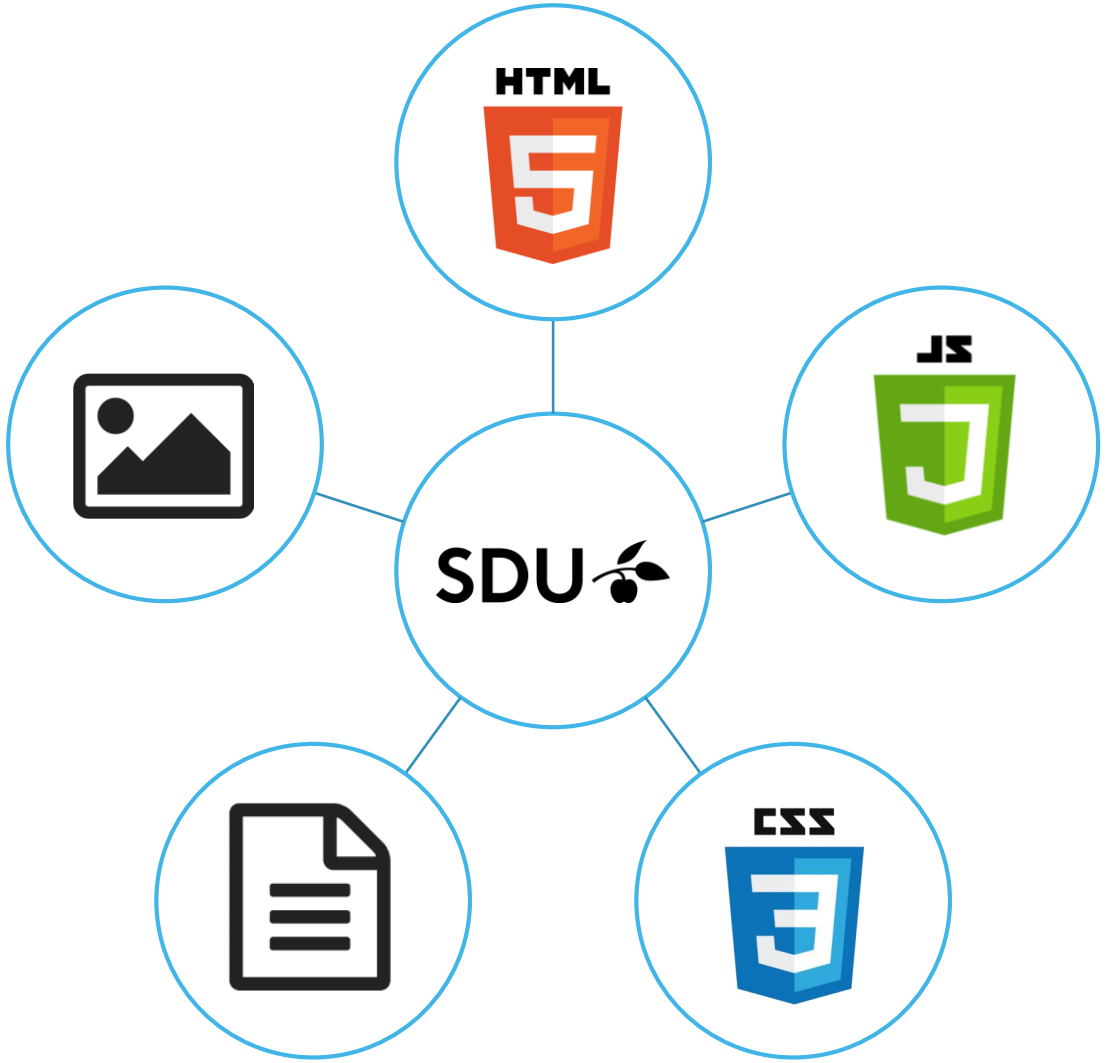
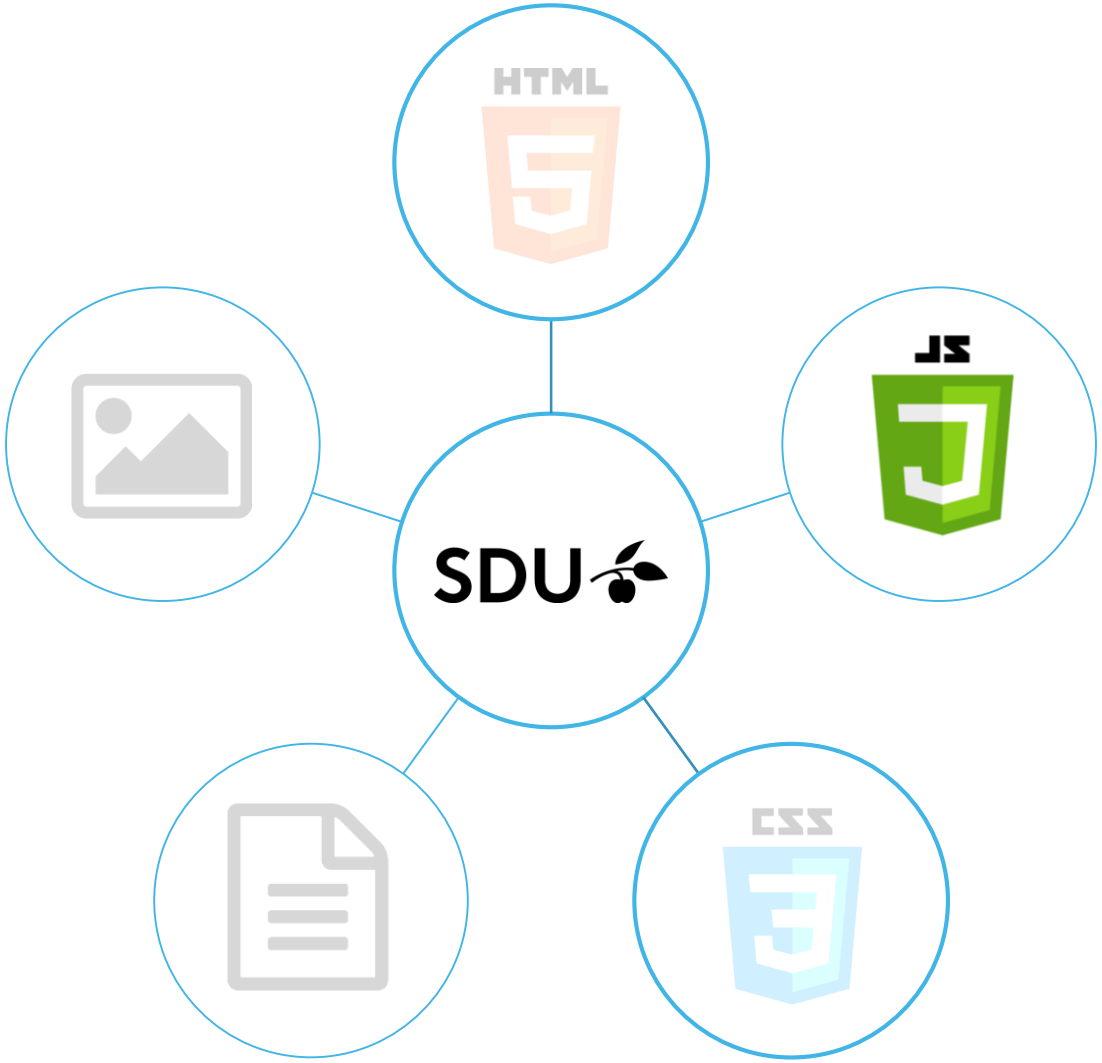
Javascript

