Introduction to JavaScript

3rd semester @ Erhvervsakademi København

GitHub

What can we do with JavaScript?

- Build dynamic and interactive web apps
- Manipulate web page content (DOM)
- Fetch data from REST APIs
- Handle user events (clicks, form submissions)
- Create single-page applications (SPAs)
- Build server-side applications with Node.js
- Develop mobile apps with frameworks like React Native

What problems does JavaScript solve?

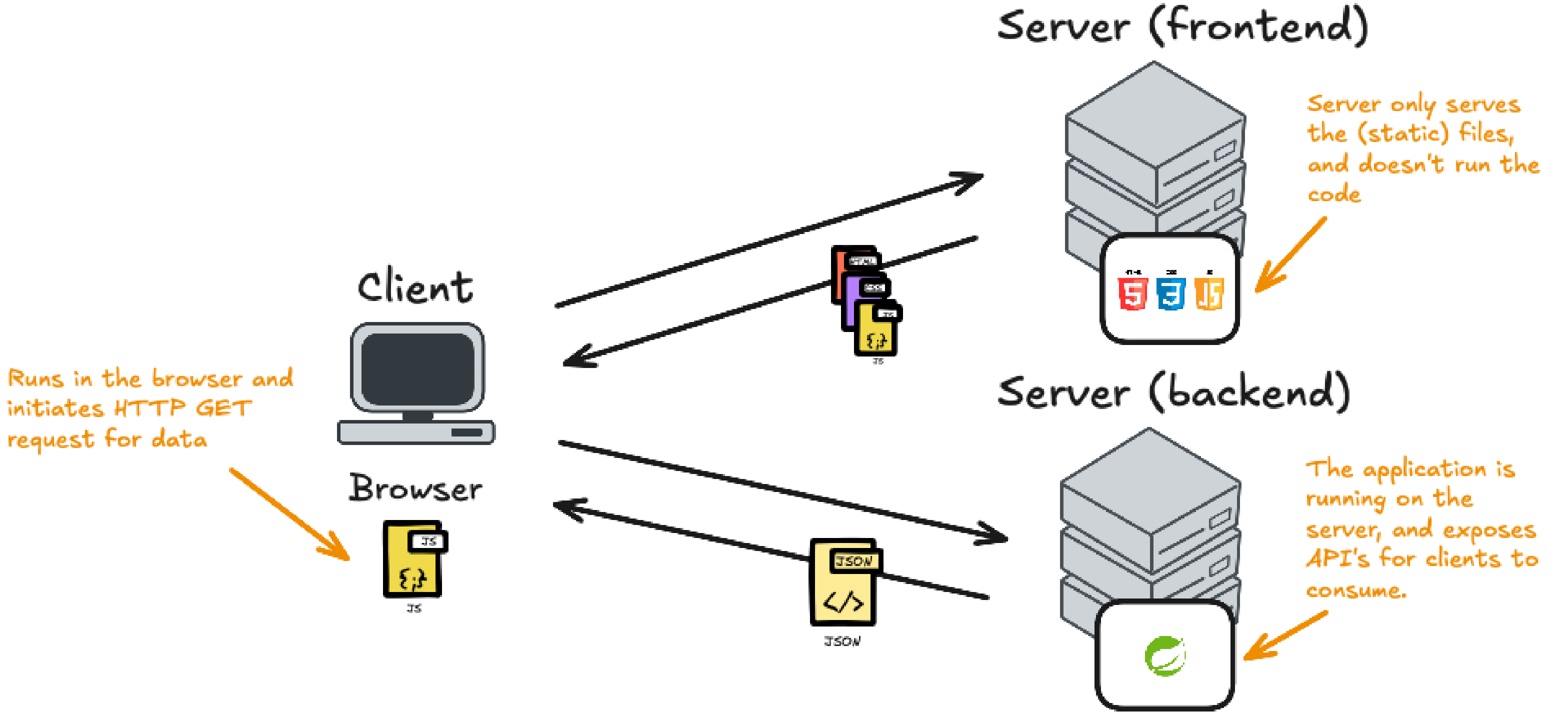
- **Making web pages interactive**: Responding to user actions (clicks, form submissions, etc.) without needing to reload the page.
- **Manipulating the DOM**: Dynamically changing the content and structure of web pages.
- **Asynchronous programming**: Fetching data from REST APIs without blocking the user interface.
- **Building web applications**: Using frameworks like React, Angular, or Vue.js to create complex, single-page applications.
- **Server-side development**: Using Node.js to build backend services and APIs.

