JavaScript Concepts

Web Development

ERHVERVSAKADEMI AARHUS
BUSINESS ACADEMY AARHUS

Content

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

JS
TIPS
Aa

100 SECONDS OF

JS

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

index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <h1>This is a Heading</h1>
    <p>This is a paragraph.</p>
    <button onclick="tryMe()">Try me</button>
    <script src="app.js"></script>
  </body>
</html>
```

app.js

```javascript
function tryMe() {
  document.body.style.backgroundColor = "red";
  document.body.style.color = "white";
}
```

What is JavaScript?

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

... is the programming language of the Web.

... is easy to learn.

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

... can change styling of HTML.

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

index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <h1>This is a Heading</h1>
    <p>This is a paragraph.</p>
    <button onclick="tryMe()">Try me</button>
    <script src="app.js"></script>
  </body>
</html>
```

app.js

```javascript
function tryMe() {
  document.body.style.backgroundColor = "red";
  document.body.style.color = "white";
}
```

With JavaScript we are able to

... build dynamic web pages and web apps.

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

... do DOM-manipulation.

... build and develop anything 🤷🏼‍♂️

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

DOM Manipulation

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

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

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

```html
<body>
  <header>
    <h1>PROJECT TEMPLATE</h1>
  </header>
  <section id="content"></section>

  <!-- main js file -->
  <script src="js/main.js"></script>
</body>
```

JavaScript HTML DOM

index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My title</title>
  </head>

  <body>
    <h1>My header</h1>
    <a href="https://cederdorff.com">My link</a>
  </body>
</html>
```

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

The HTML DOM (Document Object Model)

A model as a tree of Objects

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

https://www.w3schools.com/whatis/whatis_htmldom.asp

index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My title</title>
  </head>

  <body>
    <h1>My header</h1>
    <a href="https://cederdorff.com">My link</a>
  </body>
</html>
```

JavaScript HTML DOM

Document Object Model

The HTML document as an object

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

JavaScript can:

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

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

DOM

Elements

```html
<html>
  <head></head>
  <body>
    <div id="app">
      <h1>Develop. Preview.
Ship. 🚀</h1>
    </div>
    <script type="text/
javascript">...</script>
  </body>
</html>
```

SOURCE CODE (HTML)

index.html

```html
<html>
  <head></head>
  <body>
    <div id="app"></div>
    <script type="text/
javascript">...</script>
  </body>
</html>
```

https://nextjs.org/learn/foundations/from-javascript-to-react/updating-ui-with-javascript

Searching the DOM: `getElement*` & `querySelector*`

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

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

https://javascript.info/searching-elements-dom

Searching the DOM: `getElementsBy*`

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

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

https://javascript.info/searching-elements-dom

Searching the DOM: `querySelectorAll`

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

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

https://javascript.info/searching-elements-dom

JS HTML DOM

`getElement*` or `querySelector*`?

ES6+

Modern JavaScript

Modern JavaScript

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

read
the
documentation
read the documentation
read
The
documentation
Ask Rasmus

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

Variables

https://javascript.info/variables

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”:

```
let message;
```

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

```
let message;

message = 'Hello'; // store the string 'Hello' in the variable named message
```

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

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

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

```html
<body>
  <header>
    <h1>PROJECT TEMPLATE</h1>
  </header>
  <section id="content"></section>

  <!-- main js file -->
  <script src="js/main.js"></script>
</body>
```

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

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

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

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

var **vs** let

THE DIFFERENCE IS THE SCOPING

VAR IS FUNCTION-WIDE OR GLOBAL SCOPE

LET IS BLOCK SCOPED

VAR TOLERATES REDECLARATION

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

```javascript
// 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

A set of named values

property

value

```
let person = {
    name: "Birgitte Kirk Iversen",
    position: "Senior Lecturer",
    department: "Programmes within Digital Communication and Multimedia",
    address: "Sønderhøj 30, 8260 Viby J",
    mail: "bki@baaa.dk",
    phone: "+45 7228 6316",
    image: "https://www.baaa.dk/media/u4gorzsd/birgitte-kirk-iversen2.jpg"
};
```

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

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
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
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"
}
```