13. marts 2018

What is a website?



What is a website?



HTML – HyperText Markup Language

Defines the structure of a webpage

CSS – Cascading Style Sheet

Defines the visual presentation of HTML elements

JS – Javascript

Handles user interaction and dynamic content

Information

Today we are gonna be using two files to supplement our exercises.

Open the files **lesson4.html** and **lesson4.js** in your preferred Code Editor

Unless stated otherwise, perform the exercises in the these files.

General information and History

Javascript – General

Introduced in 1995

Dynamically typed, imperative, functional, prototyped and object oriented

Today → ECMAScript 2015 / ES6

Primarily client-side, event driven, implementations vary… a lot…

Javascript and other languages

Javascript is to Java, what Carpet is to Car

I.e. not related.

Javascript is very different from the programming languages normally taught at universities.

History

Created by Brendan Eich in 1995 for Netscape Navigator 2

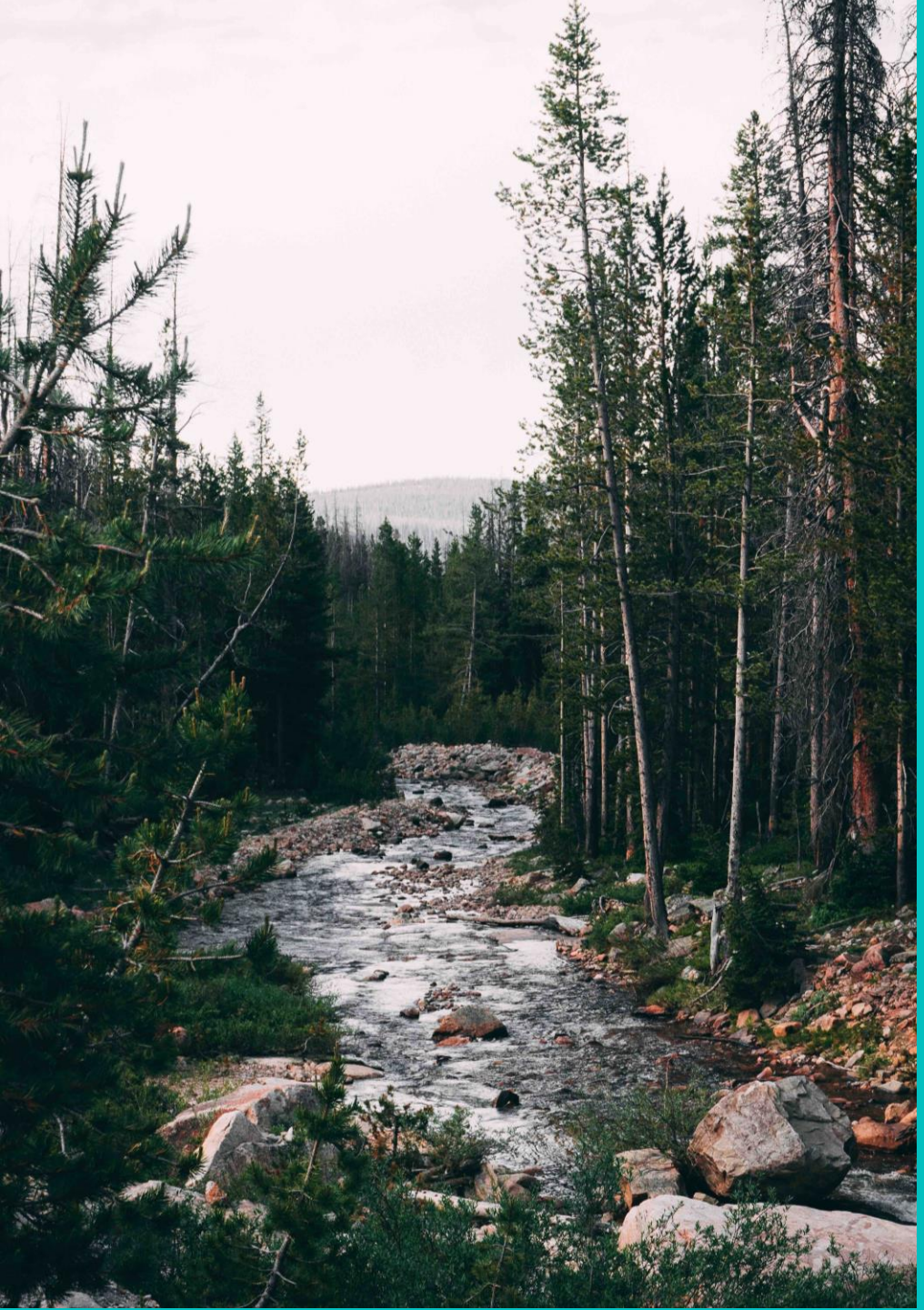
Initially called Mocha and later LiveScript

On release of 1.1, name changed to JavaScript

In 1997 Javascript 1.1 was submitted to ECMA as a proposal. TC39 was assigned to standardize the syntas and semantics of the language.

