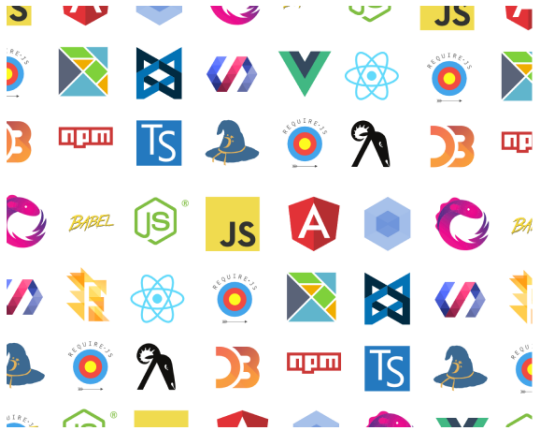
2008

2021



JavaScript

CSC309

Kianoosh Abbasi

UNIVERSITY OF TORONTO

Fall 2024

So far

- How web works
  - Client vs server
  - HTTP
- HTML tags and elements
  - Describes what should be on our page
- CSS styles
  - Describes how elements should look like
  - More on styles later in the course

This session

- JavaScript history and syntax
- Scope and closure
- Dynamic web pages

[Version 2.02]

File Edit View Go Bookmarks Options Directory Window Help

Back Forward Home Reload Images Open Print Find Stop

Location: about:

What's New! What's Cool! Handbook Net Search Net Directory Software

Netscape Navigator (TM)

Version 2.02

This software is subject to the license agreement set forth in the license. Please read and agree to all terms before using this software.

Report any problems through the feedback page.

NETSCAPE

JAVA COMPATIBLE

Contains Java™ software developed by Sun Microsystems, Inc.

Brendan Eich

RSA PUBLIC KEY CRYPTOSYSTEM

Contains security software from RSA Data Security, Inc.

This version supports International security with RSA Public Key Cryptography, MD2, MD5, RC4.

