

# **FRONTEND DEVELOPMENT WINTERSEMESTER 2020**

# ABOUT/CONTACT

- » Thomas Mayrhofer (@webpapaya)
- » E-Mail: [tmayrhofer.1ba@fh-salzburg.ac.at](mailto:tmayrhofer.1ba@fh-salzburg.ac.at)

# ROADMAP

- » 5.11. 4 EH
  - » JS History Intro etc.
  - » Modules
  - » build setup
- » 11.11. 2EH
  - » Asynchronous JS
- » 24.11. 2EH
  - » ES 6

# ROADMAP

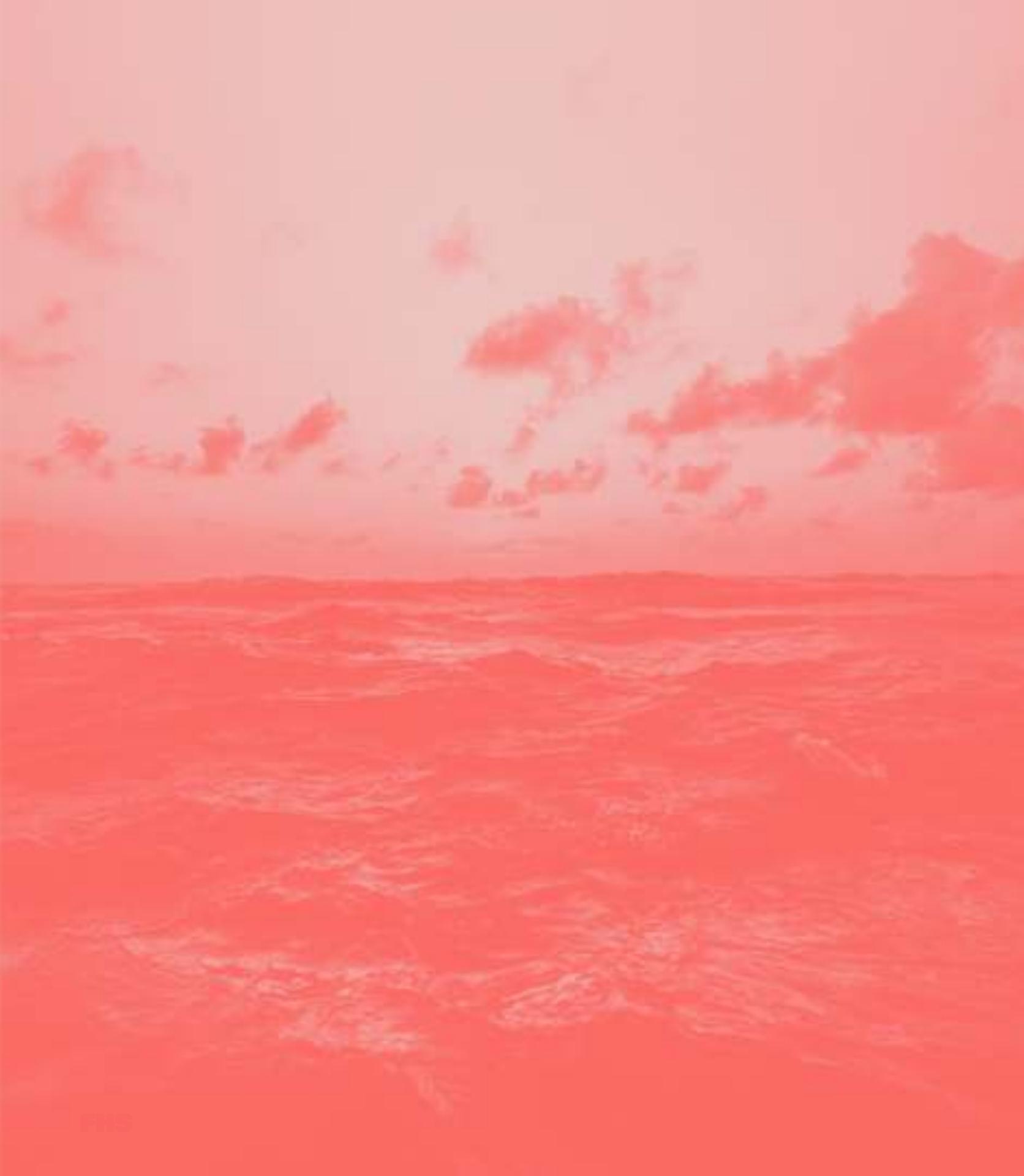
- » 3.12. 2EH
  - » Single Page App Routing
- » 17.12. 3EH
  - » JS Build tools
- » 22.12. 2EH
  - » Recap
  - » Gimme your questions!

