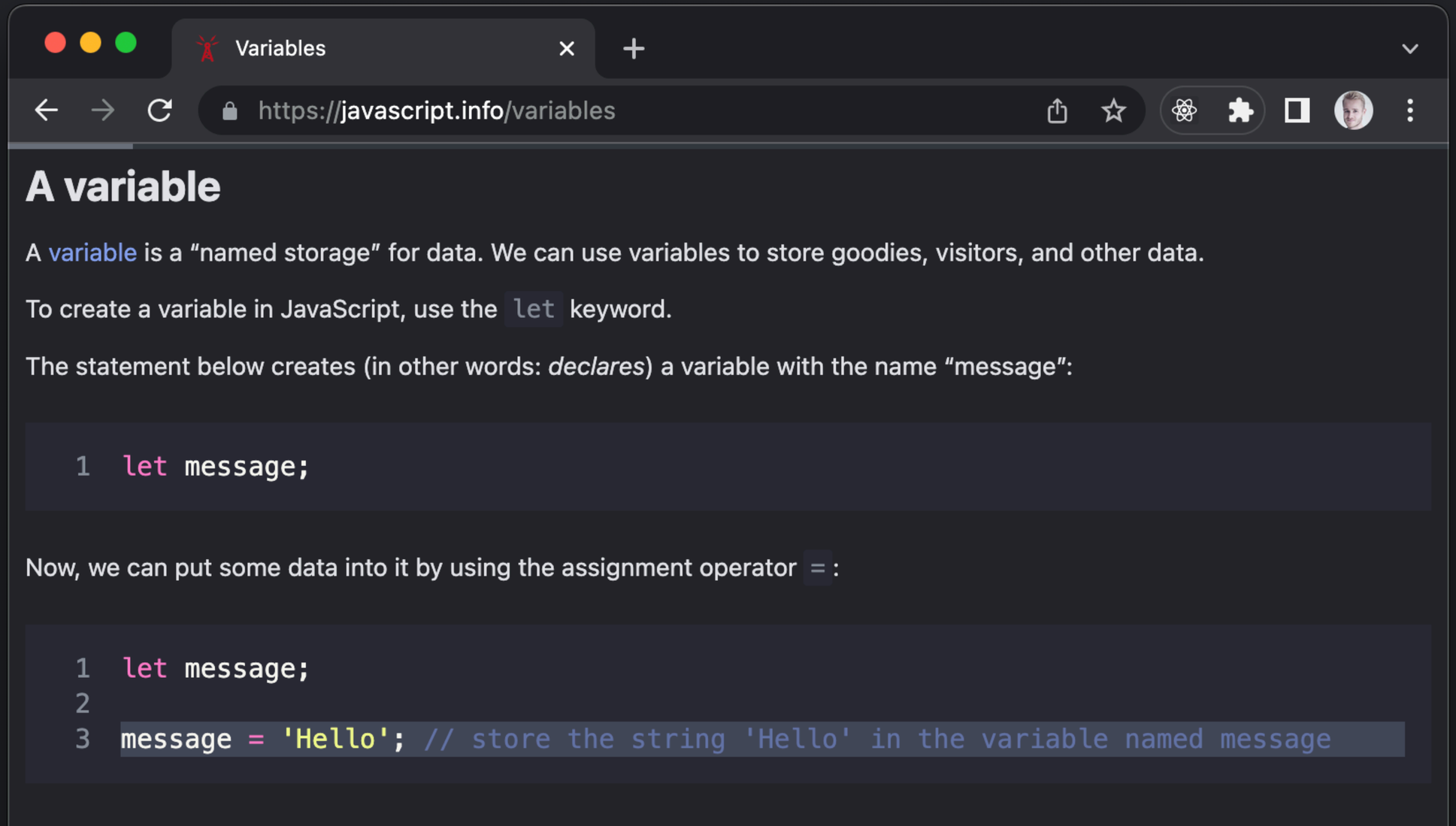
Variable

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



var **vs** let

THE DIFFERENCE IS THE SCOPING

VAR IS FUNCTION-WIDE OR GLOBAL SCOPE

LET IS BLOCK SCOPED

VAR TOLERATES REDECLARATION

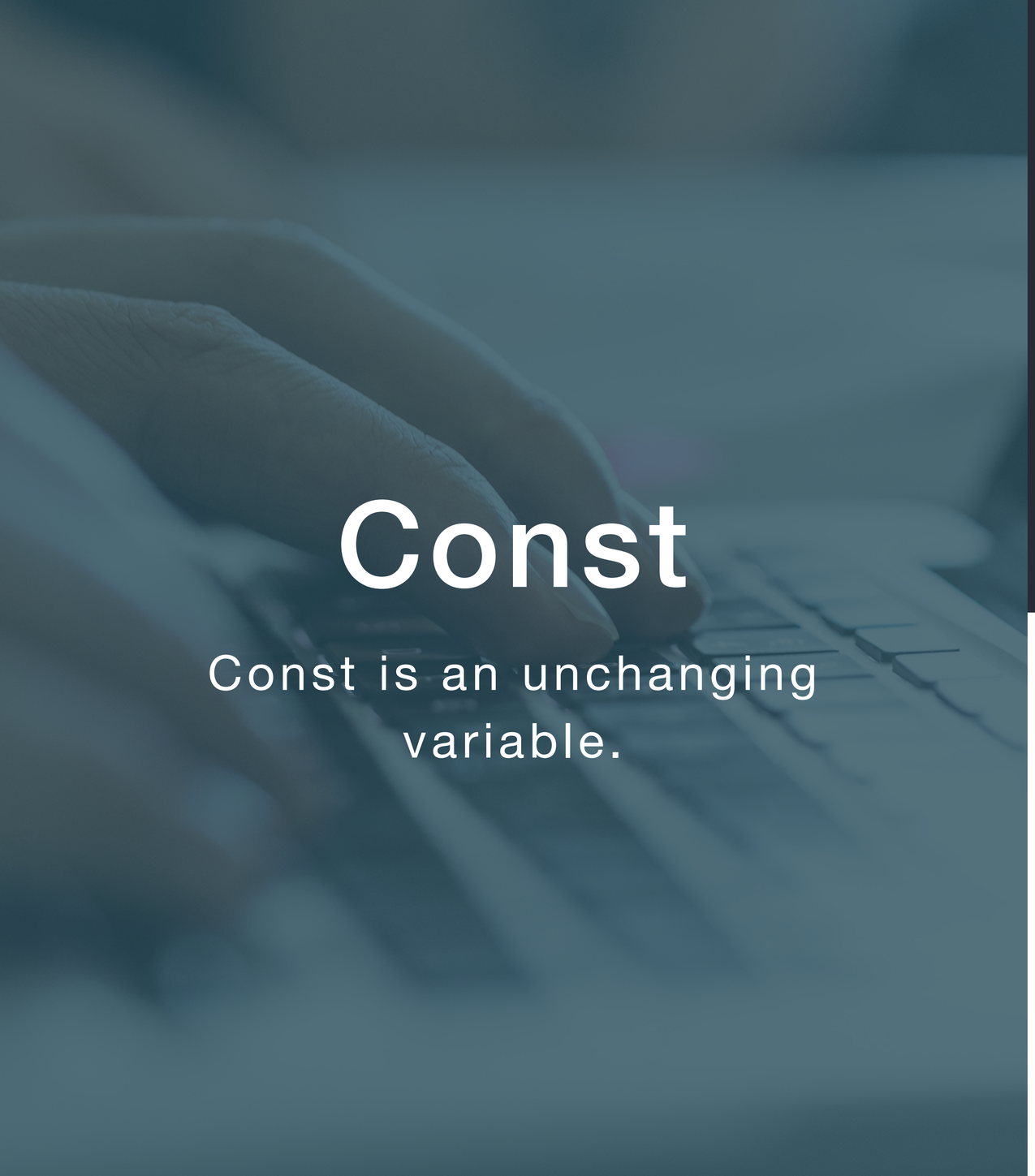
https://javascript.info/variables
https://javascript.info/var

```
// 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-vs-let

```
// “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`.

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

Data Types

In JavaScript there are two main data types:

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

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

```
let n = 123;
n = 12.345;
```

```
let user = new Object(); // "object cons
let user = {};  // "object literal" synt
```

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

Objects

A set of named values

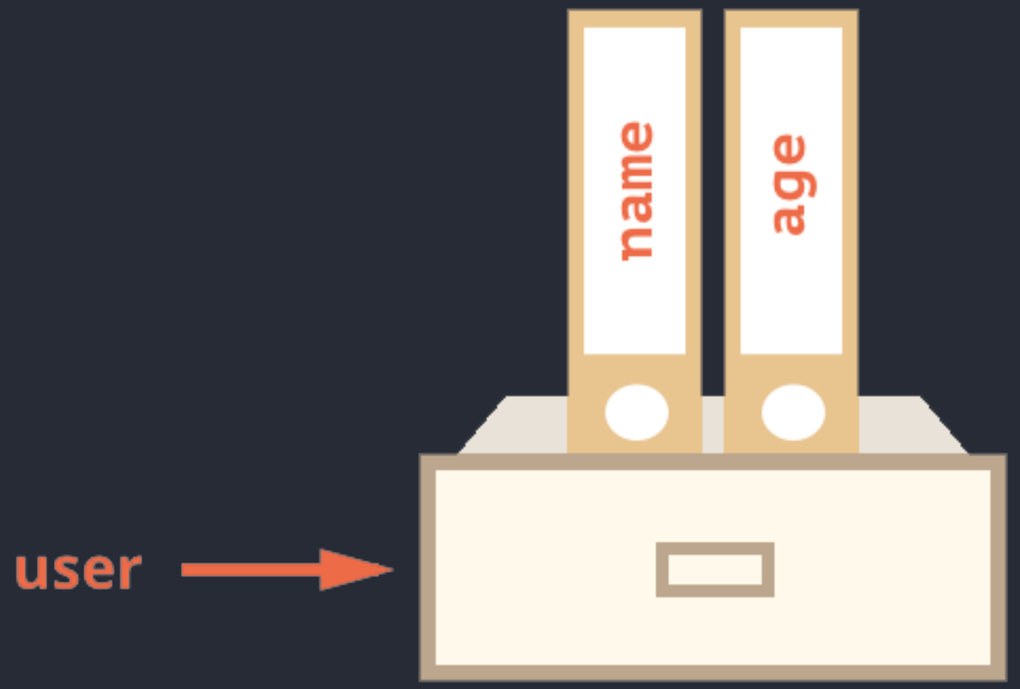
Objects are used to store keyed collections of various data