Java vs JavaScript

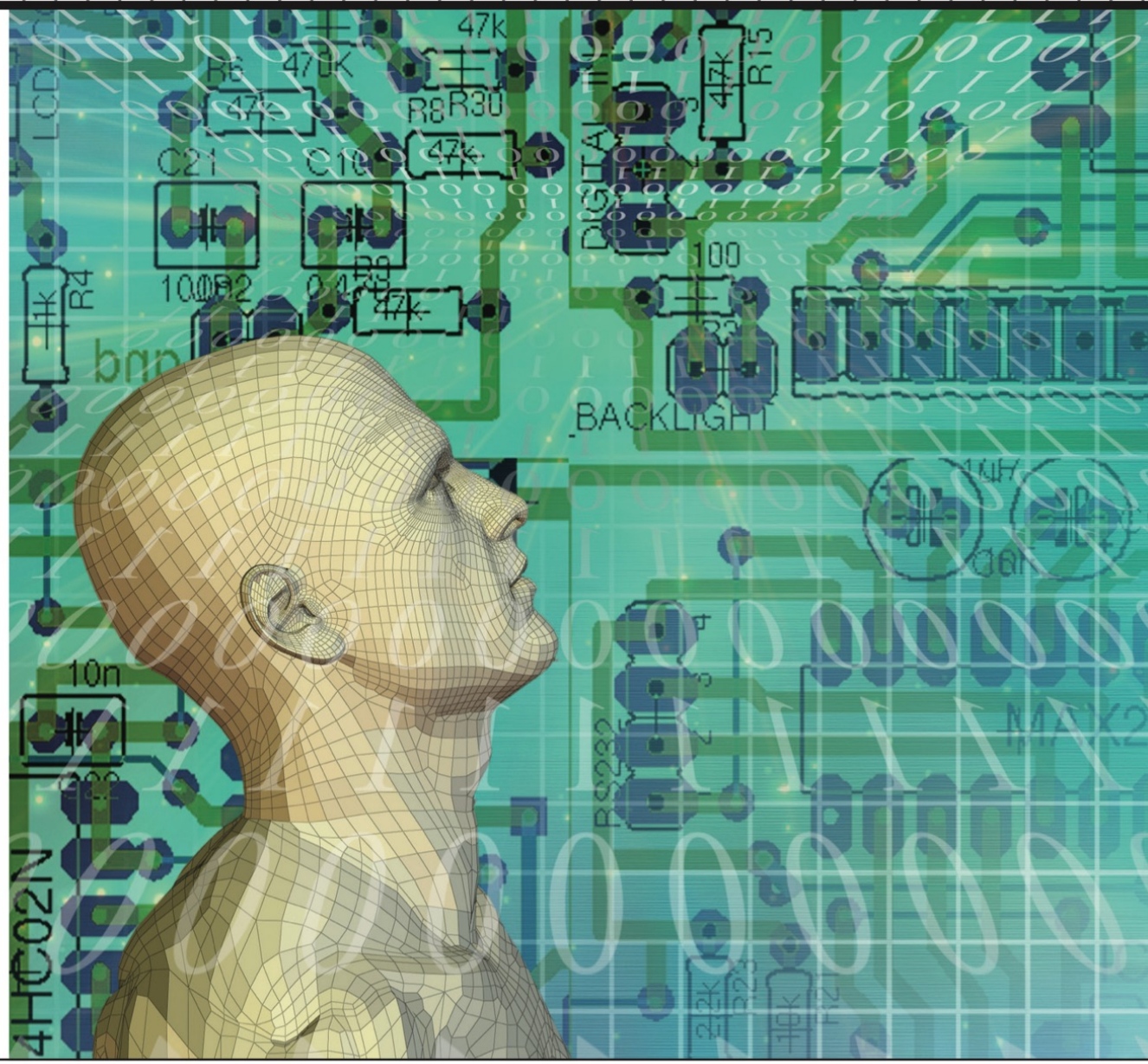
- Eich's script language had a somewhat similar syntax to Java
  - The C-style syntax was very popular in the 90s

- Netscape-Sun deal:
  - Netscape browser will support Java apps
  - Eich's language will be called JavaScript!

- No further relevance between Java and JavaScript!