# GRADING

- » 50% Homework
- » 50% Exam (Date will be sent to you)
- » Both positive

# HOMEWORK

- » can be done in pairs
- » hand-in via email  
`tmayrhofer.lba@fh-salzburg.ac.at`
- » email contains link to git repository
- » name of students who worked on the assignment



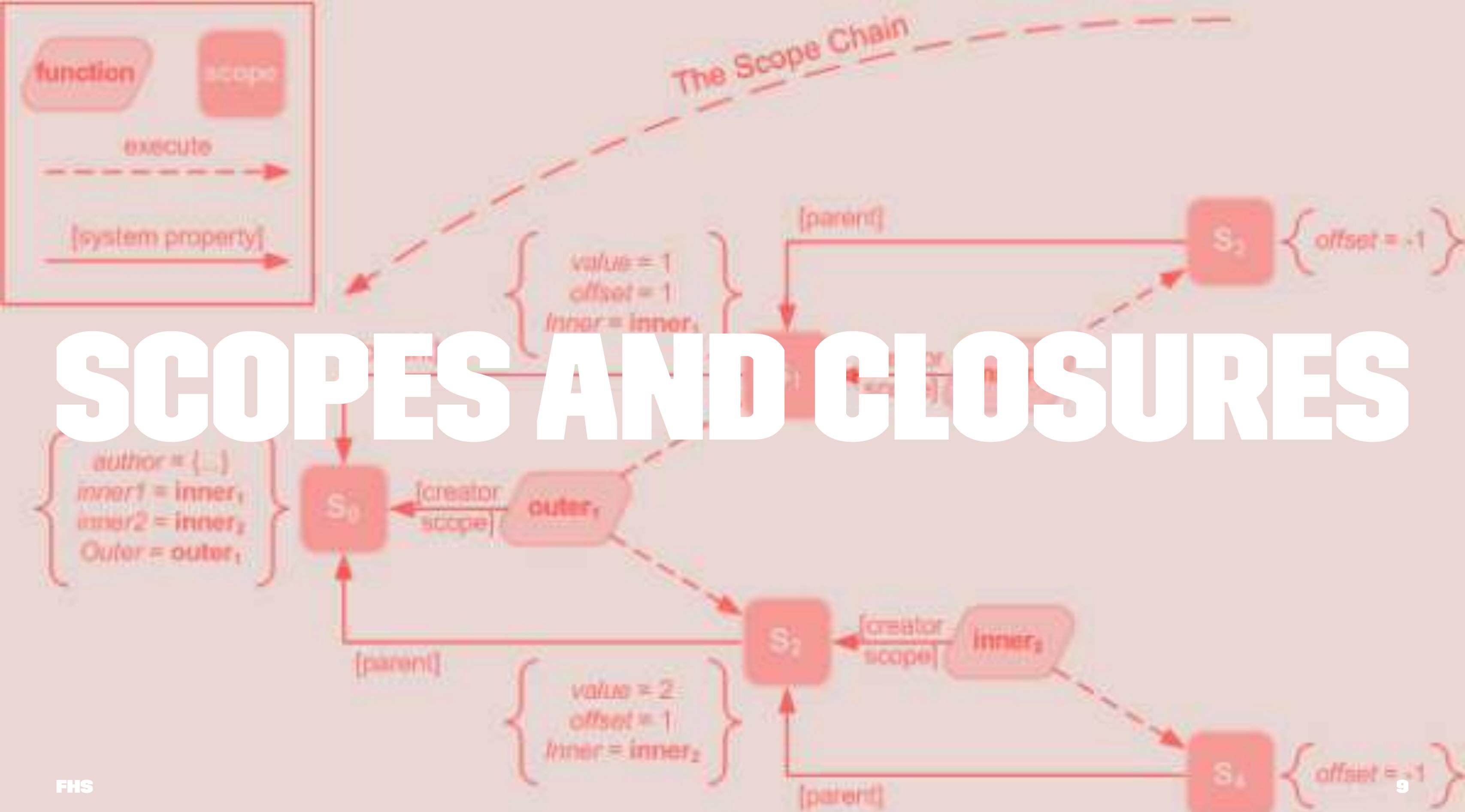
# THINGS I WILL LOOK AT

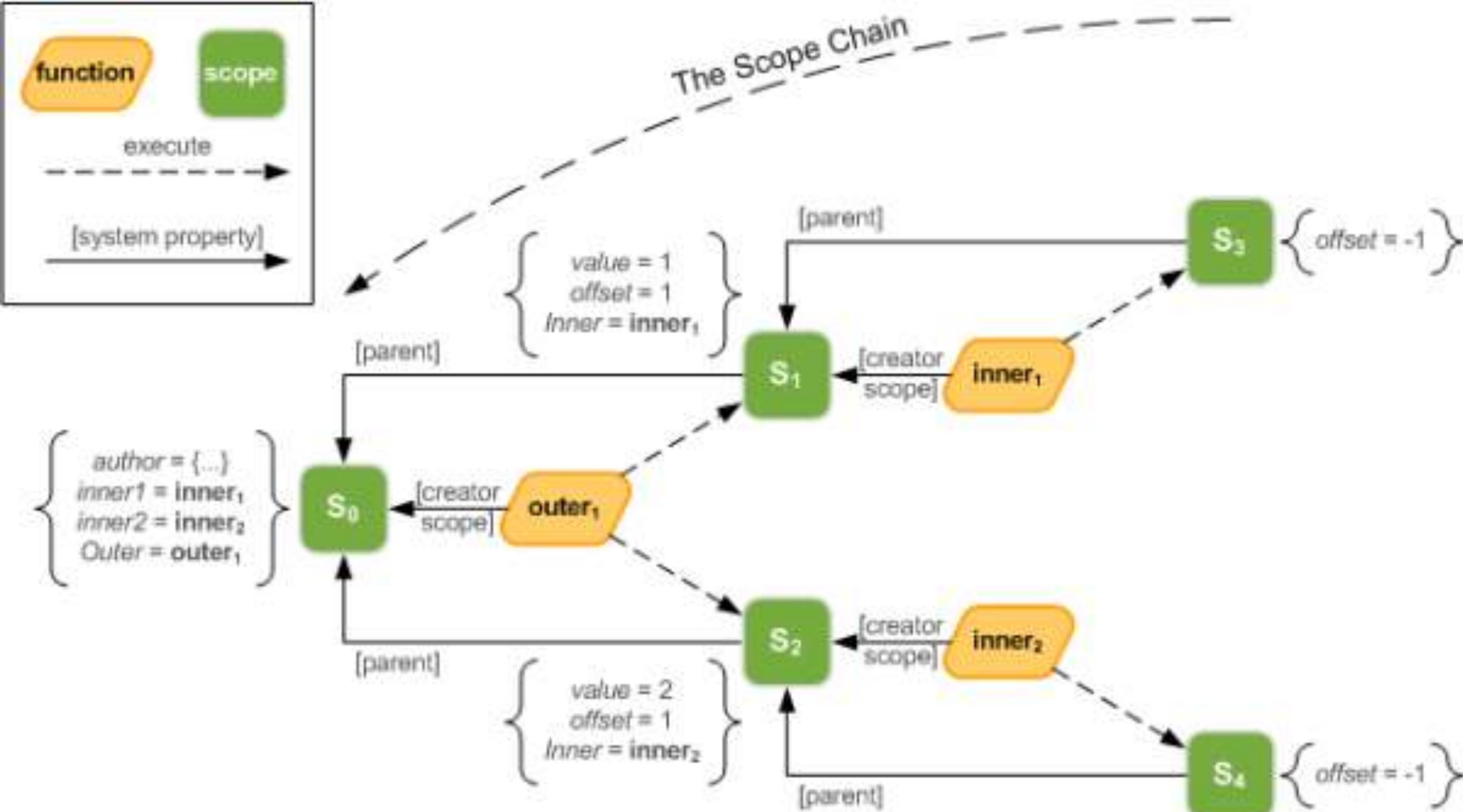
- » functionality
- » naming
- » duplications
- » code consistency
- » function/component length
- » commits + commit messages