TC39 released ECMA-262 known as ECMAScript

ECMAScript is the basis for all Javascript implementations



Programming with JavaScript

Running JavaScript

There are two ways:

```
<script>
    alert("I'm an alert box")
</script>

<script type="text/javascript" src="/scripts/script.js"></script>
```

Variables – 4 types

Local: using the **var** or **let** keyword

Global: declared without var or attached directly to **window** object.

Constants: **const** keyword

We assign values to variables using **=** (assignment operator)

var vs. *let* - Global

Very similar outside function block

```
let globalLet = 'let';  // globally scoped
var globalVar = 'var'; // globally scoped
```

However *let* variables will not be added to **window** object, *var* will.

```
console.log(window.globalLet); // undefined
console.log(window.globalVar); // 'var'
```

Exercise
- Create some variables
- Print the variables to the console

10 minutes

var vs. *let* - Function

They are identical when used like this in a function block

```
function letAndVarInsideAFunctionBlock() {
    let letItRain = 'Let it rain!'; //function block scoped
    var varIsSame = 'Same....!'; //function block scoped
}
```

var vs. *let* - Block

Here lies the difference.

let is only visible in the *for()* loop

var is visible to the whole function

```
function usingLetInABlock() {
    //tuce is *not* visible out here

    for (let tuce = 0; tuce < 5; tuce++) {
        //tuce is only visible in here (and in the for() parentheses)
        //and there is a separate tuce variable for each iteration of the loop
    }

    //tuce is *not* visible out here
}

function usingVarInABlock() {
    //nish *is* visible out here

    for (var nish = 0; nish < 5; nish++) {
        //nish is visible to the whole function
    }

    //nish *is* visible out here
}
```

var vs. *let* - Redeclaration

Assuming **strict mode**, *var* will let you re-declare the same variable in the same scope. On the other hand, let will not:

```
'use strict';
let me = 'foo';
let me = 'bar'; // SyntaxError: Identifier 'me' has already been declared

'use strict';
var me = 'foo';
var me = 'bar'; // No problem, `me` is replaced.
```

var vs. *let* – My recommendation

Use *let*, I cannot see any reason to use *var* anymore.

Statements

Can be separated with new-line or semicolon (;)

```
function letAndVarInsideAFunctionBlock() {
    let letItRain = 'Let it rain!'; //semicolon not strictly necessary
    var varIsSame = 'Same....!' //still correct
}
```

Comments

Two ways:

- //
- /* */

```
// Usually used for single-line comments

/* Usually used for multi
   line
   comments */
```

Data types

JavaScript is a loosely typed language

Java and C# are strictly typed

Data types - Numbers

Numbers: 3, -3, 3.33, -3.33 – These are numbers. We don’t differentiate integers and floats.

- Addition, subtraction, division, multiplication and modulo division.
- Results are always promoted to floats, when possible.
- Assignment operators: +=, -=, /=, *= and %=
- Special operators: increment (++) and decrement (- -)

Data types - Strings

A series /sequence of characters from zero to infinity

Can contain any character

Can be created using single or double quotes

Use backslash (\ ) to escape special characters:

```
"A \"special\" word here" //Renders: A "special" word here
"Also tabs \t\t and word" // Renders: Also tabs          and word
```

Concatenation:

```
"Concat" + " using" + " +";
"Also " + " add" + myVar;
```

Data types - Booleans

True or **false**

Cases for True:

```
"<One space here>", "string", "undefined", 1, 2, 3, -3, -2, -1, true
```

Cases for False:

```
"<empty string>", undefined, null, void(0), 0, false
```

Data types - Arrays

Holds a collection of items

Can store any data types together in a single array

```
let array = [1, 2.3, "strings", { "key": "value" }, ["other", "arrays"]]
```

2 types of arrays: 1 dimensional and multi dimensional

```
let one = [1, 2, 3, 4];
let two = [[1, 2], [2, 3]];
```

Accessiong values in an array is done via index. Ranges from 0 to mins -1 (arrays length)

Data types - Objects

Are also known as Associative Arrays

Regarded as Key-Value pairs

Example:

```
let person = [];
person["given name"] = "Nicolai";
person["sur name"] = "Oksen";
person["age"] = 27;
```

As normal object:

```
let person = {
    "given name" : "Nicolai",
    "sur name" : "Oksen",
    "age": 27
};
```

Exercise

- Create variables of type
    - Array, Object, Number, Boolean and String
- Print them to the console
- Try overwriting variables with another type

15 minutes

Controlling the flow

Conditions is basically making decisions

If statements: Expects a Boolean expression

Example:

```
if (condition == true) {
    // Do something in here...
}
```

Controlling the flow (cont.)

Comparison operators:

- Less than (<)
- Greater than (>)
- Less than equal to (<=)
- Greater than equal to (>=)
- Equal to (==)
- Not equal to (!=)
- Not (!)

Multiple conditions operators:

- And (&&)
- Or (||)

Loops

Loops helps to minimize the repetition

- While: Simplest loop. While something is true, it runs
- Do-While: Runs at least once. Condition is evaluated after running body
- For: Succinct all the above

All loops should contain three things:

- Incrementer
- conditional expression
- logic to increase incrementer.

```
while (true) {
    // Runs forever
}

do {
    // Runs once
} while (false)

for(let i = 0; i < 100; i++) {
    // Runs 100 times
}
```

Writing code for later - Functions

If we want to re-run some code again and again from different places.

Predefined functions:

- alert()
- prompt()
- confirm()

Functions can return values.

```
function myFunction(param1, param2) {
    // do something in here,
    // maybe using params?
}
```

Objects – again!

Objects exists as a way of organizing variables and functions into logical groups.