Our scope for JavaScript

> Up until now, we have been building REST APIs using Spring Boot.

In this part of the course, we will focus on using JavaScript to build interactive web applications that consume our REST APIs.

Client-server model



Server only serves the (static) files, and doesn't run the code

Runs in the browser and initiates HTTP GET request for data

The application is running on the server, and exposes API's for clients to consume.

The frontend is running in the browser. JavaScript fetches data from the backend by consuming a REST API.

The Frontend and backend are decoupled, and can live on different servers (it does not mean that they have to).

What is JavaScript?

Interpreted language:

- No need to compile code before running it.
- Works directly in **web browsers**.

Dynamic typing:

- No declared type, but values have types.
- Types can change at runtime.

Object-oriented:

- *Most things behave like objects*
- Later classes were added to the language (syntactic sugar).

JavaScript in the browser

> How do we run JavaScript code in a web page?

Inline JavaScript inside an HTML file:

```
<script>
    // JavaScript code goes here
</script>
```

External JavaScript file:

```
<script src="app.js"></script>
```

JavaScript syntax

C/C++/Java-like syntax:

- Curly braces `{}` to define code blocks.
- Semicolons `;` to end statements (optional but recommended).
- Parentheses `()` for function calls and control flow.
- Comments: `//` for single-line, `/* ... */` for multi-line.
- Case-sensitive (e.g., `myVar` and `myvar` are different).
- Usual operators: `+`, `-`, `*`, `/`, `%`, `=`, `==`, `===`, `!=`, `!==`, `<`, `>`, `<=`, `>=`.
    - Although `==` and `!=` are weird!

Declaring variables in JS vs Java

> No type declarations in JavaScript!

Java:

```java
int x = 5;
String name = "Alice";
```

JavaScript:

```javascript
let x = 5;
let name = "Alice";
```

JavaScript variables

Several ways to declare variables:

- `var`: function-scoped, can be redeclared and updated.
- `let`: block-scoped, can be updated but not redeclared.
- `const`: block-scoped, cannot be updated or redeclared (must be initialized).

Don't use `var` in modern code!

JavaScript primitive types

> Even though JavaScript is dynamically typed, it has a set of primitive types.

- **Number**: Represents both integers and floating-point numbers (e.g., `42`, `3.14`).
- **String**: Represents sequences of characters (e.g., `"Hello, World!"`).
- **Boolean**: Represents logical values (`true` or `false`).
- **Undefined**: Represents a variable that has been declared but not assigned a value.
- **Null**: Represents the intentional absence of any object value.

Checking types in JavaScript

> Use `typeof` operator to check the type of a variable:

```javascript
let x = 5;
console.log(typeof x); // Outputs: "number"
let name = "Alice";
console.log(typeof name); // Outputs: "string"
let isActive = true;
console.log(typeof isActive); // Outputs: "boolean"
let notDefined;
console.log(typeof notDefined); // Outputs: "undefined"
let emptyValue = null;
console.log(typeof emptyValue); // Outputs: "object" 🤮
```

JavaScript variables vs Java variables

In JavaScript, variables can hold values of any type and can change type at runtime:

```
let x = 5; // x is a number
x = "Hello"; // Now x is a string
```

This is in contrast to Java, where variables have a fixed type!

`console.log()`

Used for debugging and outputting values to the console:

```
let name = "Alice";
console.log("Hello, " + name); // Outputs: Hello, Alice
```

> **In our case the console is the browser's developer console.**

Similar to `System.out.println()` in Java, but more flexible (can log multiple values, objects, etc.).