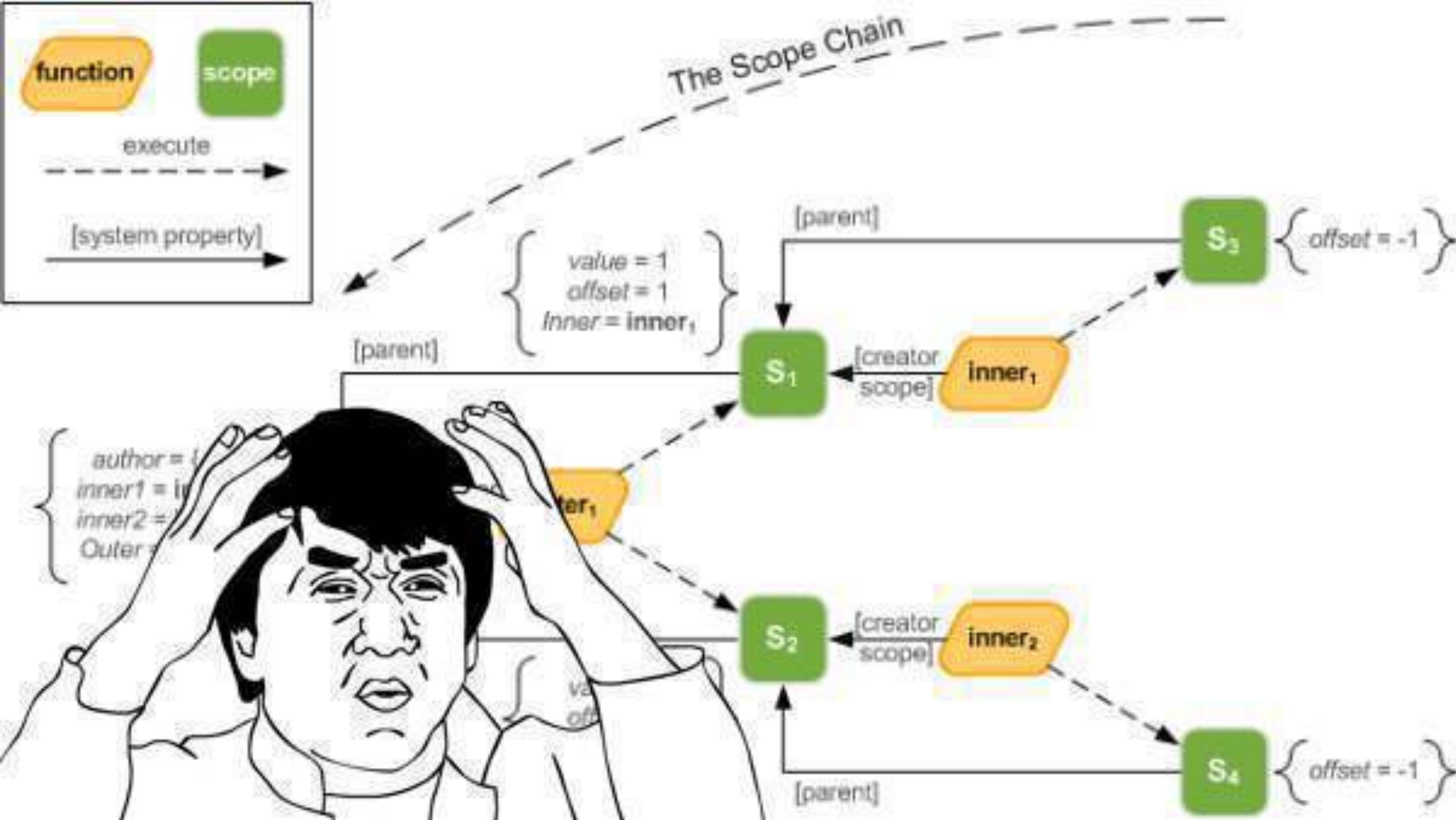
# FEEDBACK

- » Questions:  
[tmayrhofer.lba@fh-salzburg.ac.at](mailto:tmayrhofer.lba@fh-salzburg.ac.at)
- » Feedback Link









**SCOPE DETERMINES THE ACCESSIBILITY (VISIBILITY) OF VARIABLES.**

# SCOPES AND CLOSURES

In JavaScript there are 3 types of scope:

- » Global scope
- » Local scope or function scoped
- » Block Scope

Define the visibility of a variable within a program.

# LOCAL SCOPE

Variables declared within a JavaScript function, become LOCAL to the function.

```
function myFunction() {  
    var university = "FHS";  
  
    // code here CAN use university  
}
```

```
// code here CAN NOT use university
```

- » Created when function execution starts
- » Deleted when function execution completes

# BLOCK SCOPE (BEFORE ES6)

- » A block statement is used to group zero or more statements.
- » Used in loops, if/else statements
- » Everything between curly brackets {}
- » Before es6 all variables were function scoped

# BLOCK SCOPE (BEFORE ES6)

```
function myFunction() {  
  if (true) {  
    var university = 'FHS'  
    // code here CAN use university  
  }  
  // code here CAN use university as well  
}
```

# WHAT IS THE BEST UNIVERSITY?

```
function findBestUniversity() {  
    var bestUniversity = 'FHS'  
    if (true) {  
        var bestUniversity = 'Hagenberg'  
    }  
    console.log(bestUniversity)  
}
```

# WHAT IS THE BEST UNIVERSITY?

```
function findBestUniversity() {  
  var bestUniversity = 'FHS'  
  if (true) {  
    var bestUniversity = 'Hagenberg'  
  }  
  console.log(bestUniversity)  
  // Hagenberg will be logged 🤢  