Built in objects:

- String
- Date
- Math
- Boolean
- Number
- RegExp
- Object
- Array

```
let Person = new Object();
Person.givenName = "Nicolai";
Person.surName = "Oksen";
Person.age = 27;
Person.fullName = function() {
    return this.givenName + " " + this.surName;
}
Person.fullName(); // Returns "Nicolai Oksen"

// Alternative syntax:
let Person = {
    givenName: "Nicolai",
    surName: "Oksen",
    age: 27,
    fullName: function() {
        return this.givenName + " " + this.surName;
    }
}
Person.fullName(); // Returns "Nicolai Oksen"
```

Exercise

- Make a function that adds two numbers
    - Make it return the result
    - Catch the result in a variable
- Call the function multiple times
- Print the result to the console

15 minutes

Document Object Model - DOM

What is the DOM?

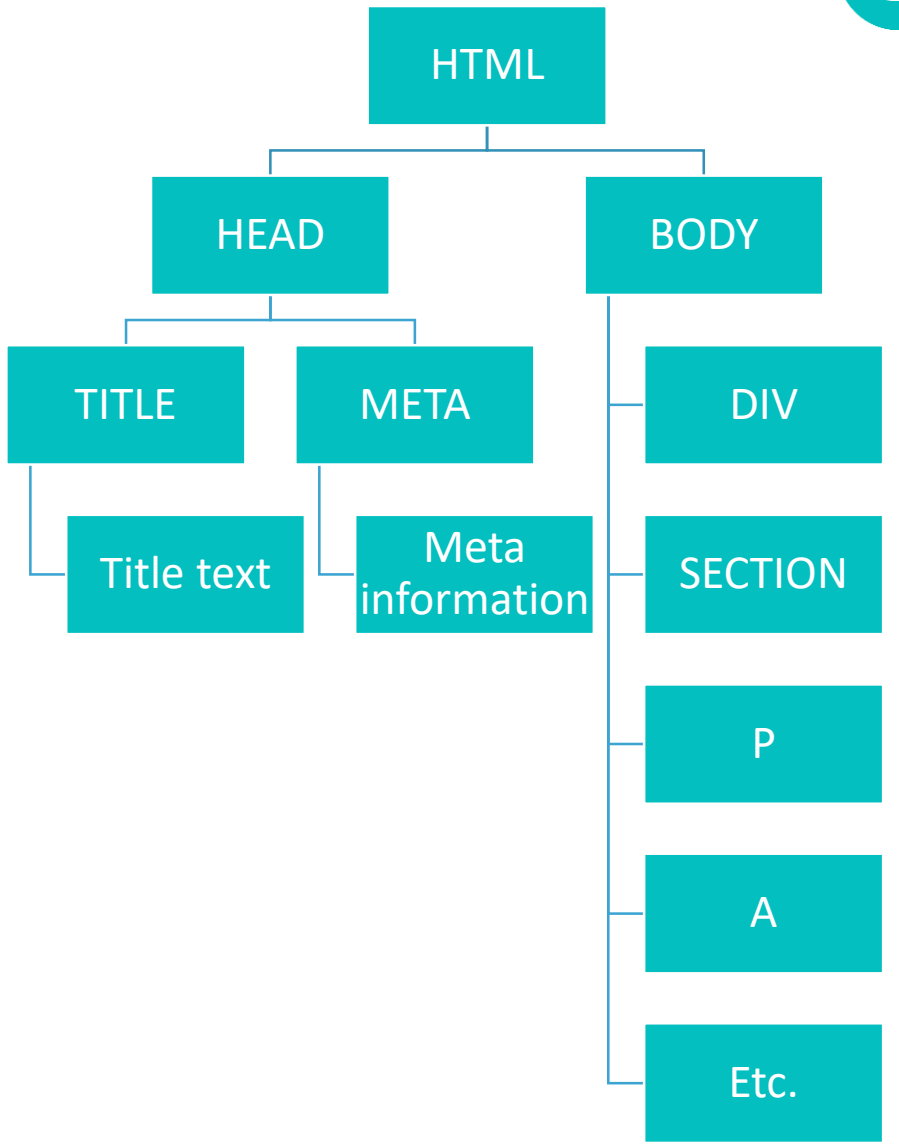
The Document Object Model (DOM) is a programming interface for HTML and XML documents

It represents the page, so that programs can change the document structure, style and content

The DOM represents the document as nodes and objects. This makes it easier for programming languages to interact with a page.

DOM Structure

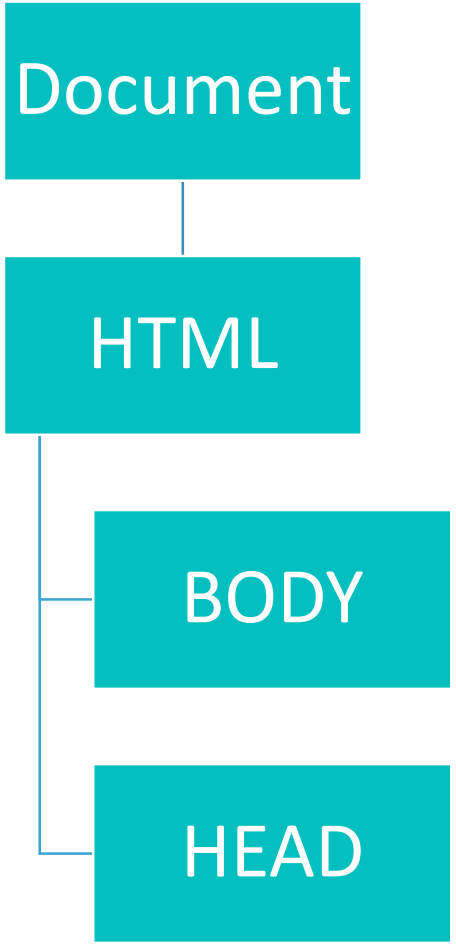
Converts the HTML document to a tree structure.



DOM Structure (cont.)

Even if the document is blank a *grand node* always exists!

It is called ***document***



DOM Manipulation

Accessing elements:

```
document.getElementById("id")
document.getElementsByClassName("class")
document.getElementsByName("name")
document.getElementsByTagName("elementName")
document.getElementsByTagNameNS("elementLocalName")
```

Alter properties (depends on the node type):

```
var myDiv = document.getElementsByTagName("div");
myDiv.innerHtml = "<p>Hello....</p>";
```

DOM Manipulation (cont.)