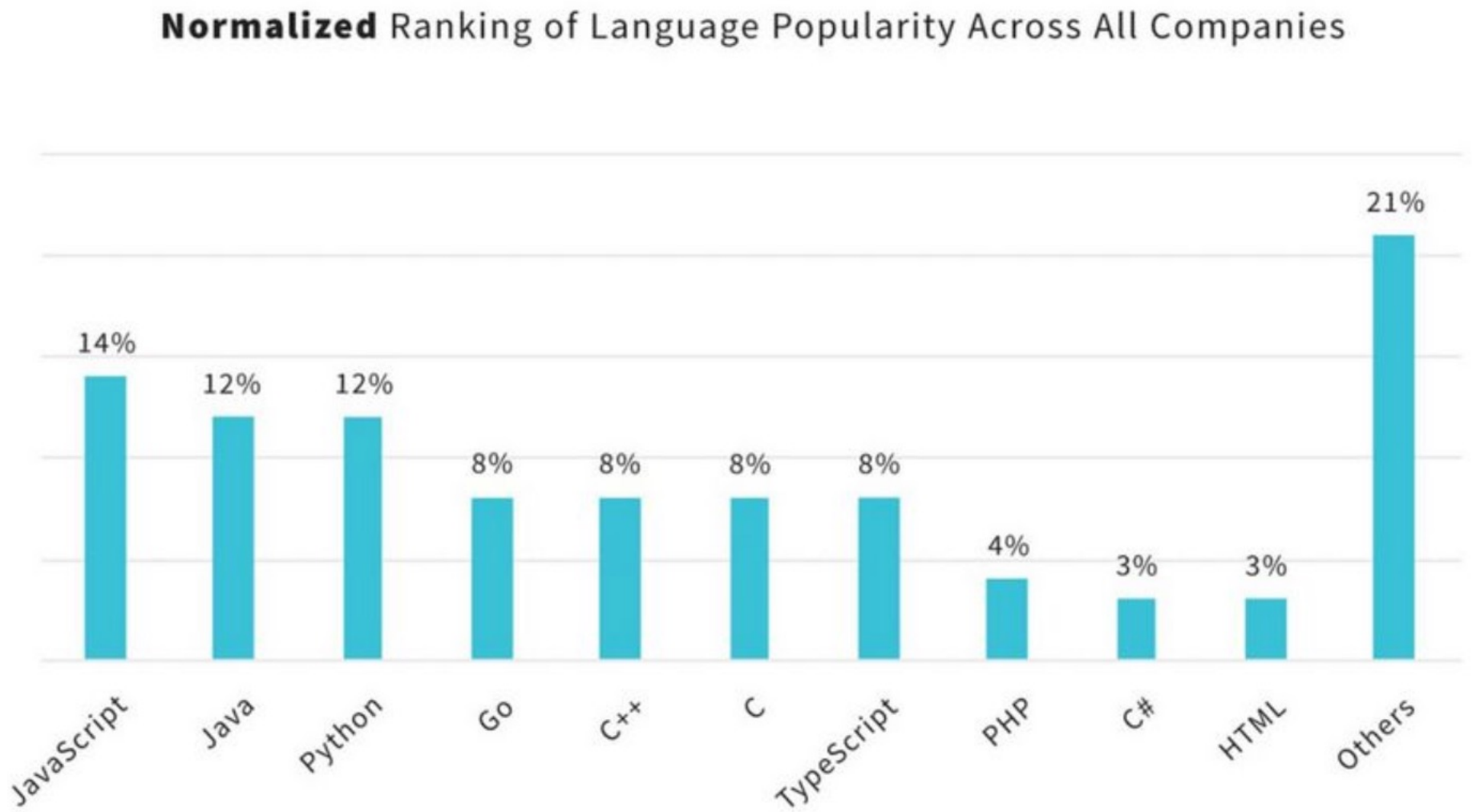
JavaScript: Designing a Language in 10 Days

Charles Severance
University of Michigan



There we go!

Visit: https://madnight.github.io/githut



Source: https://solutionshub.epam.com/blog/post/programming-language-popularity-on-github

- JavaScript is a scripting language
  - Interpreted at runtime
  - No JAR or exe file

- All browsers have a JS interpreter

- Not exclusive to client-side web applications!
  - Example: NodeJS, Next.js, Express, etc.
  - JS is just a language! You can sort an array of integers with it.

- JavaScript changed the web forever!

Syntax

- Declaring variables

```
var x = 5;
var y = "hello";
console.log(x + y);
```

- Data types:
  - Number, string, boolean, undefined
  - Object, function

- JS is dynamically-typed (like Python)

Objects

- Examples:

```
var cars = ["Saab", "Volvo", "BMW"];
var person = {firstName: "John", lastName: "Doe", age: 50, eyeColor: "blue"};
var ref = null;
```

- Ever heard of JSON?
  - Stands for JavaScript Object Notation
  - The most popular way to serialize all kind of objects

- Note: null is different than undefined
  - `typeof(null)` returns object, while `typeof(undefined)` returns undefined!