}
```

# ES6 STORE RESCUE

# BLOCK SCOPE WITH ES6

- » let and const introduced as new keywords
- » define variables in a block scope

```
function myFunction() {  
  let university = 'FHS'  
  if (true) {  
    let university = 'Hagenberg'  
    // university is set to Hagenberg  
  }  
  console.log(university)  
  // FHS will be logged   
}
```

# WHAT WILL BE THE LOGGED?

```
function myFunction() {  
    let university1 = 'FHS1'  
    var university2 = 'FHS2'  
    let university3 = 'FHS3'  
  
    if (true) {  
        let university1 = 'FHS1 overwritten'  
        var university2 = 'FHS2 overwritten'  
        university3 = 'FHS3 overwritten'  
    }  
  
    console.log(university1)  
    console.log(university2)  
    console.log(university3)  
}
```

# WHAT WILL BE THE RESULT?

```
function myFunction() {  
    let university1 = 'FHS1'  
    var university2 = 'FHS2'  
    let university3 = 'FHS3'  
  
    if (true) {  
        let university1 = 'FHS1 overwritten'  
        var university2 = 'FHS2 overwritten'  
        university3 = 'FHS3 overwritten'  
    }  
  
    console.log(university1) // FHS1  
    console.log(university2) // FHS2 overwritten  
    console.log(university3) // FHS3 overwritten  
}
```

# let vs const

- » variables declared with let can be reassigned
- » variables declared with const can NOT be reassigned

```
let mutable = "some value"  
mutable = "some updated value"  
console.log(someValue) // "some updated value"
```

```
const immutable = "some value"  
immutable = "some updated value" // Uncaught TypeError: Assignment to constant variable.
```