Define yourself as an object

with the following properties

`name, age, mail, phone, city, address`

Arrays

Collections

Ordered collection of values or objects

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

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

First element

Second element

Arrays

Rasmus Cederdorff

Position:
Lecturer

Department/p
Multimedia De
Digital Concep

Address:
Ringvej Syd 10

Mail:
race@baaa.dk

Phone:
7228 6318

Michael Hvidtfeldt

Position:
Lecturer

Department/p
Multimedia De

Address:
Ringvej Syd 10

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);
```

Teachers

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

Teachers

Birgitte Kirk Iversen

Senior Lecturer
bki@baaa.dk

Maria Louise Bendixen

Senior Lecturer
mlbe@baaa.dk

Kim Elkjær Marcher-Jepsen

Lecturer
kije@baaa.dk

Rasmus Cederdorff

Lecturer
race@baaa.dk

Console

top ▼ Filter

Default levels ▼ 1 Issue: 1

main.js:46

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

array-teachers

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

cederdorff
cederdorff
Edit Profile
Rasmus Cederdorff
Senior Lecturer & Web App Developer
Alicia & Ida
Aarhus, Denmark
cederdorff.com
AMSTER...
LONDON ...
Karla & Pi...
@villacph
IDA
181
posts
5,441
followers
635
following

Dimensions: iPhone ... 375 × 667 100%

Elements | Components | Console | Performance insights | Recorder | Sources | Network | 1

Preserve log | Disable cache | No throttling

Invert | Hide data URLs | All | Fetch/XHR | JS | CSS | Img | Media | Font | Doc | WS | Wasm | Manifest | Other

Has blocked cookies | Blocked Requests | 3rd-party requests

2000 ms | 4000 ms | 6000 ms | 8000 ms | 10000 ms | 12000 ms | 14000 ms

Name

- ?modules=PolarisBD...
- ?content_type=PROF...
- ?username=keacph
- reels_tray/
- timeline/
- ig_sso_users/
- logging_client_events
- bz?__d=dis
- falco
- bz?__a=1&__ccg=G...
- batch_fetch_web/
- get_encrypted_crede...
- highlights_tray/
- ?target_user_id=140...
- badge/
- story/
- bulk-route-definitions/
- bulk-route-definitions/
- ?query_hash=69cba4...
- bz?__a=1&__ccg=G...
- bz?__a=1&__ccg=G...
- bulk-route-definitions/
- bz?__a=1&__ccg=G...
- logging_client_events
- bz?__d=dis

26 / 96 requests | 257 kB

Headers | Payload | Preview | Response | Initiator | Timing | Cookies


```
{data: {user: {,…}}, status: "ok"}
  data: {user: {,…}}
    user: {,…}
      bio_links: []
      biography: "A visual sneak peek into life at KEA Copenhagen School of Design and Technology. \n#keacph
      biography_with_entities: {,…}
      blocked_by_viewer: false
      business_address_json: null
      business_category_name: "Home Services"
      business_contact_method: "CALL"
      business_email: null
      business_phone_number: null
      category_enum: "LOCAL"
      category_name: null
      connected_fb_page: null
      country_block: false
      edge_felix_video_timeline: {count: 116, page_info: {has_next_page: true,…},…}
      edge_follow: {count: 618}
      edge_followed_by: {count: 5508}
      edge_media_collections: {count: 0, page_info: {has_next_page: false, end_cursor: null}, edges: []}
      edge_mutual_followed_by: {count: 7, edges: [{node: {username: "retrovilla"}}, {node: {username: "finde
      edge_owner_to_timeline_media: {count: 1111, page_info: {has_next_page: true,…},…}
      edge_saved_media: {count: 0, page_info: {has_next_page: false, end_cursor: null}, edges: []}
      external_url: "https://linktr.ee/keacph"
      external_url_linkshimmed: "https://l.instagram.com/?u=https%3A%2F%2Flinktr.ee%2Fkeacph&e=ATNF7APjyvCLD
      fbid: "17841400899781985"
      followed_by_viewer: true
      follows_viewer: true
      full_name: "Københavns Erhvervsakademi"
      guardian_id: null
      has_ar_effects: false
      has_blocked_viewer: false
      has_channel: false
      has_clips: false
```

iPhone 6/7/8 Plus ▼ 414 × 736 90% ▼

Elements | Console | Sources | Network | Performance | Memory » | 1 2 | 1

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 | 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

16 / 97 requests | 17