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

Objects

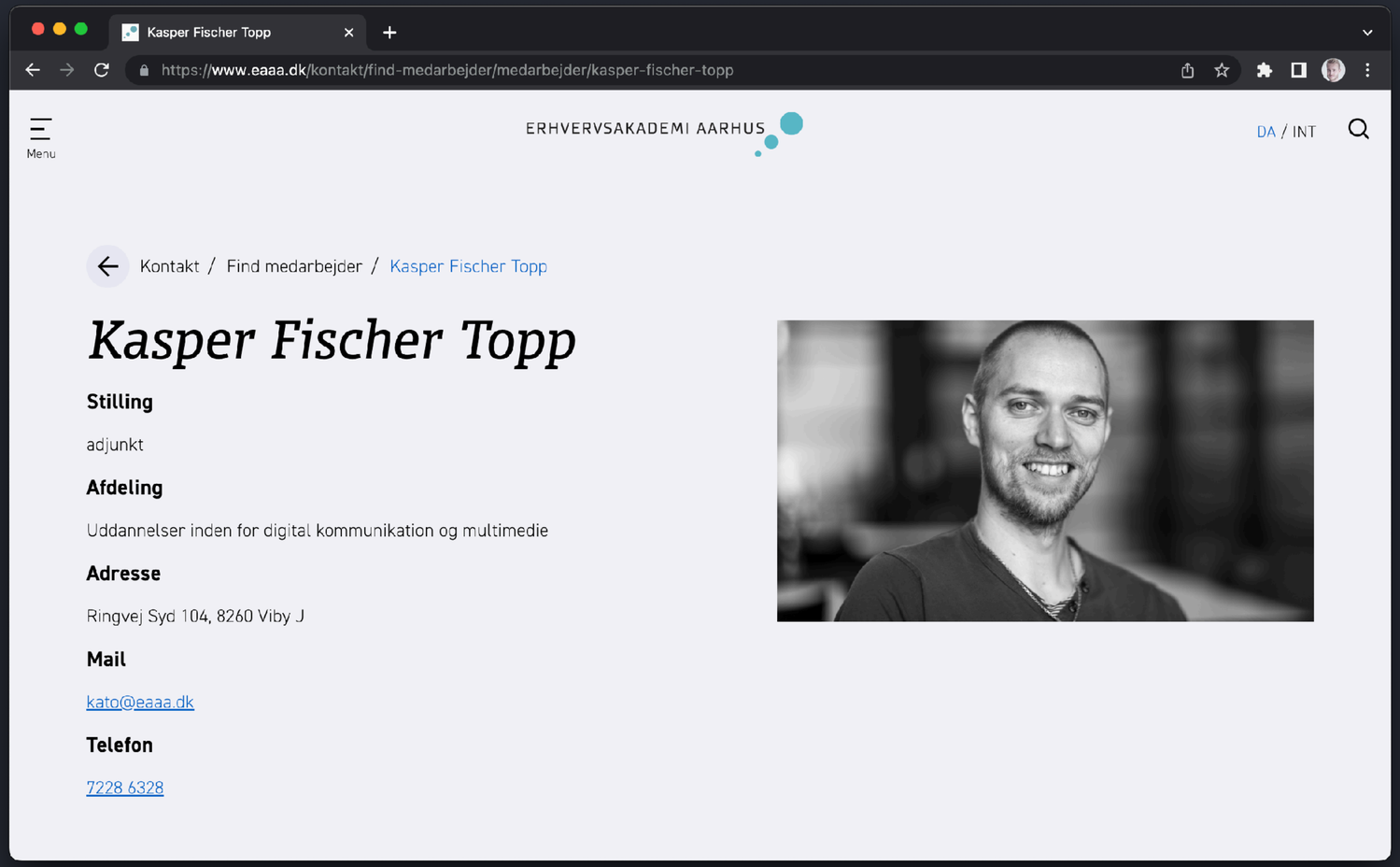
A set of named values

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

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




```
Alicia is 6 years old.          main.js:11
>
```

Objects



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.com

NETFLIX

Hvem ser?

Personen der rent faktisk betaler for profilen

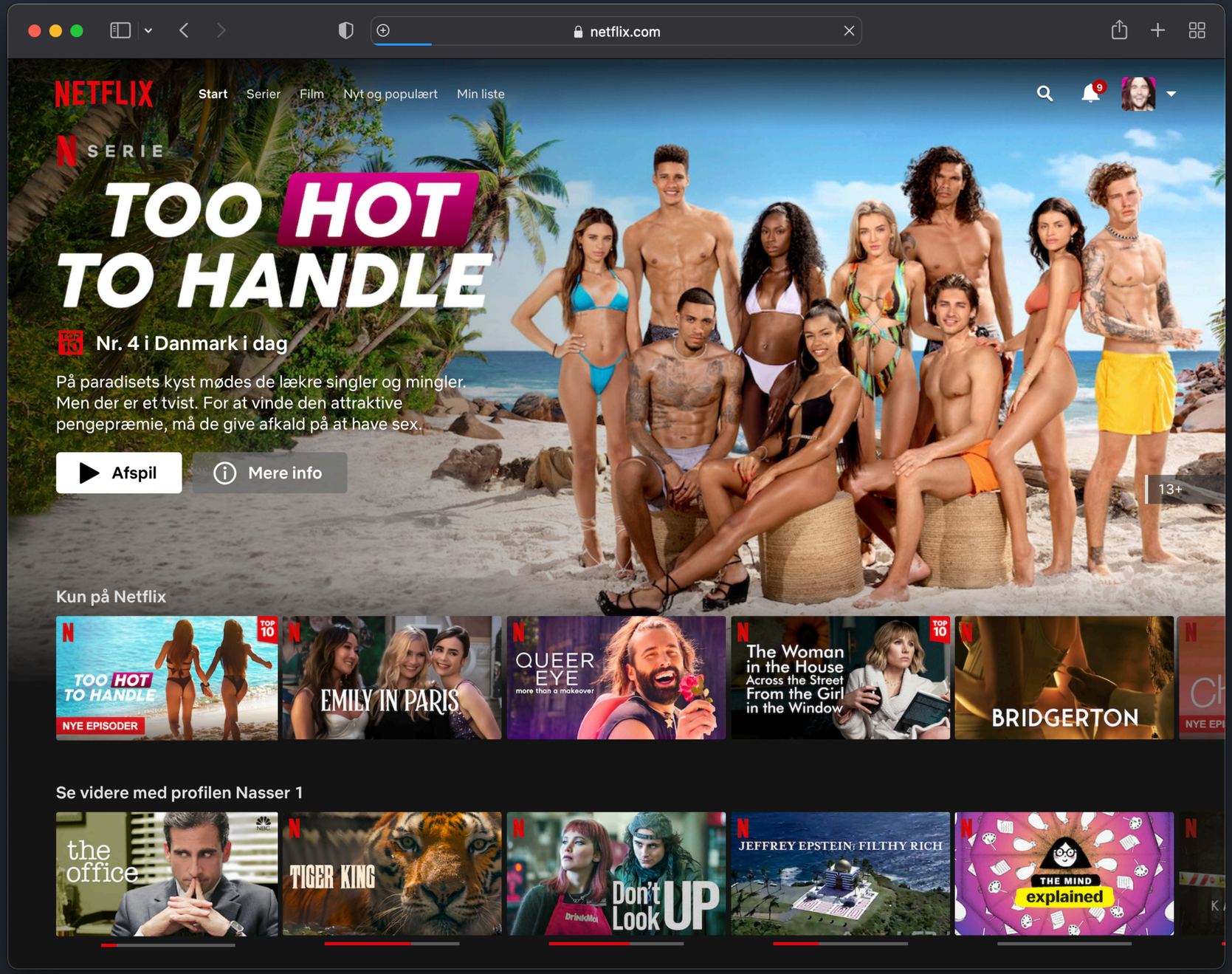
Nasser 1

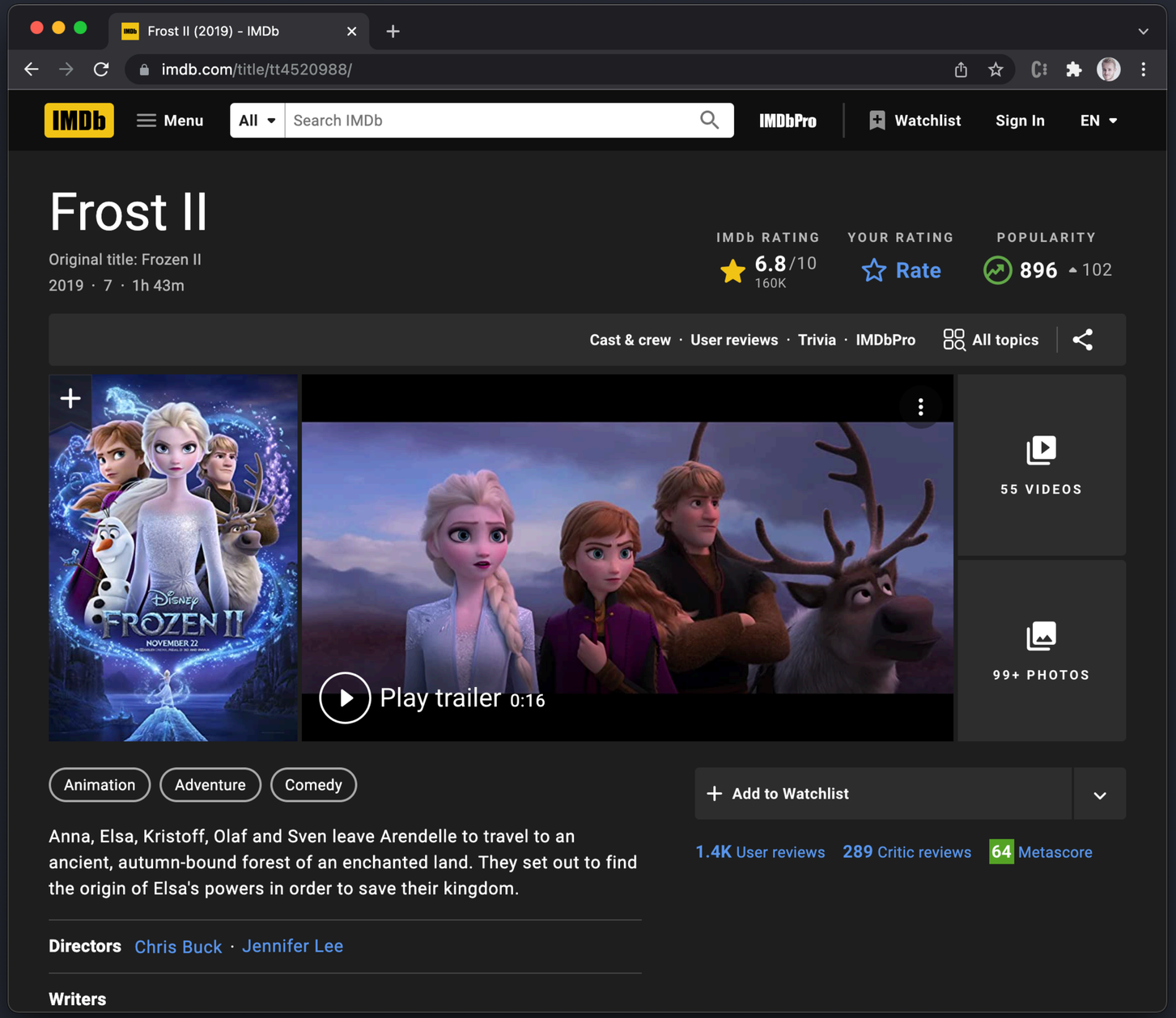
Nasser 2

Nasser 3

Nasser 4 Khader

Administrer profiler


netflix.com
NETFLIX
Start
Serier
Film
Nyt og populært
Min liste
SERIE
TOO HOT TO HANDLE
Nr. 4 i Danmark i dag
På paradisets kyst mødes de lækre 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
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
Se videre med profilen Nasser 1
the office
TIGER KING
Don't Look UP
JEFFREY EPSTEIN: FILTHY RICH
THE MIND
explained

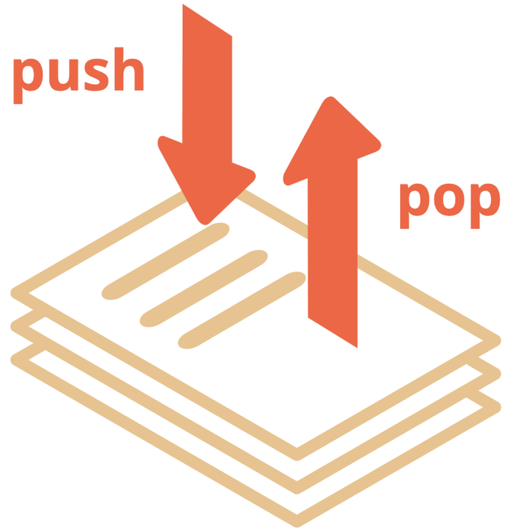



```
let movie = {
  title: "Frozen 2",
  description: "Elsa the Snow Queen has a
  trailer: "https://www.youtube.com/embed
  length: "1h 43m",
  year: "2019"
}
```