Querying elements:

```
// Returns the first paragraph element inside element with 'someId'
let firstElement = document.querySelector("#someId p");

// Returns all paragraph elements inside element with 'someId'
let arrayOfElements = document.querySelectorAll("#someId p");

// Also possible to query from a given element and downwards
let innerElement = firstElement.querySelector("span");
```

DOM Manipulation (cont.)

Traversal:

```
var myDiv = document.getElementById("divId");
myDiv.nextSibling();
myDiv.previousSibling();
myDiv.children();
myDiv.parent();
```

DOM Manipulation (cont.)

Manipulation style:

```
var myDiv = document.getElementById("divId");
myDiv.style.position = "left";
myDiv.style.background = "#ddd";
```

Manipulating classes:

```
var myDiv = document.getElementById("divId");
myDiv.className = "class-name"; // Overwrites existing classes
myDiv.className += " class-name"; // Adds to classes
```

Exercise
- Query different elements in lesson4.html
- Adjust styling to your liking
- Try multiple different accessor methods

20 minutes

Browser access

Browser Object Model (BOM)

Targets browser instead of HTML Document.

Offers up objects to allow access to browser information and manipulate information

Objects:

- window
- location
- navigator
- screen
- history

Events

JavaScript enables us to listen for DOM specific events and trigger our own custom code on those events.

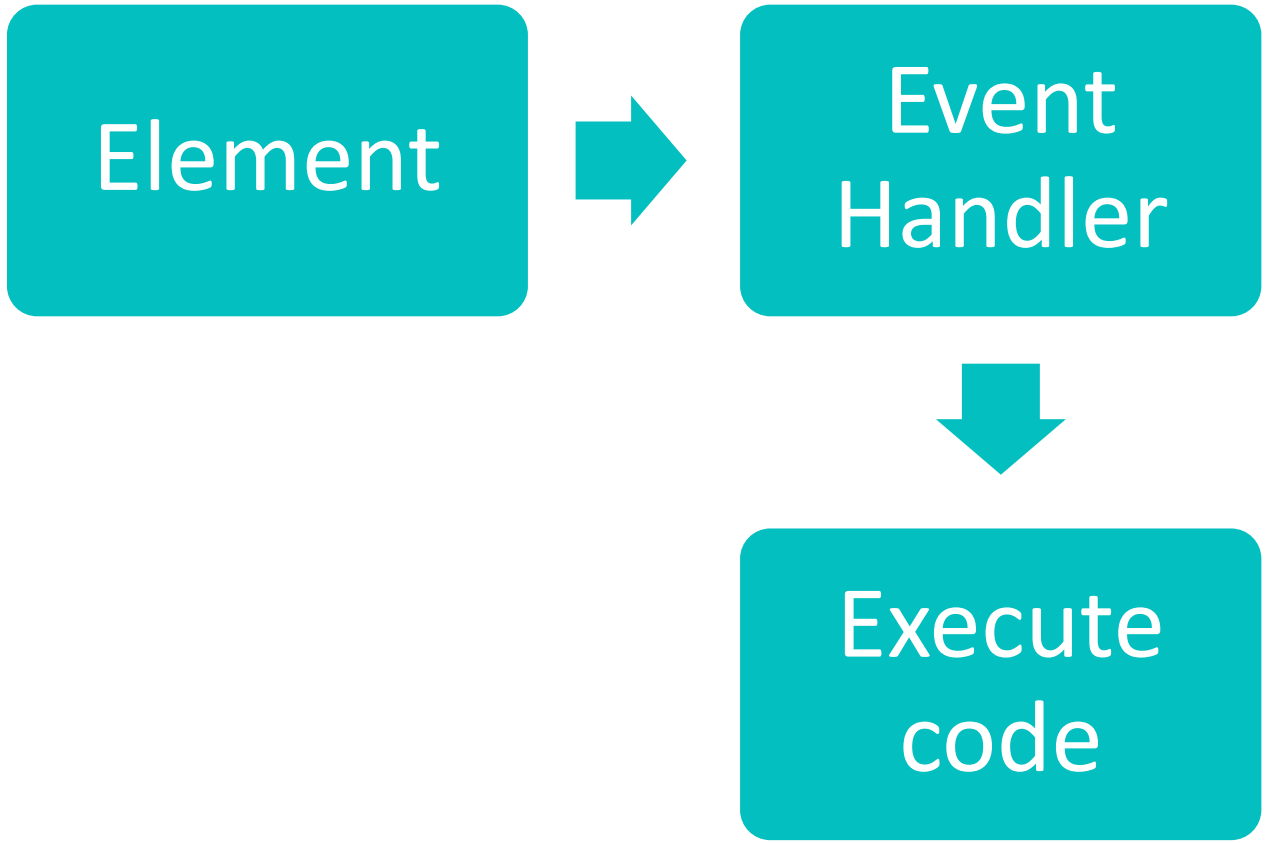
Events examples: clicking something, hovering mouse, pressing a keyboard key and submitting a form.

Events – All JavaScript events

onafterprint	onstorage	onkeypress	onmouseout	ondurationchange	
onbeforeprint	onunload	onkeyup	onmouseover	onemptied	
onbeforeunload	onblur	onclick	onmouseup	onended	onseeking
onerror	onchange	ondblclick	onmousewheel	onerror	onstalled
onhashchange	oncontextmenu	ondrag	onscroll	onloadeddata	onsuspend
onload	onfocus	ondragend	onwheel	onloadedmetadata	ontimeupdate
onmessage	oninput	ondragenter	oncopy	onloadstart	onvolumechange
onoffline	oninvalid	ondragleave	oncut	onpause	onwaiting
ononline	onreset	ondragover	onpaste	onplay	onerror
onpagehide	onsearch	ondragstart	onabort	onplaying	onshow
onpageshow	onselect	ondrop	oncanplay	onprogress	ontoggle
Onpopstate	onsubmit	onmousedown	oncanplaythrough	onratechange	
onresize	onkeydown	onmousemove	oncuechange	onseeked	

Events – Event handlers

The simplest way to hook into these events is through an event handler (which is a function)



Events – Event handlers (cont.)