Properties

- Examples:

```
person.firstName = "Joe"
person["lastName"] = "Jordan"
cars[0] = 1
cars.push(2323)
```

- Objects (including arrays) are the only mutable types in JavaScript

Functions

- Syntax:

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

- Properties can be functions (methods)

```
var obj = {f: function(x) {
   return x + 2
}}

cars.clear = function(){
    this.length = 0;
}
```

Classes

Visit https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

- A template for creating objects
- Are in fact special functions
  - Check `typeof(Rectangle)`
- Classes support inheritance

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  // Getter
  get area() {
    return this.calcArea();
  }
  // Method
  calcArea() {
    return this.height * this.width;
  }
}

const square = new Rectangle(10, 10);

console.log(square.area); // 100
```

Conditions

- If statements:

```
if (typeof(cars[0]) === "number" && cars[0] < 0)
    cars[0] *= -1
else
    console.log("Bad element")
```

- Important: notice ===
  - Visit https://codeahoy.com/javascript/2019/10/12/==-vs-===-in-javascript/

More statements

- While loops:

```
while (cars.length > 0){
    cars.pop()
}
```

- Switch statement:

```
switch(cars[0]){
    case 1:
        console.log("int")
        break
    case "name":
        console.log("str")
        break
    case x:
        console.log("var " +x);
        break
    default:
        console.log("none")
}
```

For loops

- Classic for loop:

```
for (var i=0; i<10; i++)
    console.log(i * i * i)
```

- Iterable objects:

```
for (name of names)
    console.log("There is a " + name)
```

- Array-specific forEach:

```
names.forEach(function(name, index){
    console.log(name + " at index " + index)
})
```

Scope

Visit: https://www.w3schools.com/js/js_scope.asp

- Three types of scope:
  - Global scope
  - Function scope
  - Block scope

- Global scope
  - Outside any function
  - Variables can be accessed from anywhere in the program

Function scope

- Variables defined anywhere inside a function are local to that function
- Can be used anywhere inside that function
- Cannot be used outside that function

```
// code here can NOT use carName

function myFunction() {
  var carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Code by Sadia Sharmin (rainsharmin.com)

Block scope

- To limit a variable to its block inside the function, use let

```
function f(n){
    if (n > 10){
        var tmp = 2;
    }
    // tmp CAN be accessed here
}
```

```
function f(n){
    if (n > 10){
        let tmp = 2;
    }
    // tmp can NOT be accessed here
}
```

Codes by Sadia Sharmin (rainsharmin.com)

Let vs var

- At global and function scopes, let and var work similarly
- var supports redeclaration, while let does not
- Both support re-assignment. Use const to disallow it
- let is more like regular variables in other languages
  - Preferred over var

Arrow functions

- A more convenient way to define functions
- Almost equivalent to regular functions
  - More on that later

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

const arrow = (a, b) => {
    return a + b;
}

const conciseArrow = (a, b) => a + b;
```

Simplify the code!

- Today, for loops are rarely used
- Instead of a for loop, use forEach or map
- Example:

```
var names = ["ali", "hassan"]
names.forEach((item, index) => console.log(item + " at " + index))
upper = names.map(item => item.toUpperCase())
```

Simplify even further

- Use the filter method to take out elements with a specific condition

- Example:

```
let students = [{name: "John", id: 1}, {name: "Ali", id:2}]
let john = students.filter(item => item.name === "John")
```

- reduce lets you do a lot of cool things with just 1 inline arrow function

- Example:

```
let maxCredit = employee.reduce(
    (acc, cur) => Math.max(cur.credit, acc), Number.NEGATIVE_INFINITY)
```

Power of arrow functions!

Regular functions

```
var totalJediScore = personnel
  .filter(function (person) {
    return person.isForceUser;
  })
  .map(function (jedi) {
    return jedi.pilotingScore + jedi.shootingScore;
  })
  .reduce(function (acc, score) {
    return acc + score;
  }, 0);
```

Arrow functions

```
const totalJediScore = personnel
  .filter(person => person.isForceUser)
  .map(jedi => jedi.pilotingScore + jedi.shootingScore)
  .reduce((acc, score) => acc + score, 0);
```

Source: https://medium.com/poka-techblog/simplify-your-javascript-use-map-reduce-and-filter-bd02c593cc2d

Destructuring

Visit https://dmitripavlutin.com/javascript-object-destructuring/