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

Position:
Lecturer

Department/p
Multimedia De
Digital Concep

Address:
Ringvej Syd 1(

Mail:
race@baaa.dk

Phone:
7228 6318

Michael Hvidtfeldt

Position:
Lecturer

Department/p
Multimedia De

Address:
Ringvej Syd 1(

Mail:
mhv@baaa.dk

Phone:
7228 6328

Birgitte Kirk Iversen

Position:
Senior Lecturer

Department/programme:
Multimedia Design

Address:
Sønderhøj 30, 8260 Viby J

Mail:
bki@baaa.dk

Phone:
7228 6316

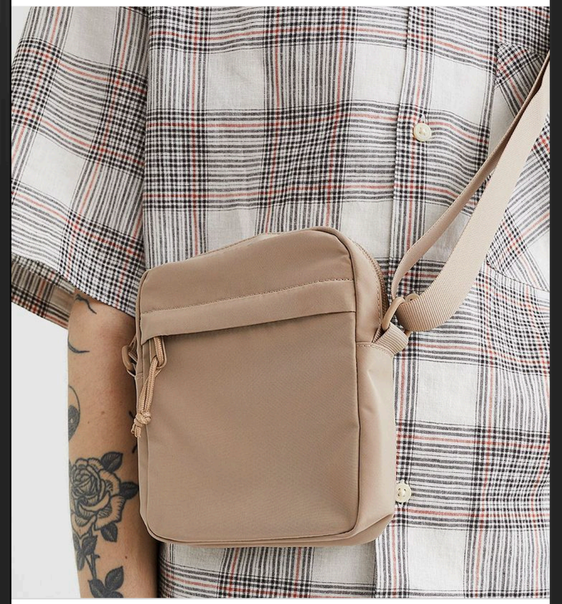
```
main.js:21
▼ (3) [{…}, {…}, {…}] i
  ▶ 0: {name: "Birgitte Kirk Iversen", mail: "bki@baaa…
  ▶ 1: {name: "Michael Hvidtfeldt", mail: "mhv@baaa.dk…
  ▶ 2: {name: "Rasmus Cederdorff", mail: "race@baaa.dk…
    length: 3
  ▶ __proto__: Array(0)
main.js:22
▶ {name: "Michael Hvidtfeldt", mail: "mhv@baaa.dk"}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

Instagram

Your story | fhergram | vladaafa... | kristybra... | gabriel

hm_man



Instagram

Liked by **praaajyot** and **others**

hm_man Shades, style and summer vibes, they all fit inside. #HMMan... more

View all 5 comments

44 MINUTES AGO

programmerhumor_io

I Am Devloper
@iamdevloper

Elon Musk: I'm putting people on Mars!
Developers: Fantastic, more timezones to support

cederdorff

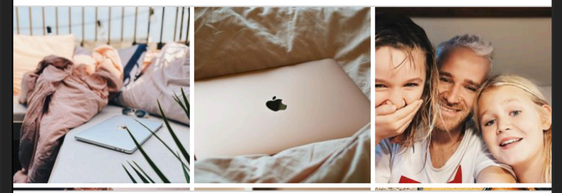
cederdorff

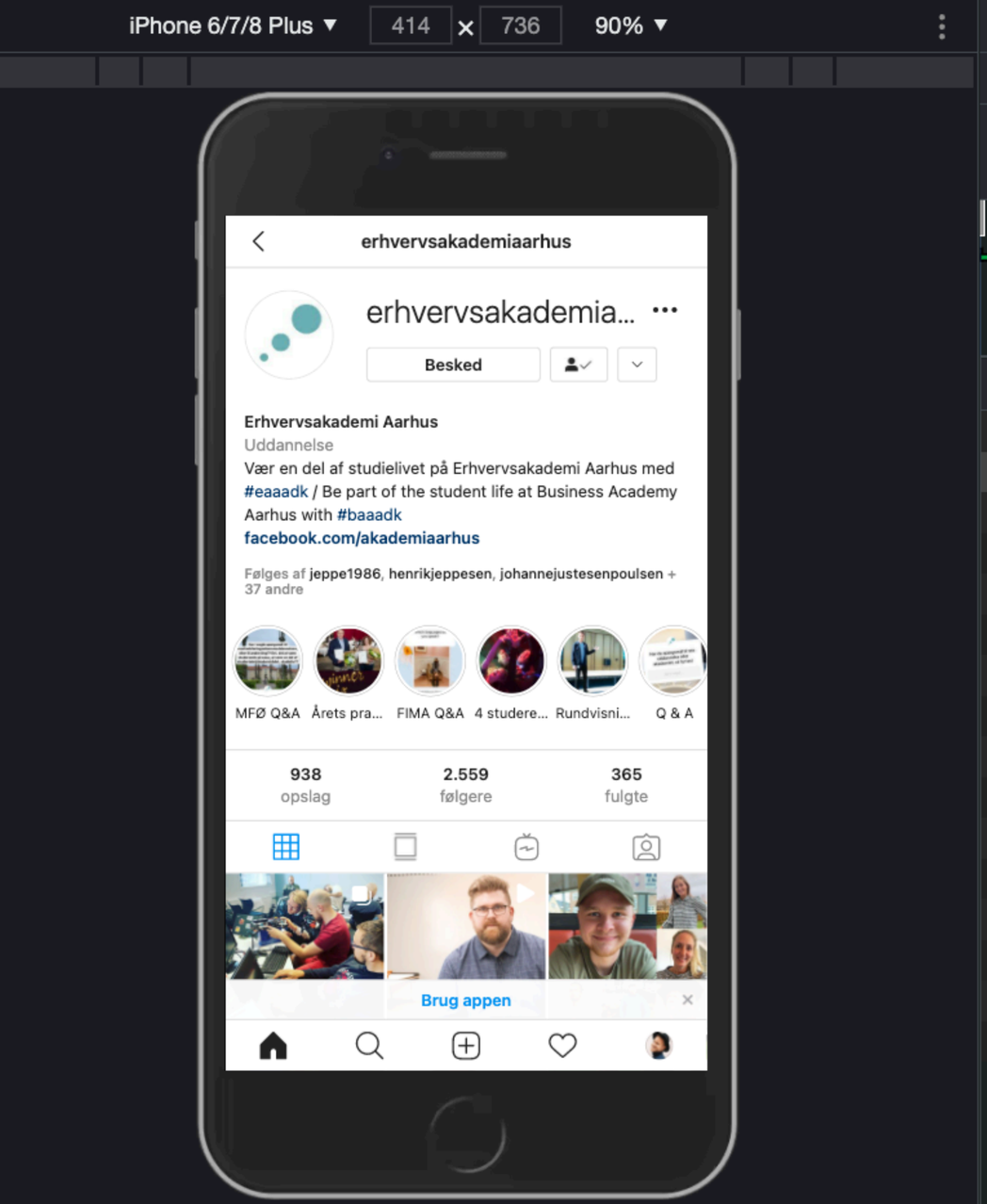
Edit Profile

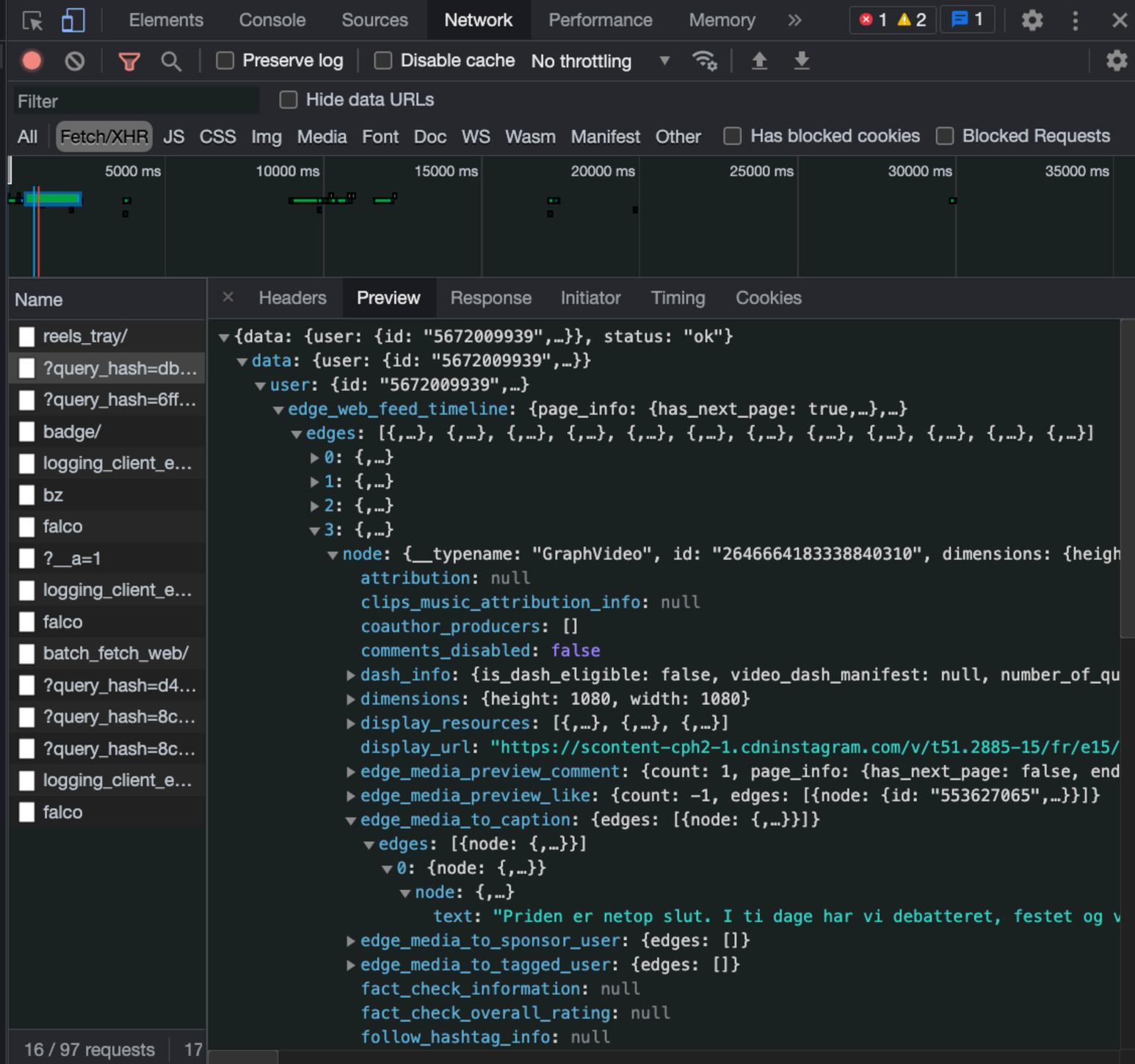
Rasmus Cederdorff
Senior Lecturer & Web App Developer
Alicia & Ida
Aarhus, Denmark
cederdorff.com

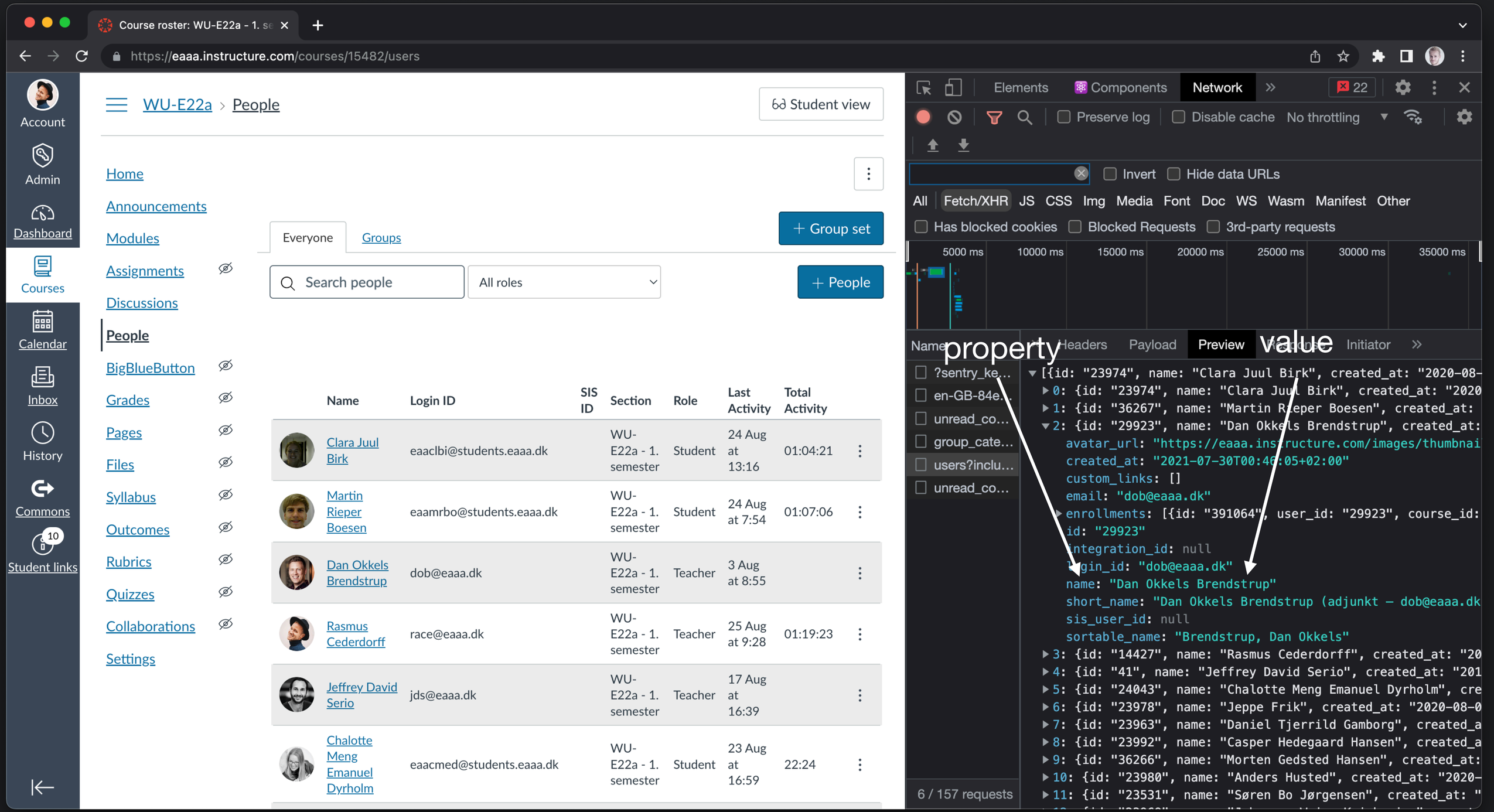
AMSTER... | LONDON ... | Karla & Pi... | @villacph | IDA

181	5,441	635
posts	followers	following



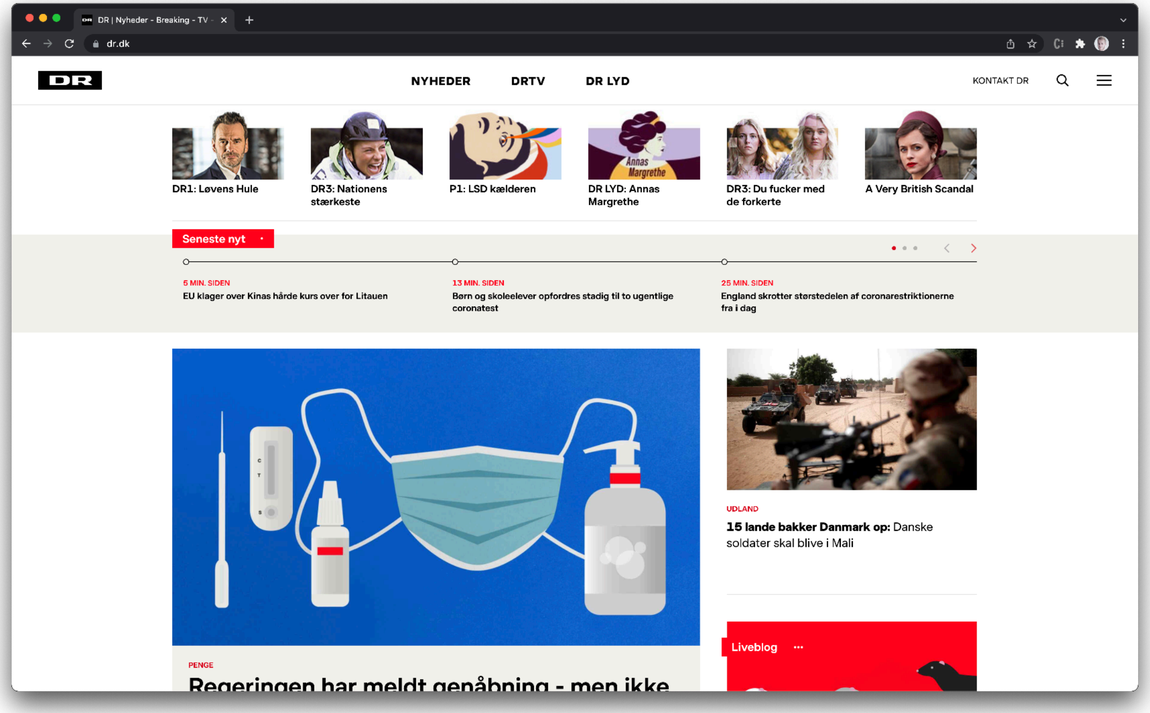

iPhone 6/7/8 Plus
414
736
90%
erhvervsakademiaarhus
erhvervsakademia...
Besked
Erhvervsakademi Aarhus
Uddannelse
Vær en del af studielivet på Erhvervsakademi Aarhus med #eaaadk / Be part of the student life at Business Academy Aarhus with #baaadk
facebook.com/akademiaarhus
Følges af jeppe1986, henrikjeppesen, johannejustesenpoulsen + 37 andre