Attaching to an event:

```
var myDiv = document.getElementById("divId");
myDiv.onclick = function(event) {
    // Event code here...
}
```

Events – default actions

Default actions, when talking about events, is what the browser normally does in response to events.

How to prevent?

- Return *true* or *false*

Events – this object

Whenever entering an event handler, the *this* object is populated as the clicked HTML element, represented as an object.

We cannot change the value of *this*

Events – Event listeners

Firstly – what is wrong with event handlers?

- Well, you can't assign multiple handlers to a single event! Remember it was assigned using **=**
- I.e. in the image only *eventHandler2* will be called

```
var myDiv = document.getElementById("divId");
myDiv.onclick = eventHandler1;
myDiv.onclick = eventHandler2;
```

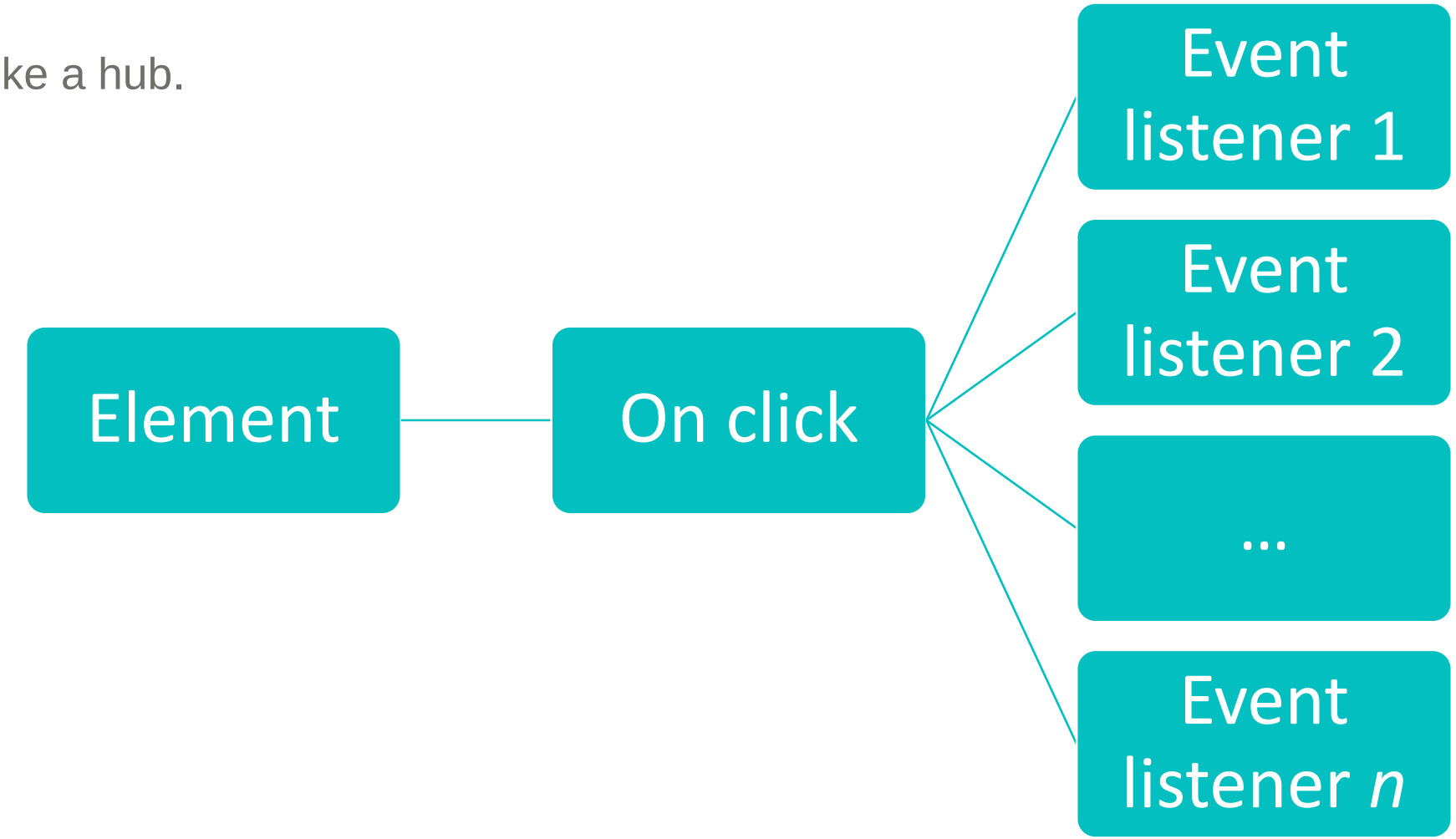
Using event listeners, we can assign

As many as we want.

All assigned event listeners will be called.

Events – Event listeners (cont.)

It works like a hub.



Events – Attaching event listeners

```
var myDiv = document.getElementById("divId");
myDiv.addEventListener("onlick", eventHandler1, false);
myDiv.addEventListener("onlick", eventHandler2, false);
```

Events – The event parameter

An event parameter is passed to all event listener functions.

```
function eventHandler1(event) {
    // event handler code.
    event.preventDefault();
    event.stopPropagation();
}
```

The event parameter has some important methods. Two are seen above.

Exercise: What does the above do? 10 minutes!

Events – Detaching events

```
myDiv.removeEventListener("onclick", eventHandler1, false);
```

Events - Propagation

What is event propagation?