```
const hero = {
  name: 'Batman',
  realName: 'Bruce Wayne'
};

const { name, realName } = hero;

name;     // => 'Batman',
realName; // => 'Bruce Wayne'
```

```
const hero = {
  name: 'Batman',
  realName: 'Bruce Wayne'
};

const { name, ...realHero } = hero;

realHero; // => { realName: 'Bruce Wayne' }
```

```
const heroes = [
  { name: 'Batman' },
  { name: 'Joker' }
];

const names = heroes.map(
  function({ name }) {
    return name;
  }
);

names; // => ['Batman', 'Joker']
```

Subtlety

- Regular functions have their own `this` value

- The object that called the function
  - Methods and event listeners (described in later slides): the actual object/element
  - Global function: global object (window)

- Arrow functions do not have their own `this`

- Do not use arrow functions as object method or even listener
  - You can use them as class methods though. PERFECTLY WEIRD ISN'T IT?

- However, unlike regular functions, they can bind (capture) `this` like any other closure value

```
const person = {
    name: 'Kianoosh',
    sayHi() {
        setTimeout(() =>
            console.log(this.name + " says hi!"), 500);
    }
}
```

- For more information, visit
  https://www.javascripttutorial.net/es6/when-you-should-not-use-arrow-functions

Client-side JavaScript

```
alert("Are you REALLY sure you want to leave??")
```

Where to put JS

- JS code should be placed inside the <script> tag

- Inline JS:

```
<script>
   console.log(1 + 2 + 3)
</script>
```

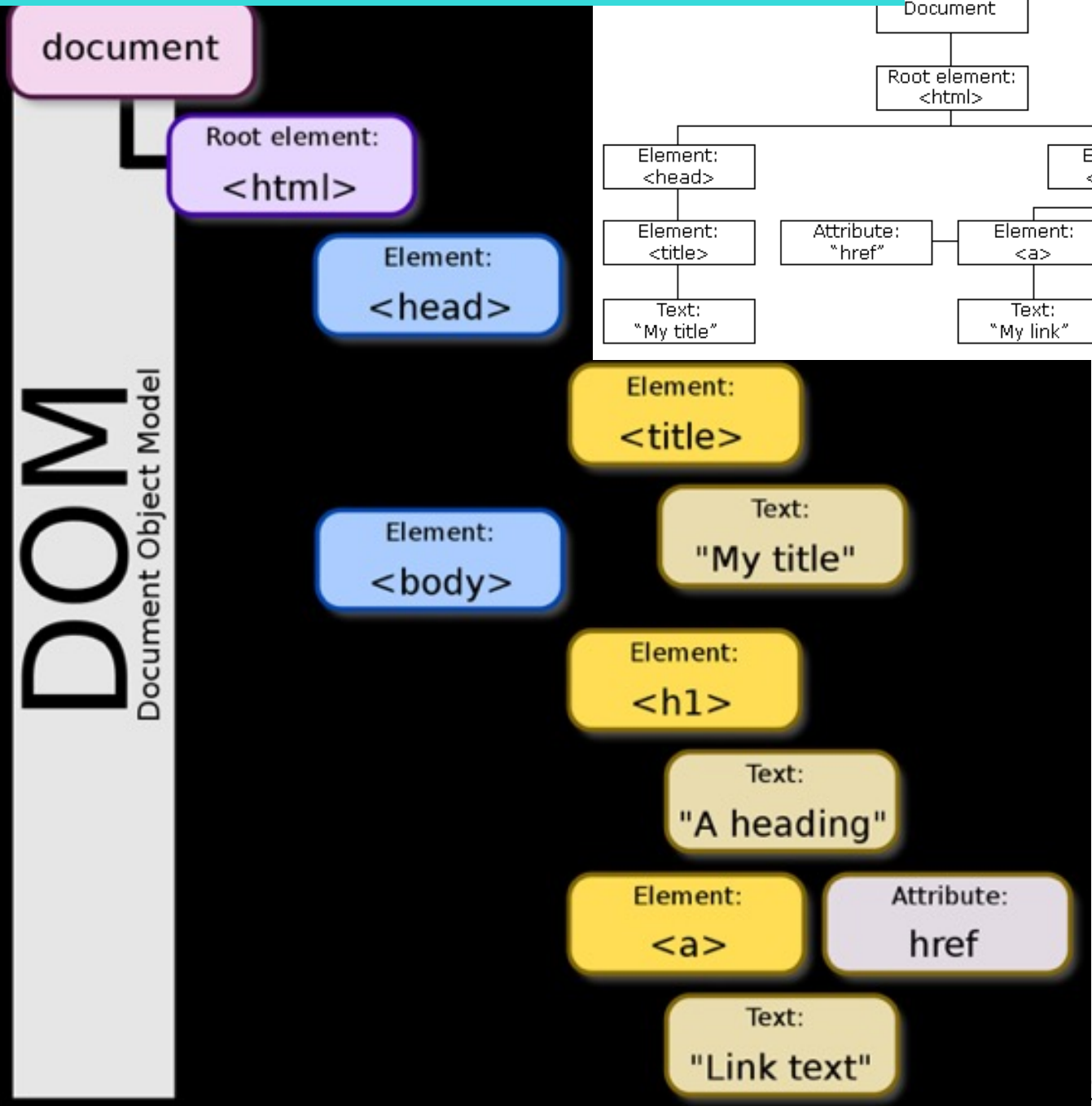
- JS file

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

Document object model

- Browser creates the DOM tree of the page
- Each element is a node
- Child elements are children of the parent node
- Scripts access DOM through the document variable