MFØ Q&A
Årets pra...
FIMA Q&A
4 studere...
Rundvisni...
Q & A
938
opslag
2.559
følgere
365
fulgte
Brug appen

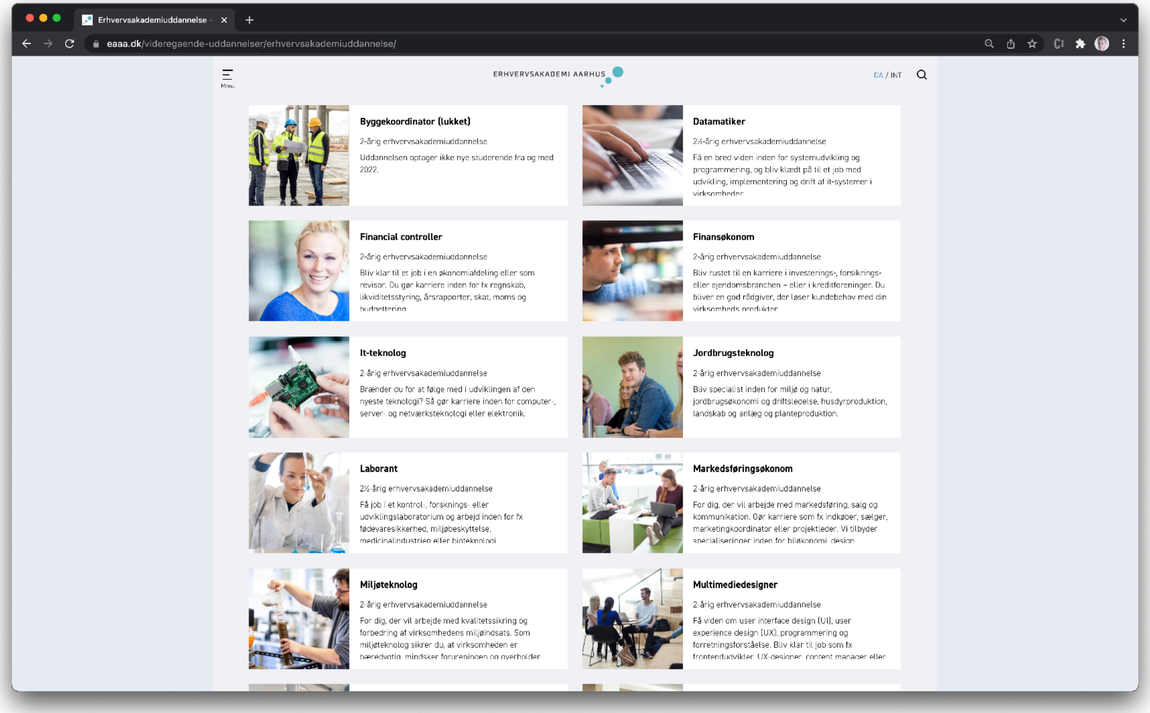

Elements
Console
Sources
Network
Performance
Memory
Preserve log
Disable cache
No throttling
Filter
Hide data URLs
All
Fetch/XHR
JS
CSS
Img
Media
Font
Doc
WS
Wasm
Manifest
Other
Has blocked cookies
Blocked Requests
5000 ms
10000 ms
15000 ms
20000 ms
25000 ms
30000 ms
35000 ms
Name
reels_tray/
?query_hash=db...
?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
16 / 97 requests
17
Headers
Preview
Response
Initiator
Timing
Cookies
{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: {heigh
attribution: null
clips_music_attribution_info: null
coauthor_producers: []
comments_disabled: false
dash_info: {is_dash_eligible: false, video_dash_manifest: null, number_of_qu
dimensions: {height: 1080, width: 1080}
display_resources: [{,…}, {,…}, {,…}]
display_url: "https://scontent-cph2-1.cdninstagram.com/v/t51.2885-15/fr/e15/
edge_media_preview_comment: {count: 1, page_info: {has_next_page: false, end
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



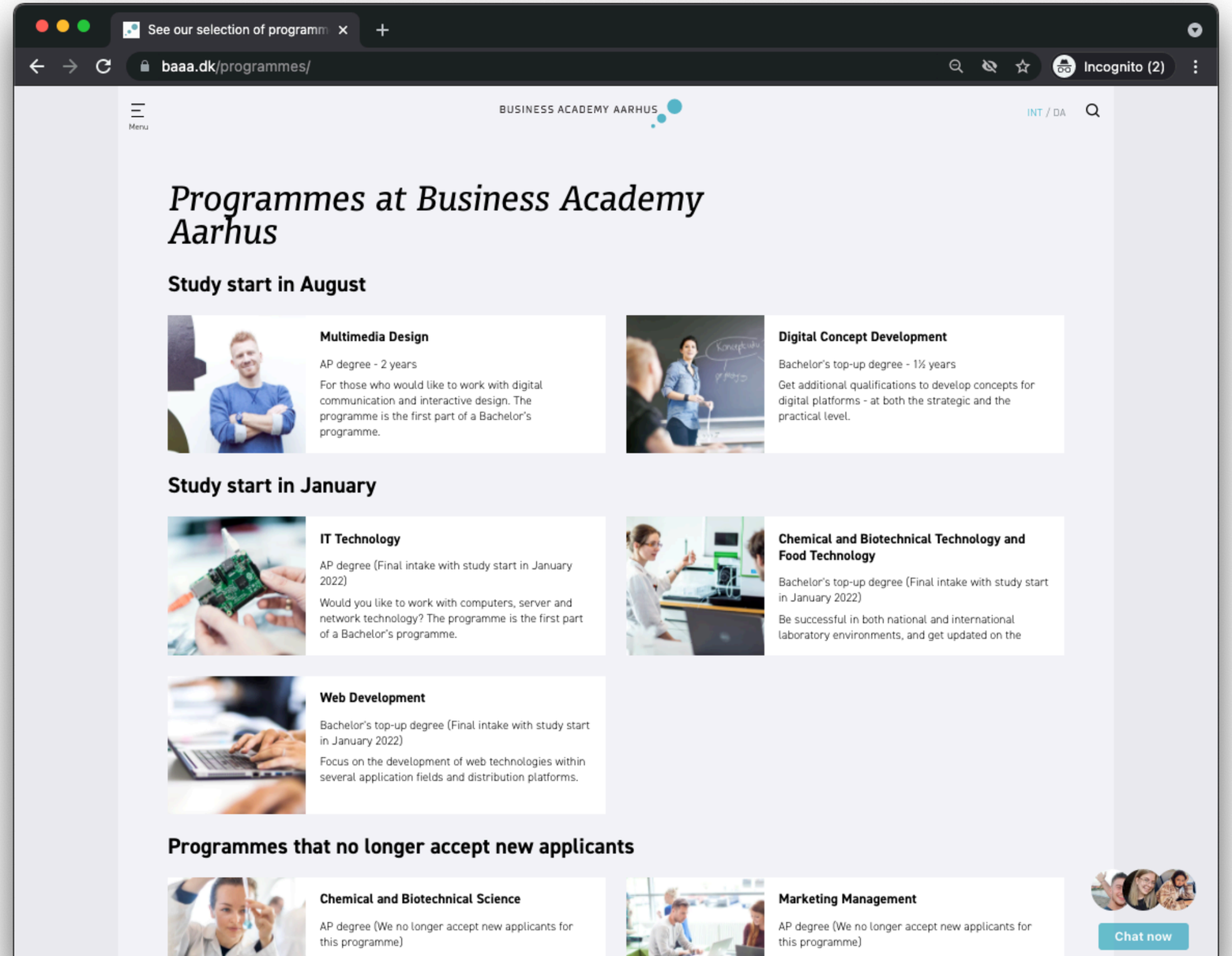
eaaa.instructure.com/api/v1/courses/15482/users?include_inactive=true&include%5B%5D=avatar_url&include%5B%5D=email&include%5B%5D=observed_users&per_page=50

Objects? Arrays?





Objects with properties in arrays



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/

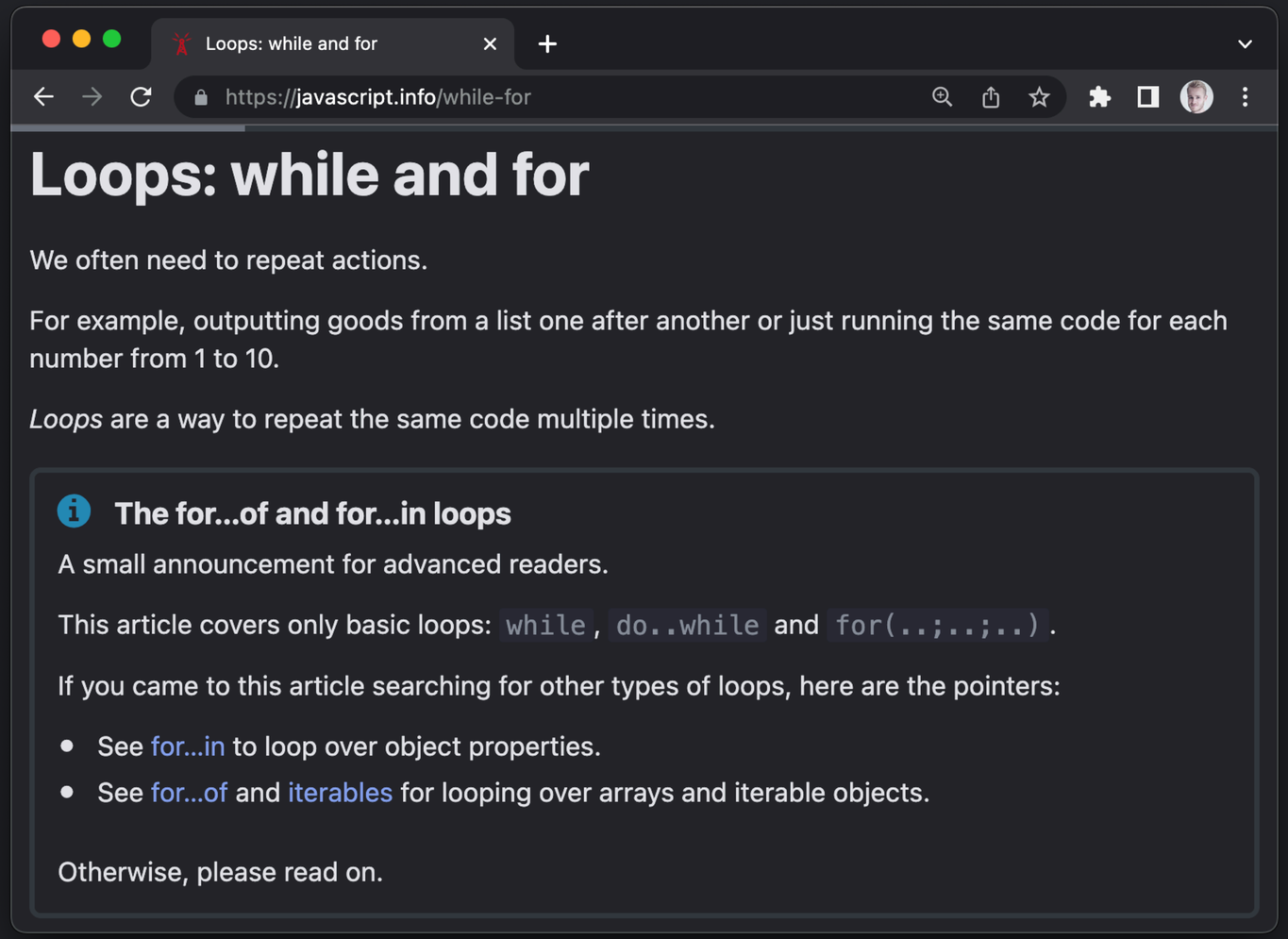
Arrays

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



https://javascript.info/while-for

For of loop

iterate over arrays or other iterable objects

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

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

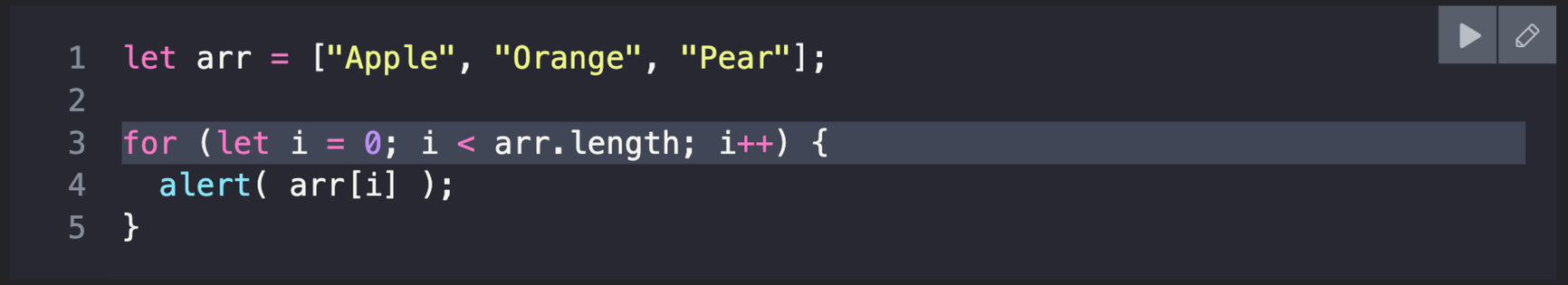
Loops

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

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

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

JavaScript.info/array#loops

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