JavaScript conditionals

All values are either "truthy" or "falsy":

- Falsy values: `false`, `0`, `""` (empty string), `null`, `undefined`, `NaN`.
- Everything else is truthy.

Equality operators:

- `==`: loose equality, performs type coercion (means it converts one or both values to a common type before making the comparison).
- `===`: strict equality, no type coercion.

JavaScript conditionals (contd.)

```
let x = 0;
if (x) {
    console.log("x is truthy");
} else {
    console.log("x is falsy");
}
```

Typically, you would use `===` for comparisons to avoid unexpected type coercion:

```
console.log(5 == "5"); // true (loose equality, type coercion)
console.log(5 === "5"); // false (strict equality, no type coercion)
```

JavaScript loops

Basic loops look similar to Java

Difference: for-in vs for-of:

- `for-in` : Iterates over the keys of an object (or indices of an array).
- `for-of` : Iterates over the values of an iterable (like an array or string).

```
const arr = ['a', 'b', 'c'];
for (const index in arr) {
    console.log(index); // Outputs: 0, 1, 2
}
for (const value of arr) {
    console.log(value); // Outputs: 'a', 'b', 'c'
}
```

JavaScript objects

Objects are collections of key-value pairs:

```
const person = {
    name: "Alice",
    age: 30,
    greet: () => console.log("Hello, " + person.name)
};
```

Access properties using dot notation or bracket notation:

```
console.log(person.name); // Dot notation
console.log(person["age"]); // Bracket notation
```

Objects in JavaScript vs Java

> **JavaScript objects are more flexible than Java objects**

> **With Java, you need to define a class with specific fields and methods. In JavaScript, you can create objects on the fly without class definitions.**

For example, you can add properties to an object at runtime:

```
person.city = "New York"; // Add new property
```

> No class definitions needed, can have properties added or removed at runtime.

JavaScript arrays

Arrays are ordered collections of values:

```
const numbers = [1, 2, 3, 4, 5];
```

Access elements using index (0-based):

```
console.log(numbers[0]); // Outputs: 1
```

Common array methods:

- `push()` : Add element to the end.
- `pop()` : Remove last element.
- `shift()` : Remove first element.
- `unshift()` : Add element to the beginning.

Arrays JavaScript vs Java

> **JavaScript arrays are more flexible than Java arrays**

- Can hold values of different types (e.g., numbers, strings, objects).

```
const mixedArray = [1, "Hello", { name: "Alice" }, [2, 3]];
```

- Can change size dynamically (no need to specify length).

```
const numbers = [1, 2, 3];
numbers.push(4); // Now numbers is [1, 2, 3, 4]
```

> The dynamic resizing reminds us of `ArrayList` in Java, but JavaScript arrays are even more flexible, since the elements can be of any type!

JavaScript arrays (contd.)

Arrays can also contain objects and even other arrays:

```javascript
const people = [
    { name: "Alice", age: 30 },
    { name: "Bob", age: 25 }
];

// Print names of all people
for (const person of people) {
    console.log(person.name); // Outputs: Alice, Bob
}
```

JavaScript arrays (contd.)

More array methods:

- `forEach()` : Iterate over elements.
- `map()` : Transform elements and return a new array.
- `filter()` : Filter elements based on a condition.
- `reduce()` : Reduce array to a single value.

```
const doubled = numbers.map(n => n * 2); // [2, 4, 6, 8, 10]
const evens = numbers.filter(n => n % 2 === 0); // [2, 4]
const sum = numbers.reduce((acc, n) => acc + n, 0); // 15
```

NOTE: They take functions as arguments!

JavaScript functions

Function declaration:

```
function myFunction(param1, param2) {
    return param1 + param2;
}
```

Function expression:

```
const myFunction = function(param1, param2) {
    return param1 + param2;
};
```

Arrow function:

```
const myFunction = (param1, param2) => param1 + param2;
```

JavaScript functions (contd.)