```
{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
```

Course roster: WU-E22a - 1. se

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

Account
Admin
Dashboard
Courses
Calendar
Inbox
History
Commons
Student links

WU-E22a › People

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. semester	Student	24 Aug at 13:16	01:04:21
	Martin Rieper Boesen	eaamrbo@students.eaaa.dk		WU-E22a - 1. semester	Student	24 Aug at 7:54	01:07:06
	Dan Okkels Brendstrup	dob@eaaa.dk		WU-E22a - 1. semester	Teacher	3 Aug at 8:55	
	Rasmus Cederdorff	race@eaaa.dk		WU-E22a - 1. semester	Teacher	25 Aug at 9:28	01:19:23
	Jeffrey David Serio	jds@eaaa.dk		WU-E22a - 1. semester	Teacher	17 Aug at 16:39	
	Chalotte Meng Emanuel Dyrholm	eaacmed@students.eaaa.dk		WU-E22a - 1. semester	Student	23 Aug at 16:59	22:24

Elements | Components | Network | 22

Preserve log | Disable cache | No throttling

Invert | Hide data URLs

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

Has blocked cookies | Blocked Requests | 3rd-party requests

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

Name | Headers | Payload | Preview | Response | Initiator

property

value

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

6 / 157 requests


```
[{id: "23974", name: "Clara Juul Birk", created_at: "2020-08-
  0: {id: "23974", name: "Clara Juul Birk", created_at: "2020
  1: {id: "36267", name: "Martin Rieper Boesen", created_at:
  2: {id: "29923", name: "Dan Okkels Brendstrup", created_at:
    avatar_url: "https://eaaa.instructure.com/images/thumbnai
    created_at: "2021-07-30T00:46:05+02:00"
    custom_links: []
    email: "dob@eaaa.dk"
    enrollments: [{id: "391064", user_id: "29923", course_id:
    id: "29923"
    integration_id: null
    login_id: "dob@eaaa.dk"
    name: "Dan Okkels Brendstrup"
    short_name: "Dan Okkels Brendstrup (adjunkt – dob@eaaa.dk
    sis_user_id: null
    sortable_name: "Brendstrup, Dan Okkels"
  3: {id: "14427", name: "Rasmus Cederdorff", created_at: "20
  4: {id: "41", name: "Jeffrey David Serio", created_at: "201
  5: {id: "24043", name: "Chalotte Meng Emanuel Dyrholm", cre
  6: {id: "23978", name: "Jeppe Frik", created_at: "2020-08-0
  7: {id: "23963", name: "Daniel Tjerrild Gamborg", created_a
  8: {id: "23992", name: "Casper Hedegaard Hansen", created_a
  9: {id: "36266", name: "Morten Gedsted Hansen", created_at:
  10: {id: "23980", name: "Anders Husted", created_at: "2020-
  11: {id: "23531", name: "Søren Bo Jørgensen", created_at: "
```


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

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

Loops: while and for

https://javascript.info/while-for

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.

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.

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

Ternary Operator

1.5. Ternary Operator

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

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

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

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

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

const age = 43;

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

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

The use of

; : () [] {} $ ” ` ` .

== === < > ?

== VS ===

=== IS MORE STRICT

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

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

Array Methods to know

`.push(…)`

`.map(…)`

`.filter(…)`

`.find(…)`

`.sort(…)`

Computer science student

Senior developer, 10+ years experience

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

Array methods

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

Chapter

Data types

Lesson navigation

Add/remove items

Iterate: forEach

Searching in array

Transform an array

Array.isArray

Most methods support "thisArg"

Summary

Tasks (13)

Comments

Share

Edit on GitHub

Ads

filter

The `find` method looks for a single (first) element that makes the function return `true`.

If there may be many, we can use arr.filter(fn).

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

```
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);
```

```
▼ Array(2) i
  ▶ 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>";
  }
}
```

`template string`

...EMBED VARIABLES AND EXPRESSIONS IN A STRING

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

→

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

`VS Code ES6 String HTML`

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

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

→

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

Add language support in template string

Functions

A block of code to perform a specific task.

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

Best practice: write reusable code

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

log("Hi Frontenders!");
```

https://javascript.info/function-basics

Functions

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

JavaScript Functions

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

HTML CSS JAVASCRIPT SQL PYTHON PHP BOOTSTRAP

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.