```
let arr = ["Apple", "Orange", "Pear"];

for (let i = 0; i < arr.length; i++) {
  alert( arr[i] );
}
```

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

```
let fruits = ["Apple", "Orange", "Plum"];

// iterates over array elements
for (let fruit of fruits) {
  alert( fruit );
}
```

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
```

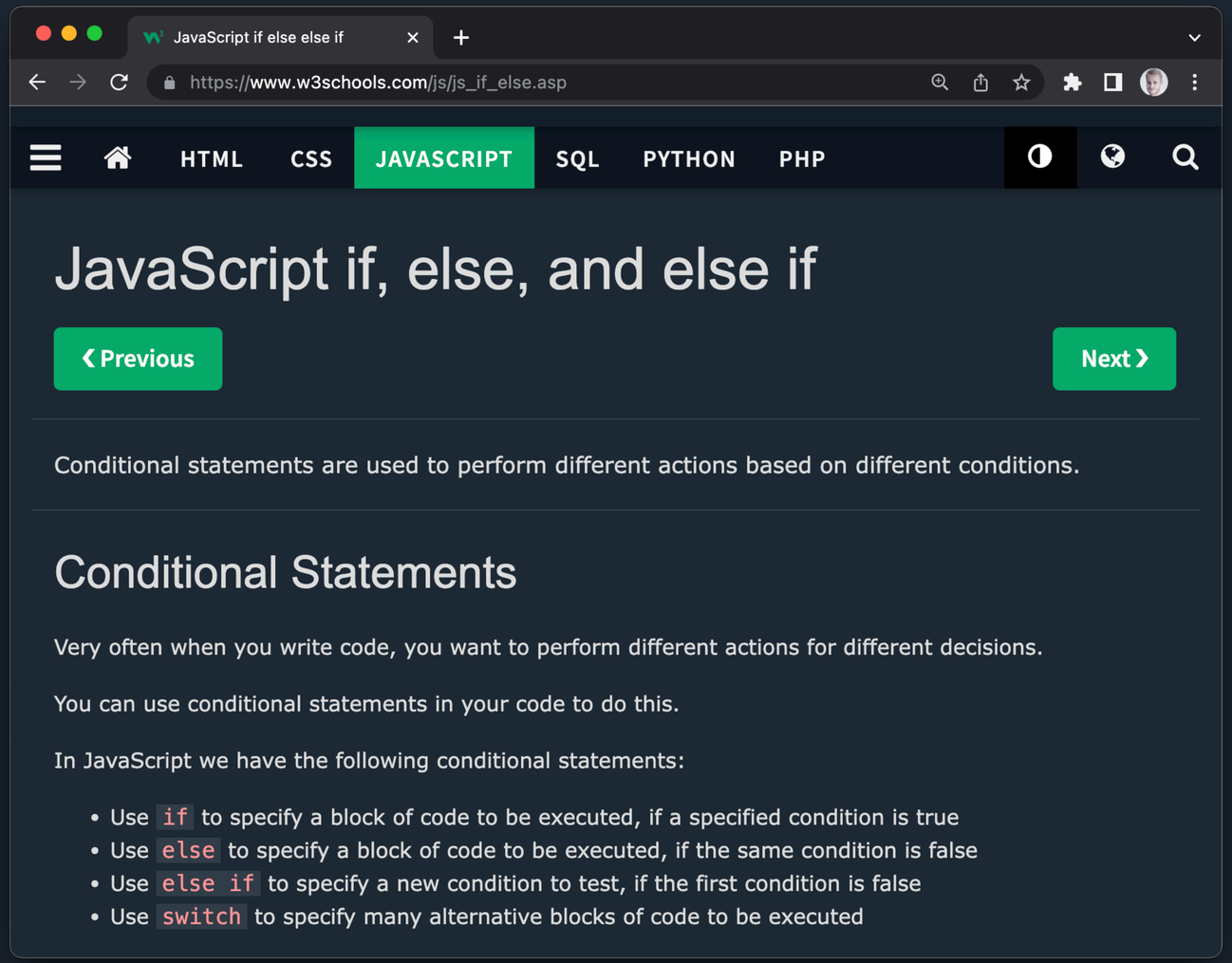
LOOPS

... LOOP THROUGH AN ARRAY AND ADD A CONDITION

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

```
▶ {name: "Rasmus Cederdorff", mail: "race@baaa.dk"}        main.js:21
```

Conditional Statements



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(…)`

Array methods

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

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:

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

For instance:

```
let users = [
  {id: 1, name: "John"},
  {id: 2, name: "Pete"},
  {id: 3, name: "Mary"}
];

// returns array of the first two users
let someUsers = users.filter(item => item.id < 3);

alert(someUsers.length); // 2
```