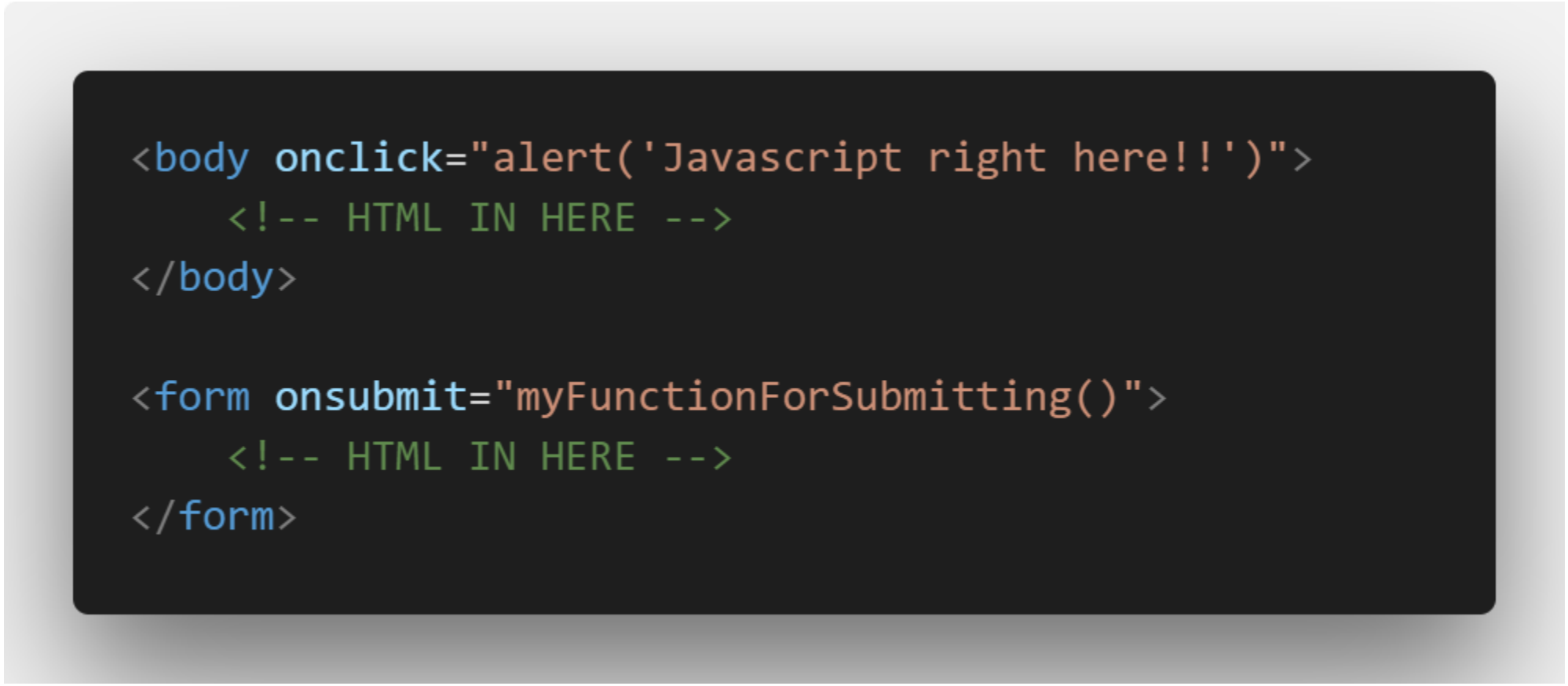
- Well you already researched the method!

Propagation: *transmission* or *dissemination*.

Event propagation happens in three phases

1. Capture: Document to element
2. Target: Element which triggered event
3. Bubbling: Element to document

Events - HTML


<body onclick="alert('Javascript right here!!')">
<!-- HTML IN HERE -->
</body>
<form onsubmit="myFunctionForSubmitting()">
<!-- HTML IN HERE -->
</form>

Exercise

- Create event handlers/listeners for onclick
    - Or any other event you like
- Execute the event on lesson4.html and observe that your function is called.

15 minutes

AJAX
Asynchronous JavaScript and XML

What is AJAX?

As mentioned is stands for **A**synchronus **J**ava**S**cript and **X**ML

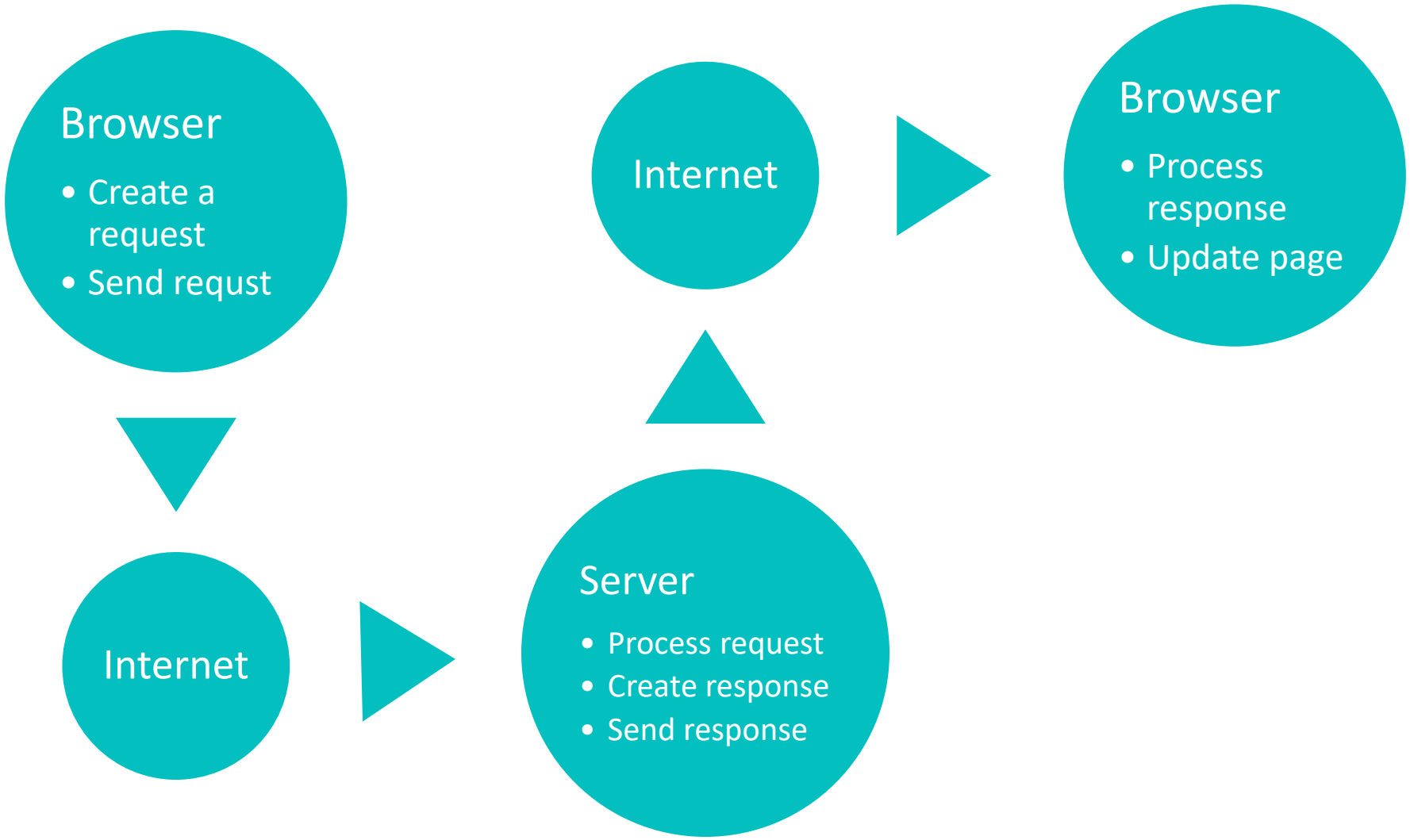
It is a technique aimed at making better and more interactive websites