```
<html>
  <head>
    <title>My title</title>
  </head>
  <body>
    <h1>A heading</h1>
    <a href="...">Link
    text</a>
  </body>
</html>
```



Getting elements

- Various ways to get an element

```
document.getElementById("st-2")
document.getElementsByClassName("ne-share-buttons")
document.getElementsByTagName("ul")

document.querySelector("#submit-btn")
document.querySelectorAll(".col-md-12")
```

- Good exercise at:

  https://javascript.info/task/find-elements/table.html

Navigating through DOM

- Relevant nodes can be accessed through properties

```
parentNode, firstChild, lastChild, childNodes, nextSibling
```

- Example

```
let img = document.querySelector("#my-image")
let par = img.parentNode

console.log(par.childNodes.length)
```

Manipulating elements

- Element properties
  - innerHTML, style, getAttribute()

- Example

```
let body = document.body
body.innerHTML = "<h3>hello!</h3>"

h3 = document.getElementsByTagName("h3")
h3.style.color = "green"
h3.setAttribute("class", "title")
console.log(h3.getAttribute("style"))
```

Events

Visit https://www.w3schools.com/tags/ref_eventattributes.asp

- Various events are monitored by the browser

- document events

  `onload, onkeydown, onkey, …`

- Element events

  `onclick, onmouseover, ondrag, oncopy, onfocus, onselect, onsubmit, …`

Event listeners

- You can define a function

```
h3.onclick = function() {
    this.innerHTML = "you just clicked on me!"
}
```

- Alternative:

```
<script>
    function h3click(h3){
        h3.style.color = "blue"
    }
</script>

<h3 onclick="h3click(this)" onmouseover="console.log(new
Date())"></h3>
```

Exercise: A form with client-side validation

- Examples:
- Checks if a security question is answered correctly
- Checks if the email input is valid
- Checks password and repeat password are the same

- Errors should appear dynamically and disappear if user has fixed the issue

Next session

- Modern architecture of web apps
  - Frontend & backend
  - APIs
- Server-side JavaScript
  - JS projects with Node.js
- Intro to Next.js