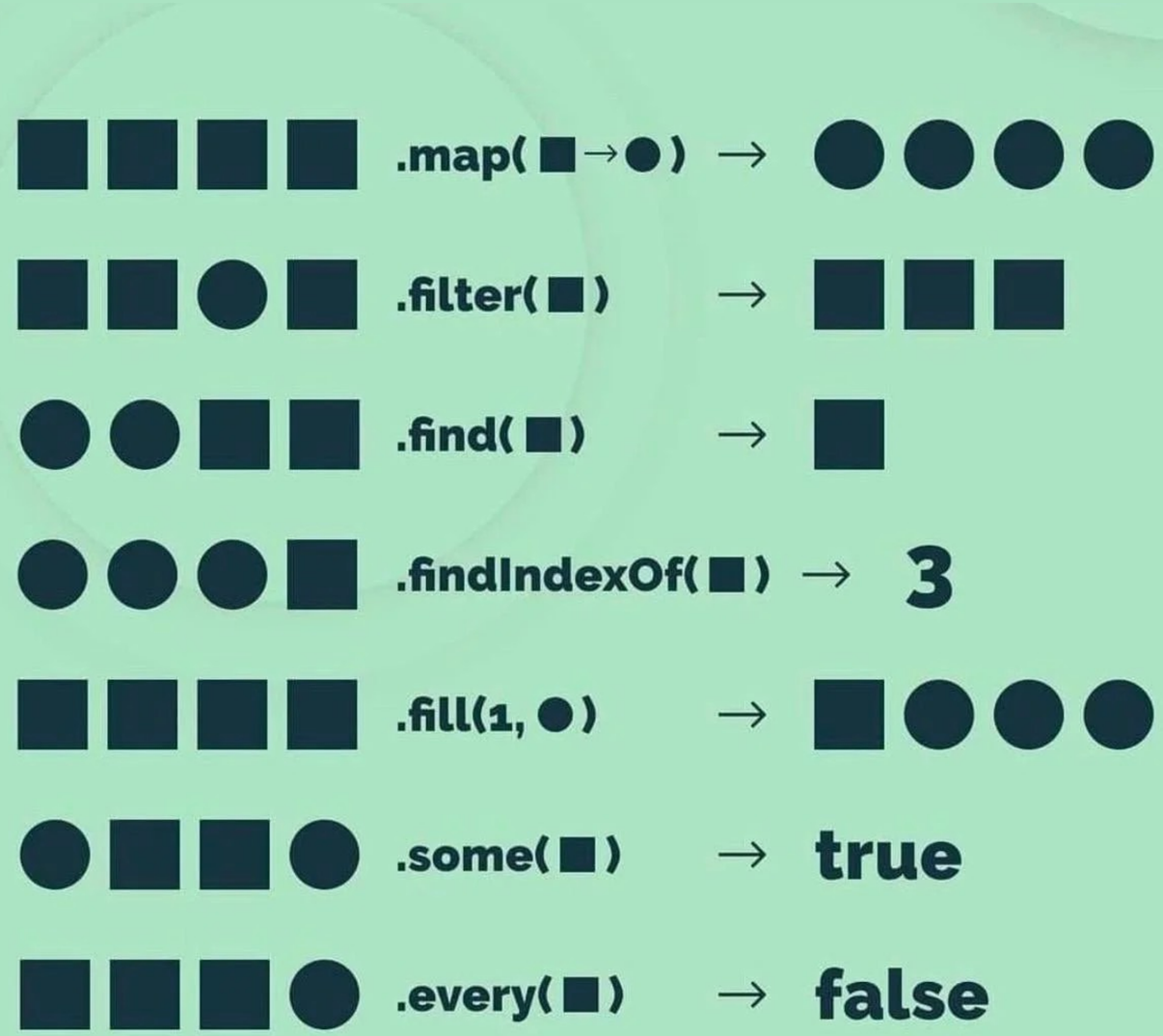
Transform an array

Let's move on to methods that transform and reorder an array.

map

The arr.map method is one of the most useful and often used.

It calls the function for each element of the array and returns the array of results.



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"
```

https://www.youtube.com/watch?v=DC471a9qrU4

Array.map(...)

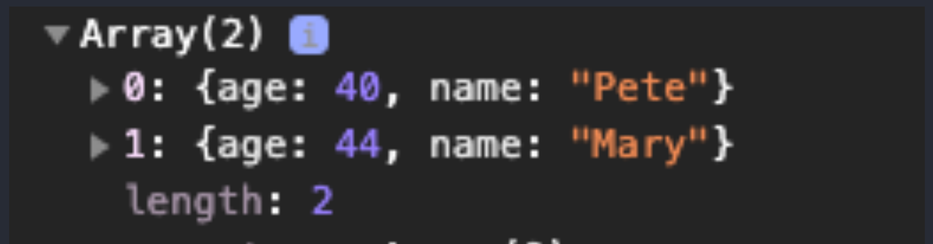
- Project template: `array-map`
- Use `.map` to map over the `persons` array and concatenate `firstName` and `lastName` to a new property called `name`.
- `console.log()` the result
- Add a property called `birthYear` for each `person` object. Use `map` to map over the array and add the property `birthYear` 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);
```



`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`

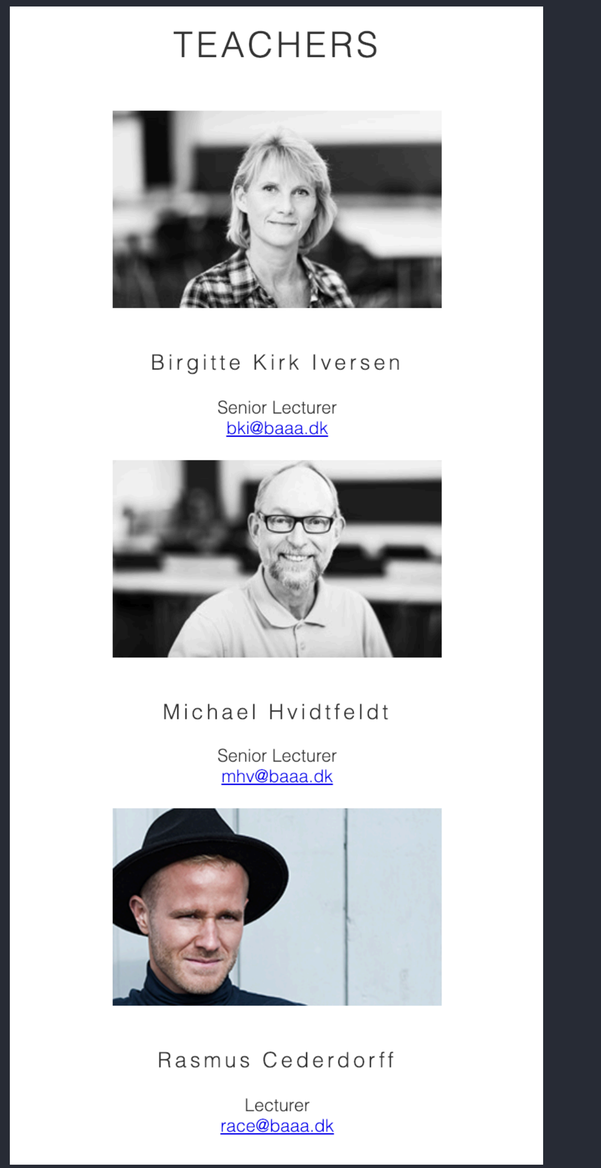
```
let name = "Alicia";
let age = 6;

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>";
  }
}
```