# IMMUTABILITY<sup>5</sup>

“An immutable data structure is an object that doesn't allow us to change its value.  
(Remo H. Jansen)”

<sup>5</sup> [https://exploringjs.com/es6/ch\\_modules.html#sec\\_importing-exporting-details](https://exploringjs.com/es6/ch_modules.html#sec_importing-exporting-details)

# CONST REALLY IMMUTABLE

- » const variables cannot be reassigned
- » the objects itself can change

```
const immutable = { some: "value" }
immutable.some = "updated"
console.log(immutable.some) // "some updated value"
```

# IMMUTABLE OBJECTS IN JS

```
const immutableObject = Object.freeze({ test: 1 })
immutableObject.test = 10
console.log(immutableObject) // => { test: 1 }
```

# GLOBAL SCOPE

A variable declared outside a function, becomes **GLOBAL**.

```
var university = "FHS";  
  
function myFunction() {  
    // code here CAN use university  
}  
  
// code here CAN use university as well
```

All scripts and functions can access this variable.

# GLOBAL SCOPE

- » window is the global scope in the browser
- » global is the global object in nodejs

```
var university = "FHS";
window.university // "FHS"
```

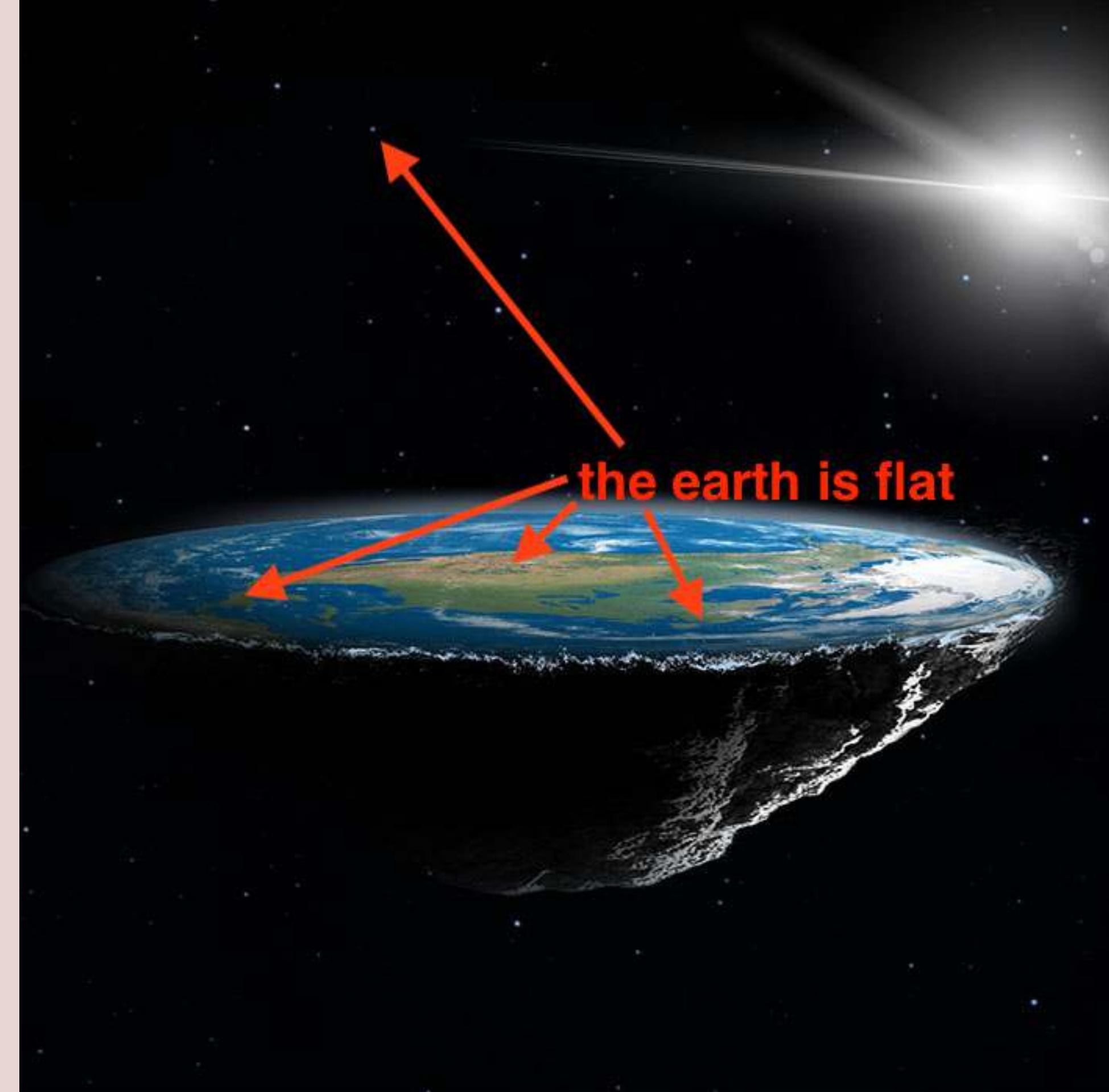


# GLOBAL NAMESPACE POLLUTION







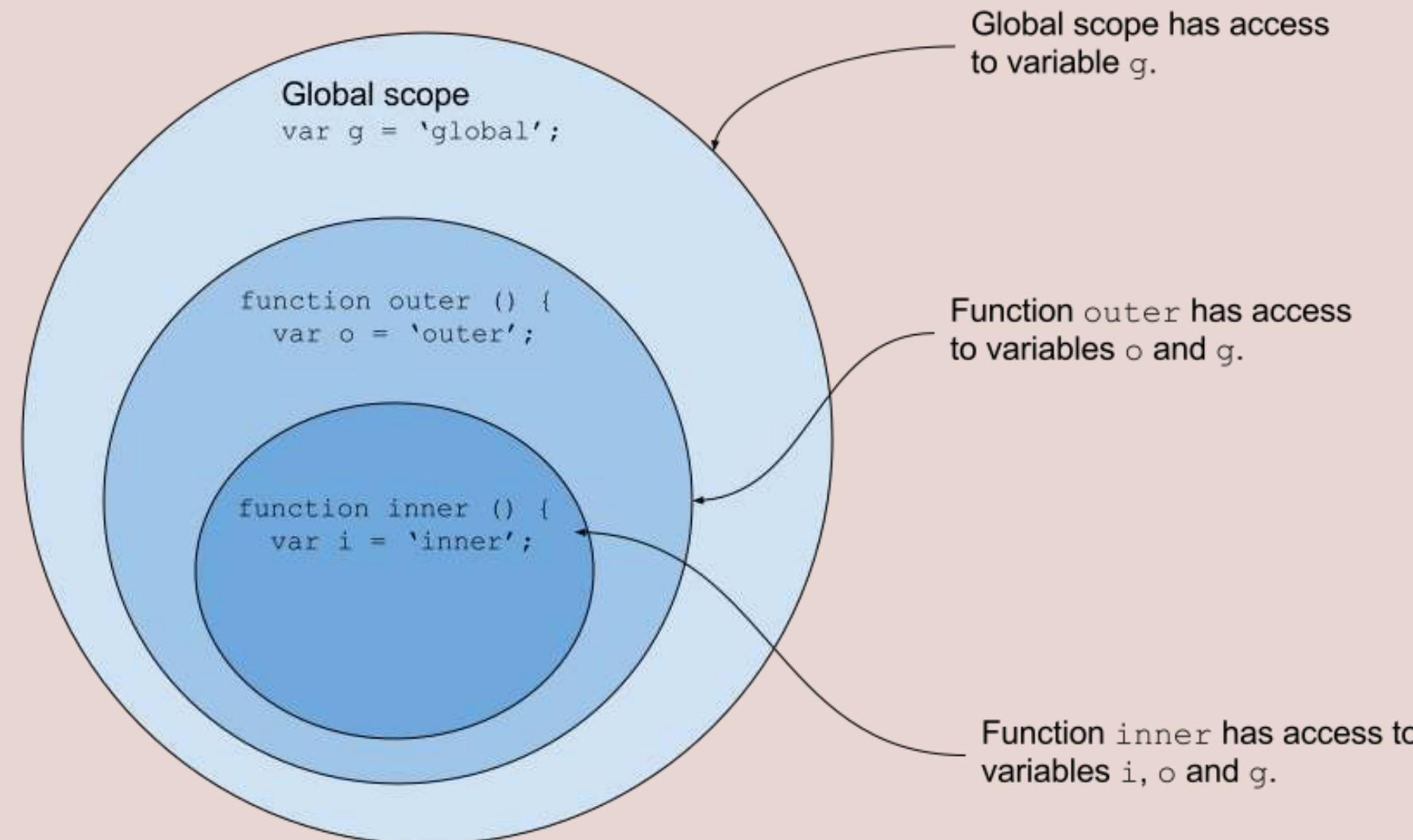


# CLOSURES

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment)