Conventional websites transfer data synchronously between server and client

In essence AJAX allows us to update a webpage with new data, without reloading the page

AJAX is a misleading name, it is equally common to transport data using JSON, as compared to XML.

How it works


Browser
• Create a request
• Send requst
Internet
Server
• Process request
• Create response
• Send response
Internet
Browser
• Process response
• Update page

Working with AJAX

```
// Create an empty request
let xhr = new XMLHttpRequest();
// Supply information about endpoint and method
xhr.open("GET", "api/data", true);
// Send the request
xhr.send();
```

Adding an event listener

We have seen something like this before!

```
// Create an empty request
let xhr = new XMLHttpRequest();
// Add event listener to request, will be called whenever the state changes
xhr.addEventListener("readystatechange", callbackFunction, false);
// Supply information about endpoint and method
xhr.open("GET", "api/data", true);
// Send the request
xhr.send();
```

… and event handlers

```
// Create an empty request
let xhr = new XMLHttpRequest();
// Add event handler to request, will be called whenever the state changes
xhr.onreadystatechange = eventHandler;
// Supply information about endpoint and method
xhr.open("GET", "api/data", true);
// Send the request
xhr.send();
```

Short side-track – HTTP Verbs

HTTP Verb	CRUD	Entire Collection (e.g. /endpoint)	Specific Item (e.g. /endpoint/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update /Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update /Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Retrieving data

```
// Create an empty request
let xhr = new XMLHttpRequest();
// Add event handler to request, will be called whenever the state changes
xhr.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
// Supply information about endpoint and method
xhr.open("GET", "api/data", true);
// Send the request
xhr.send();
```

Sending data

```
// Create an empty request
let xhr = new XMLHttpRequest();
// Add event handler to request, will be called whenever the state changes
xhr.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 201) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
// Supply information about endpoint and method
xhr.open("POST", "api/data", true);
let data = {
    fname: "Nicolai",
    lname: "Oksen"
}
// Send the request
xhr.send(JSON.stringify(data) || null);
```

JavaScript Object Notation

What is JSON?

Stands for **J**ava**S**cript **O**bject **N**otation

Is pronounced as Jason

Invented to enable easy transfer between server and client

XML serves the same purpose, but is not as lightweight

JSON is "self-describing" and easy to understand

Why use JSON?

JSON format is text only. Therefore it is easy to send to and from server

Since it is text only, it easy to use for any programming language

JavaScript natively support converting JSON to JavaScript objects

Syntax rules

Data is always in name/value pairs

Data is separated by commas

Curly braces holds objects

Square brackets hold arrays

JSON Data

JSON Data - Objects


```
// JSON
{
    "name" : "value"
}
```

```
// JavaScript
{
    name : "value"
}
```

Data types in JSON

In JSON, *values* must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- Null

In JavaScript values can be all of the above, plus any valid JavaScript expression, including:

- a function
- a date
- undefined

In JSON, *string values* must be written with double quotes:

JSON Example

JSON to JavaScript… and back again!

Since JSON and JavaScript objects are based on the same syntax, it is easy to move between each format:

```
let object = {
    givenName : "Nicolai",
    surName : "Oksen"
}

let jsonString = JSON.stringify(object);
// Produces: "{"givenName":"Nicolai","surName":"Oksen"}"

object = JSON.parse(jsonString);
// Produces { givenName: "Nicolai", surName: "Oksen" }
```

JSON vs. XML

```json
{
    "employees": [
        {
            "firstName": "John",
            "lastName": "Doe"
        },
        {
            "firstName": "Anna",
            "lastName": "Smith"
        },
        {
            "firstName": "Peter",
            "lastName": "Jones"
        }
    ]
}
```

```xml
<employees>
    <employee>
        <firstName>John</firstName>
        <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName>
        <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName>
        <lastName>Jones</lastName>
    </employee>
</employees>
```

JSON vs. XML (cont.)

JSON is Like XML Because

Both JSON and XML are "self describing" (human readable)

Both JSON and XML are hierarchical (values within values)

Both JSON and XML can be parsed and used by lots of programming languages

Both JSON and XML can be fetched with an XMLHttpRequest

JSON is Unlike XML Because:

JSON doesn't use end tag

JSON is shorter

JSON is quicker to read and write

JSON can use arrays

JSON vs. XML (cont.)

Is JSON better than XML?

That is up to you!

Benefits of JSON

- Easy to parse to a JavaScript object

With XML, you have to access it via a DOM and convert

elements and values to a JavaScript object

I personally prefer JSON, but the syntax rules and ceremony around XML makes it a better choice sometimes.

Exercise

- Create a function to calculate find Fibonacci numbers.
- You should be able to enter the amount of Fibonacci numbers you wish
- Allow entering a number on a HTML page and execute function with a button
- Add requested Fibonacci numbers to HTML page

```
F(n) = F(n-1) + F(n-2)
```

```
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
```


codecademy

Steps

1. Create an account on codecademy.com (You should have done that)
2. Start the course ”Learn JavaScript”

This course should teach everyone without any prior knowledge, the fundementals of JavaScript

Questions?