`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

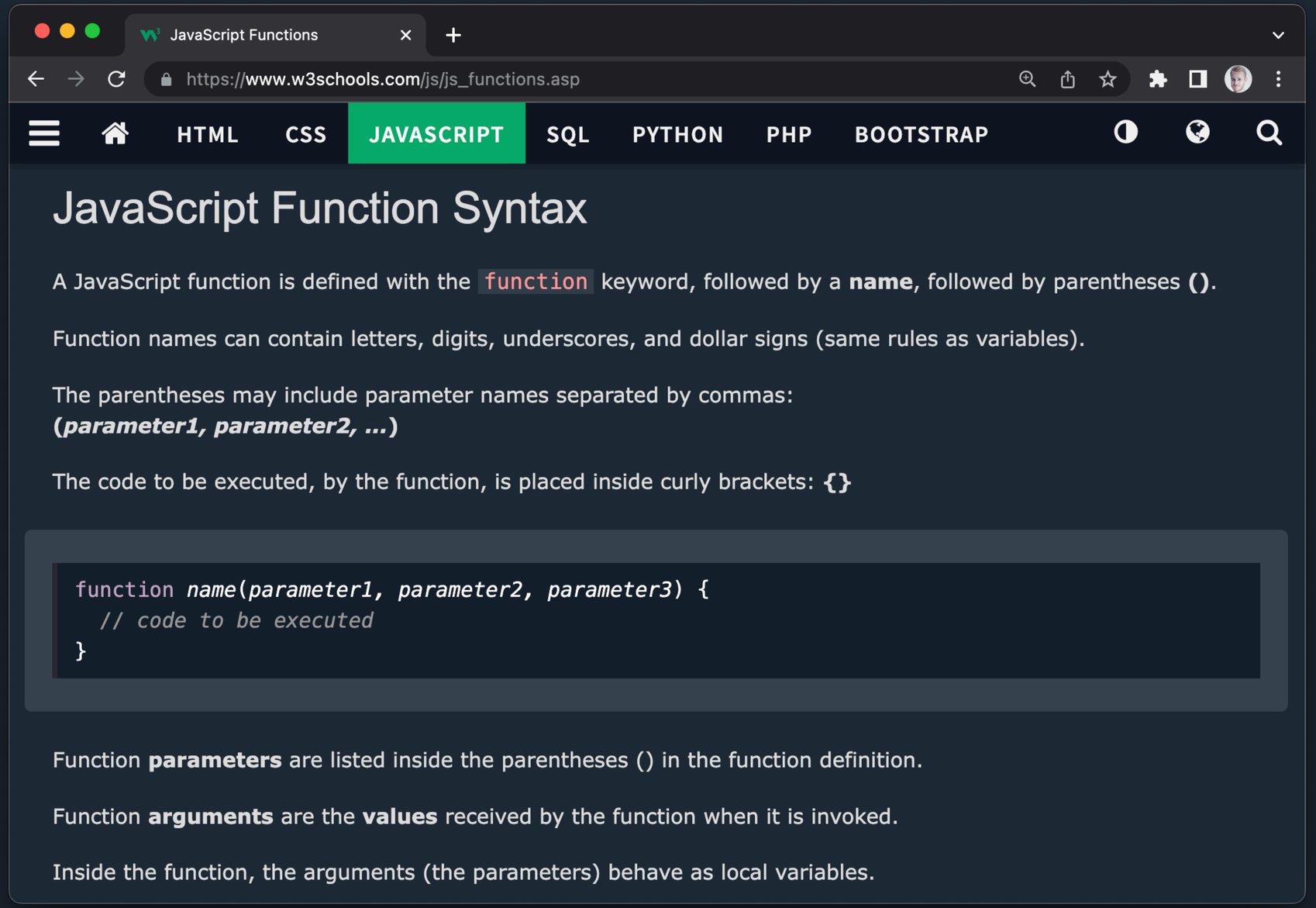
Functions

Function declaration

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

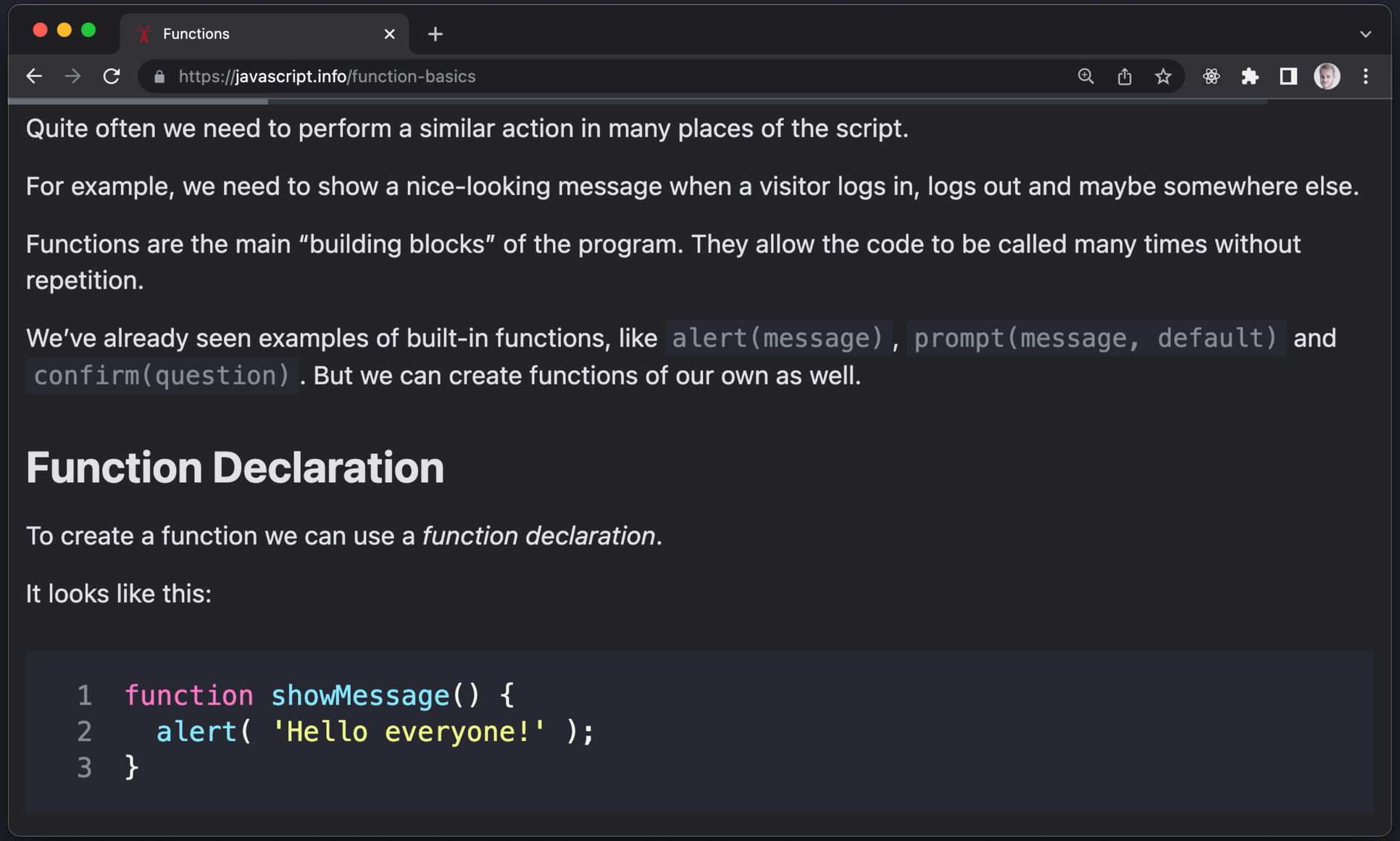
```
function log(message) {
  console.log(message);
}

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

https://javascript.info/function-basics



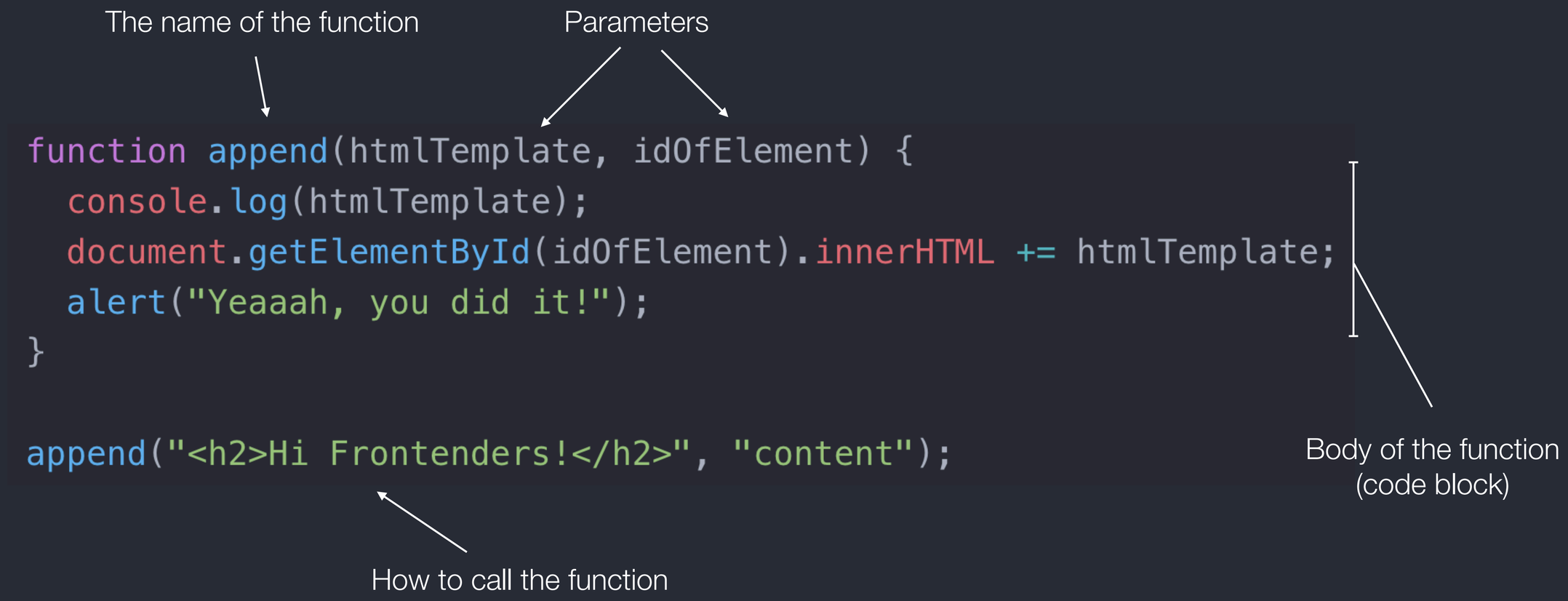
https://www.w3schools.com/js/js_functions.asp

JavaScript.info/Function-Basics



Functions

Function declaration



https://javascript.info/function-basics

Functions

Arrays & Loops

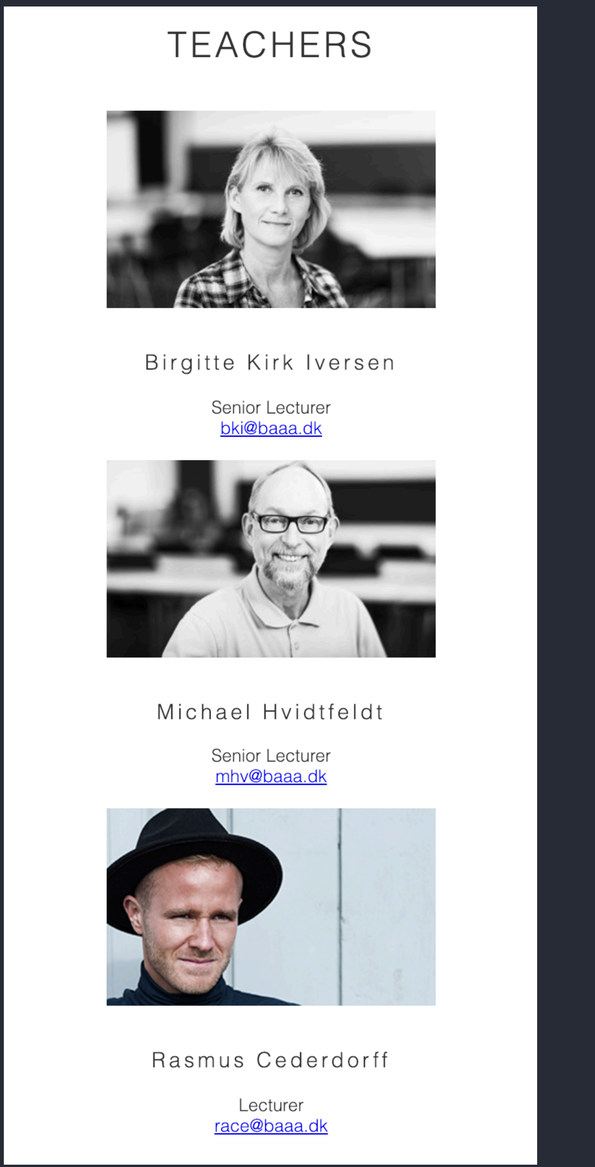
The name of the function

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);
```

Body of the function (code block)

How to call the function



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:

```
function showMessage() {
  let message = "Hello, I'm JavaScript!"; // local variable

  alert( message );
}

showMessage(); // Hello, I'm JavaScript!

alert( message ); // <-- Error! The variable is local to the function
```

Outer variables

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

```
let userName = 'John';

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

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
```

https://javascript.info/function-basics#outer-variables

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.

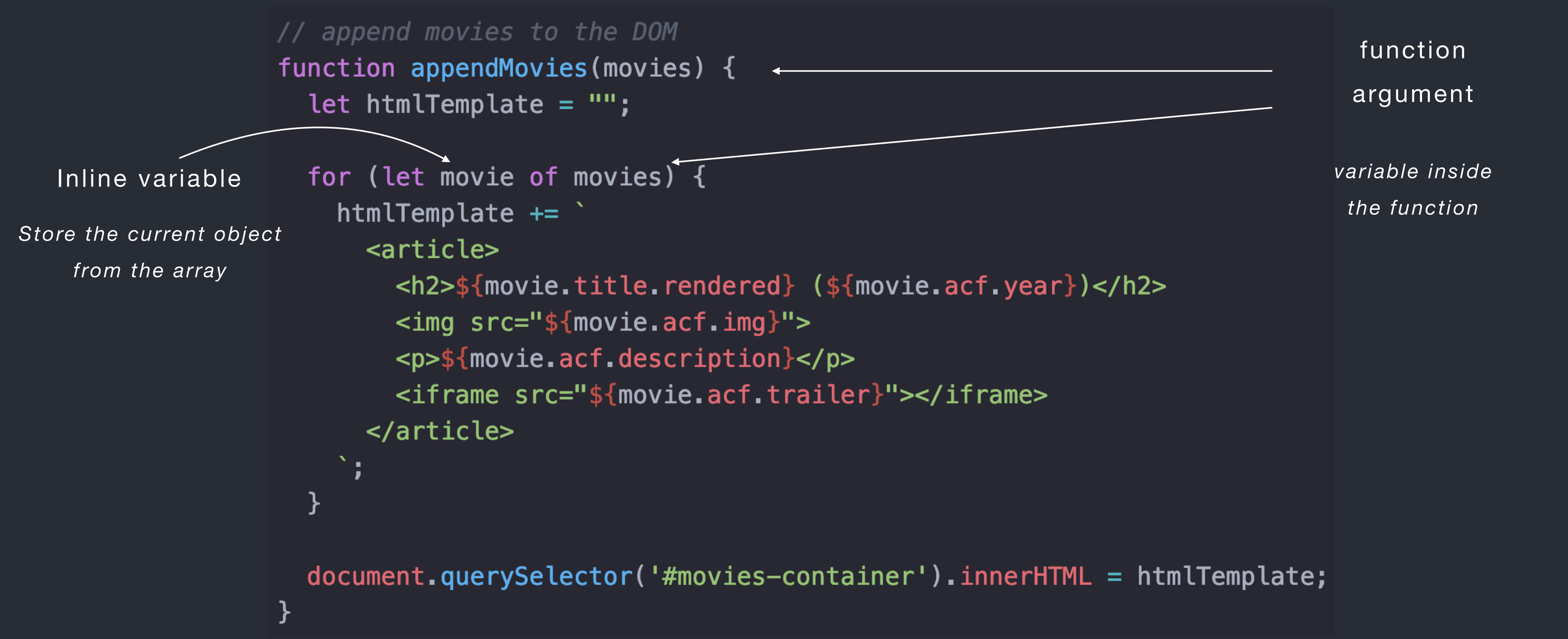
https://javascript.info/function-basics#outer-variables

Global Variable

ARRAY

movies

```
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();
```


// 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>
<img src="${movie.acf.img}">
${movie.acf.description}
<iframe src="${movie.acf.trailer}"></iframe>
</article>
`;
}
document.querySelector('#movies-container').innerHTML = htmlTemplate;
}
function argument
variable inside the function
Inline variable
Store the current object from the array

```
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();

// 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>
        <img src="${movie.acf.img}">
```

function argument

variable inside the function

Inline variable

Store the current object from the array

```
// ========== 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

Scopes:

- Local Variable
- Global Variable

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
```