```
const globalScope = `Global`  
  
function outerFunction () {  
  const outer = `I'm the outer function!`  
  
  function innerFunction() {  
    const inner = `I'm the inner function!`  
    console.log(outer) // I'm the outer function!  
  }  
  
  console.log(inner) // Error, inner is not defined  
}
```

# CLOSURES



# ENCAPSULATING STATE IN CLOSURES

```
function counter(name) {  
  let i = 0;  
  return function () {  
    i++;  
    console.log(name + ' ; ' + i);  
  }  
}  
  
const counter1 = counter('Counter1')  
counter1(); // Counter1: 1  
counter1(); // Counter1: 2  
const counter2 = counter('Counter2')  
counter2(); // Counter2: 1  
counter2(); // Counter2: 2
```



# EXERCISE



- » go to (jsfiddle)[<https://jsfiddle.net/>] or open dev tools
- » implement a calculator using a closure (see previous slide)
- » calculator should support adding and subtracting numbers



# EXERCISE USAGE

```
const calculator1 = calculator()  
calculator1(10) // logs: 10  
calculator1(-1) // logs: 9
```

```
const calculator1 = calculator()  
calculator1(5) // logs: 5
```

nick



**POSSIBLE SOLUTION WILL BE ADDED AFTER THE LECTURE**

# JAVASCRIPT MODULES

# JAVASCRIPT MODULES

- » A way to split applications into smaller pieces
- » Similar to chapters/paragraphs in a book

# WHY MODULES<sup>1</sup>

- » Maintainability