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

JavaScript.info/Function-Basics

Functions

https://javascript.info/function-basics

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:

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

Functions

Function declaration

The name of the function

Parameters

```
function append(htmlTemplate, idOfElement) {
  console.log(htmlTemplate);
  document.getElementById(idOfElement).innerHTML += htmlTemplate;
  alert("Yeaaah, you did it!");
}

append("<h2>Hi Frontenders!</h2>", "content");
```

Body of the function
(code block)

How to call the function

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`.

```
function showMessage(from, text) { // parameters: from, text
  alert(from + ': ' + text);
}

showMessage('Ann', 'Hello!'); // Ann: Hello! (*)
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

```
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

movie**s**

```
// 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="${movie.acf.img}">
        <p>${movie.acf.description}</p>
        <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:

```
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
```

Function called by another function

```
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();
```

Use the name of the function to call / execute

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

Javascript.info/Arrow-Functions-Basics

Arrow functions, the basics

https://javascript.info/arrow-fu...

31st March 2022

Arrow functions, the basics

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:

```
let func = (arg1, arg2, ..., argN) => expression;
```

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:

```
let func = function(arg1, arg2, ..., argN) {
  return expression;
};
```

```
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.

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


// 📂 main.js
import {sayHi} from './sayHi.js';

alert(sayHi); // function...
sayHi('John'); // Hello, John!
```

A MODULE (SCRIPT)

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.

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


// 📂 main.js
import {sayHi} from './sayHi.js';

alert(sayHi); // function...
sayHi('John'); // Hello, John!
```

A MODULE (SCRIPT)

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

```javascript
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

```javascript
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>
        <a href="mailto:${this.mail}">${this.mail}</a>
        <p>Birth date: ${this.birthDate}</p>
        <p>Age: ${this.getAge()} years old</p>
      </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>
```

Exercises

Web Development

ERHVERVSAKADEMI AARHUS
BUSINESS ACADEMY AARHUS

OBJECT TEACHERS EXERCISE

OBJECT TEACHERS

Create a grid view with your new teachers.

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

ARRAY TEACHERS

- Make a copy of your local project `object-teachers` and rename to `array-teachers`

- Modify the structure of your script (your JavaScript), in order to use an array and a loop to display your teachers:

  - Define an array and add the object to the array.

  - Test and debug the array using `console.log`

  - Use a for of loop to loop through the array and display the teachers (append to the DOM).

  - Add another teacher object and test.

FAMILY MEMBERS

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

FAMILY MEMBERS

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

FAMILY MEMBERS 2

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

ARRAY TEACHERS 2

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

EXERCISE

TEACHER AND/OR FAMILY MEMBER

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

EXERCISE

TEACHER AND/OR FAMILY MEMBER

- Improvements:
  - Add search functionality: Add an input field to your `index.html` and implement a search function in your script.
  - Add edit functionality
  - Add delete functionality

- JSON & Fetch (+++):
  - Create a JSON file consisting of an array of `teachers` or `familyMembers`.
  - Make use of `fetch(…)` to fetch the data from the JSON file and append it to the DOM.
- JSON: https://www.w3schools.com/js/js_json_intro.asp
- FETCH: https://javascript.info/fetch or https://scrimba.com/scrim/cVye2ztW

JSON & FETCH
FAMILY MEMBERS

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

spa-products-fetch-json-enhanced

PRODUCT WEB APP

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

PRODUCT WEB APP (EXTRA)

- Project template: `spa-products-template`
- Use your knowledge about JavaScript and SPA to build a web app.
- Declare a new array with at least 4 products. The product objects must have the following properties: `model`, `brand`, `price`, `img` and `status`
- Two of your products must have the status `outOfStock` and two `inStock`.
- The img should just be an URL to an image like: https://store.storeimages.cdn-apple.com/4668/as-images.apple.com/is/mbp13touch-space-select-201807?wid=904&hei=840&fmt=jpeg&qlt=80&op_usm=0.5,0.5&.v=1529520060550
- Display all products by appending the products to the PRODUCTS page.
- Another tab/page with a form to add new products.
- Customise and style your app.
- Add functionality to filter and/or sort by the `status`. You could add an input checkbox to show and hide `outOfStock` products.
- Add functionality to handle edit and delete.

JSON & FETCH PRODUCTS

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

Code
Every
Day