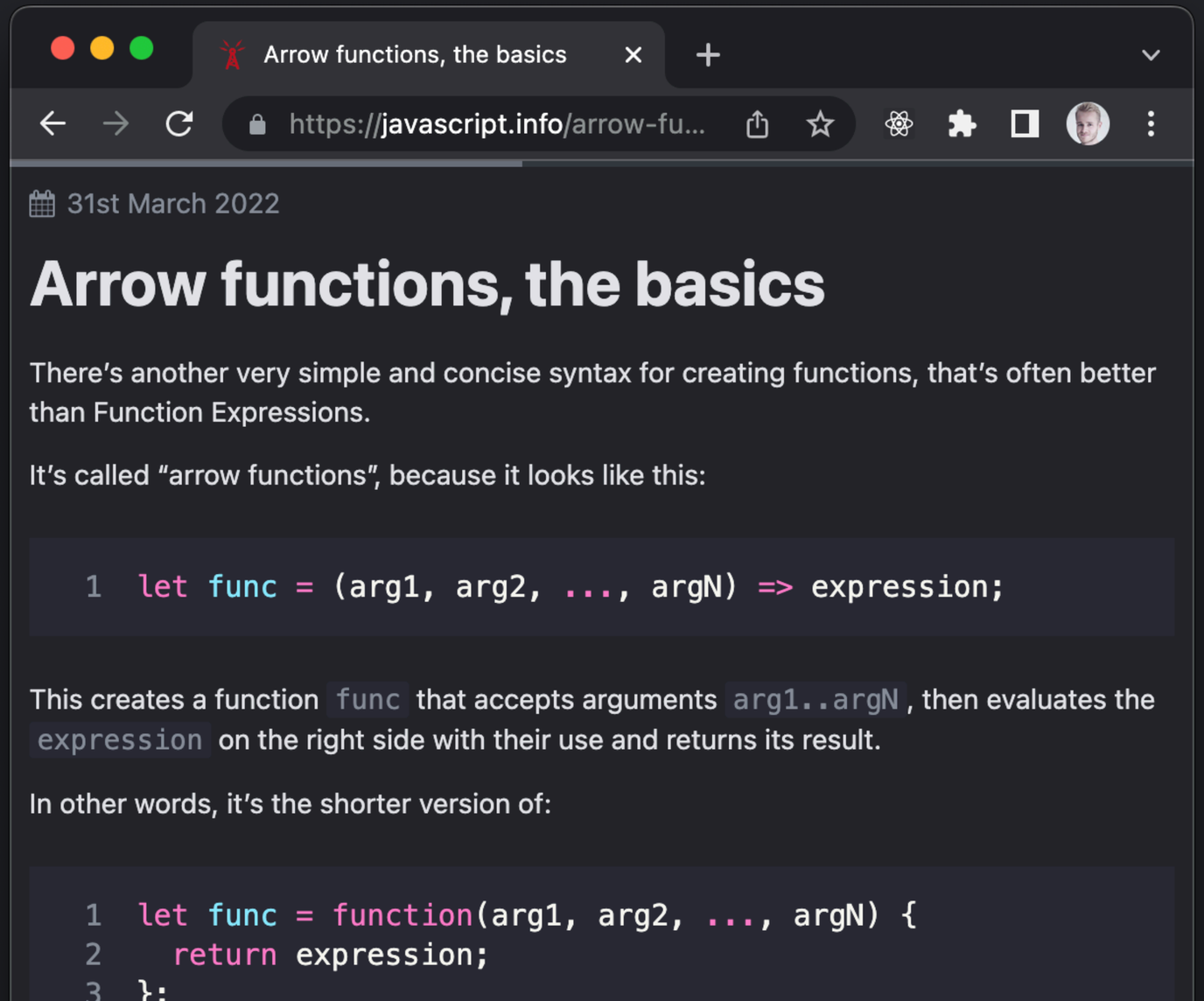
Function called by another function

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>
                <img src="${user.avatar_url}">
                <h2>${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



```
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"
```

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

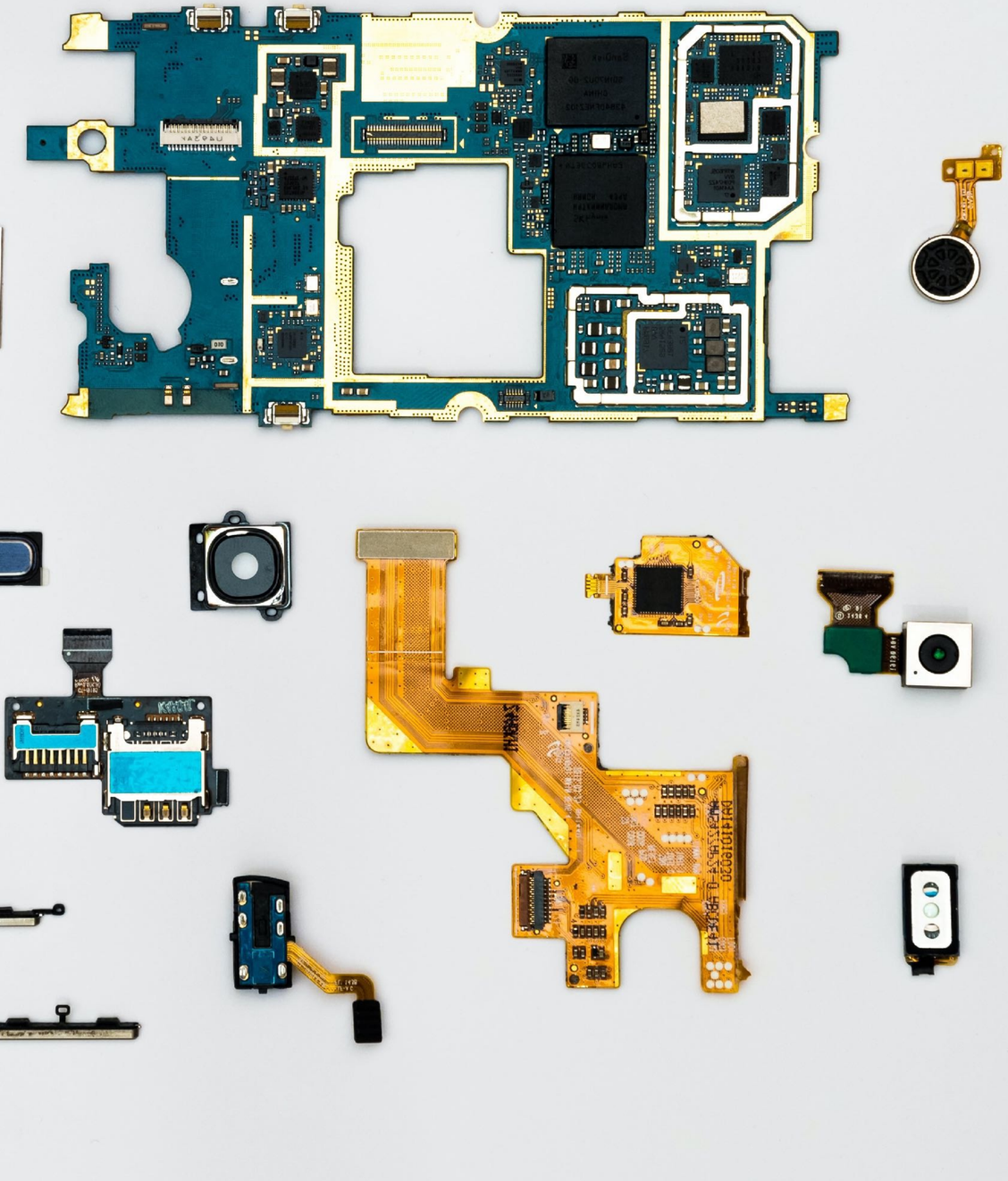
What: A module is just a script file. One is script or one file is one module. We use the keywords `export` and `import` to tell what we would like to export from one script and import in another script.

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}!`);
}
```

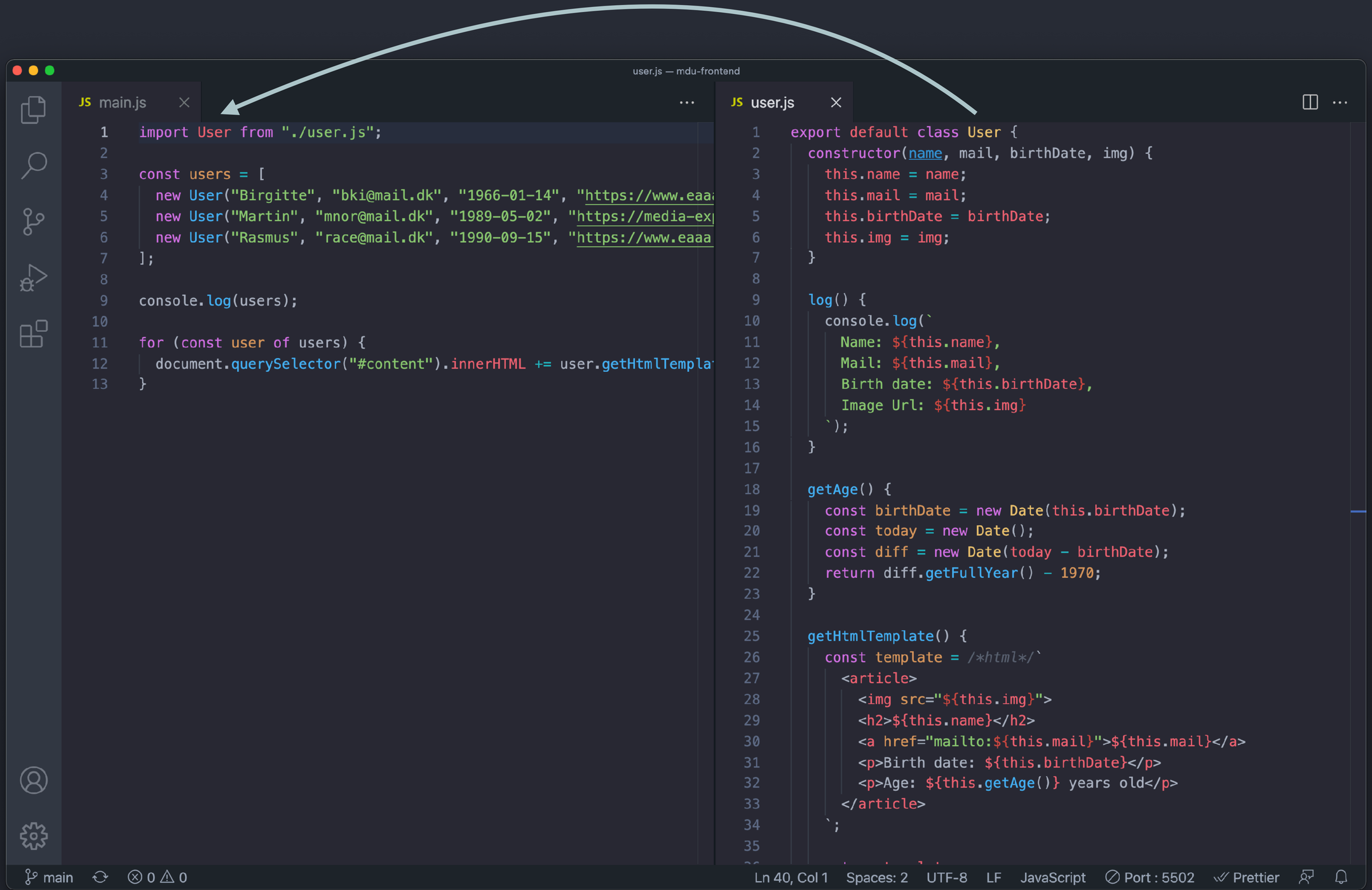
A MODULE (SCRIPT)

```
// 📁 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');
```


user.js — mdu-frontend
main.js
import User from "./user.js";
const users = [
new User("Birgitte", "bki@mail.dk", "1966-01-14", "https://www.eaa
new User("Martin", "mnor@mail.dk", "1989-05-02", "https://media-ex
new User("Rasmus", "race@mail.dk", "1990-09-15", "https://www.eaaa
];
console.log(users);
for (const user of users) {
document.querySelector("#content").innerHTML += user.getHtmlTempla
}
user.js
export default class User {
constructor(name, mail, birthDate, img) {
this.name = name;
this.mail = mail;
this.birthDate = birthDate;
this.img = img;
}
log() {
console.log(`
Name: ${this.name},
Mail: ${this.mail},
Birth date: ${this.birthDate},
Image Url: ${this.img}
`);
}
getAge() {
const birthDate = new Date(this.birthDate);
const today = new Date();
const diff = new Date(today - birthDate);
return diff.getFullYear() - 1970;
}
getHtmlTemplate() {
const template = /*html*/`
<article>
<img src="${this.img}">
<h2>${this.name}</h2>
${this.mail}
Birth date: ${this.birthDate}
Age: ${this.getAge()} years old
</article>
`;
main
0 0
Ln 40, Col 1
Spaces: 2
UTF-8
LF
JavaScript
Port : 5502
Prettier

TELL THE BROWSER IT'S A MODULE

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


Code
Every
Day