Functions are first-class citizens (can be assigned to variables, passed as arguments, returned from other functions).

```
function add(a, b) {
    return a + b;
}
const operate = (fn, x, y) => fn(x, y);
console.log(operate(add, 2, 3)); // Outputs: 5
```

Can have default parameters:

```
function greet(name = "Guest") {
    console.log("Hello, " + name + "!");
}
```

JavaScript functions vs Java methods

> **In Java, methods are tied to classes. In JavaScript, functions can exist independently of objects or classes.**

```
function add(a, b) {
    return a + b;
}
```

```
public int add(int a, int b) {
    return a + b;
}
```

> Notice that the Java method must be inside a class, while the JavaScript function can exist on its own, and can be passed around as a value.

JavaScript special features (contd.)

Destructuring:

A convenient way to extract values from arrays or properties from objects into distinct variables.

```
// Array destructuring
const rgb = [255, 200, 100];
const [red, green, blue] = rgb;
console.log(red, green, blue); // Outputs: 255 200 100

// Object destructuring
const person = { name: "Osman", age: 33 };
const { name, age } = person;
console.log(name, age); // Outputs: Osman 33
```

JavaScript special features (contd.)

Template Literals:

A way to create strings that can span multiple lines and include embedded expressions.

```
const person = { name: "Osman", age: 33 };
const { name, age } = person;

const greeting = `Hello, my name is ${name} and I am ${age} years old.`;

console.log(greeting);
// Outputs: Hello, my name is Osman and I am 33 years old.
```

DOM (Document Object Model)

> The DOM is a programming interface for HTML and XML documents. It represents the page as a tree of nodes.

JavaScript can manipulate the DOM to change the content and structure of web pages dynamically.

```
<section id="content"></section>

<script>
    const message = "Hello, World!";
    const content = document.getElementById("content");
    content.textContent = message;
</script>
```

Searching the DOM

We can search for elements in the DOM using various methods:

- `getElementById()` : Selects an element by its ID.
- `querySelector()` : Selects the first element that matches a CSS selector.

```
<section id="elem">
    <article class="card"></article>
</section>
<script>
    const elem = document.getElementById("elem");
    elem.style.color = "green";
    const card = document.querySelector(".card");
    card.innerHTML = "<h2>Card Title</h2>";
</script>
```

Searching the DOM

> `querySelectorAll()`: Selects all elements that match a CSS selector and returns a NodeList (which can be iterated over).

```html
<section>
    <article class="card"></article>
    <article class="card"></article>
    <article class="card"></article>
</section>
<script>
    const cards = document.querySelectorAll(".card");
    for (const card of cards) {
        card.innerHTML = "<p>This is a card.</p>";
    }
</script>
```

DOM manipulation - changing content, styles etc.

We can change the content, attributes, and styles of DOM elements.

```
<section id="content"></section>
<script>
    const content = document.getElementById("content");
    const message = "Hello, World!";
    content.textContent = message; // Change text content
    content.innerHTML = `<p>${message}</p>`; // Change HTML content
    content.style.color = "blue"; // Change CSS style
</script>
```

DOM manipulation - creating and removing elements

We can also create new elements and add them to the DOM, or remove existing elements.

```
<section id="content"></section>
<script>
    const content = document.getElementById("content");
    const newElement = document.createElement("div"); // Create new element
    newElement.textContent = "This is a new div.";
    content.appendChild(newElement); // Add to DOM
    // To remove an element:
    // content.removeChild(newElement);
</script>
```

Events

> JavaScript can respond to user interactions and other types of events.

```
<button id="myButton">Click me</button>
<script>
    const button = document.querySelector("#myButton");

    // Using an anonymous function:
    button.addEventListener("click", () => {
        console.log("Button was clicked!");
    });
    // or using a named function:
    button.addEventListener("click", handleClick);
    function handleClick() {
        console.log("Button was clicked!");
    }
</script>
```

Teaser: Fetching data from REST APIs

> JavaScript can fetch data from REST APIs using the `fetch()` function.

```javascript
fetch("https://api.example.com/data")
    .then(response => response.json())
    .then(data => {
        console.log(data);
    });
```

Or using async/await syntax:

```javascript
async function fetchData() {
    const response = await fetch("https://api.example.com/data");
    const data = await response.json();
    console.log(data);
}
fetchData();
```

Exercises