- » Namespacing
- » Reusability

<sup>1</sup> <https://www.freecodecamp.org/news/javascript-modules-a-beginner-s-guide-783f7d7a5fcc/>

# MODULE PATTERN MIMIC OBJECTS WITH PRIVATE VARIABLES

# ANONYMOUS CLOSURE

```
(function () {  
  var someVariable = 'irrelevant'  
  
  console.log(someVariable);  
}());  
// `irrelevant` will be logged
```

# ANONYMOUS CLOSURE

```
(function () { // create a new function
  var someVariable = 'irrelevant' // function

  console.log(someVariable);
}()); // immediately execute the newly created function
```

# ANONYMOUS CLOSURE

- » Hides variables from global scope
- » global variables can't be overwritten
  - » var defined in new function scope
- » no global namespace pollution



# GLOBAL IMPORT

```
var myGlobalModule = {}  
  
(function (myModule) {  
  myModule.value = 'some value'  
}(myGlobalModule)); // "import" myGlobalModule into module  
  
console.log(myGlobalModule.value) // 'some value'
```

# GLOBAL IMPORT

- » myGlobalModule is only global variable
- » myGlobalModule is explicitly passed to module
- » values will be assigned to myGlobalModule

# OBJECT INTERFACE

```
const myCalculator = (function () {
  let value = 0
  return {
    increment: function() {
      value += 1
    },
    getValue: function() {
      return value
    }
  }
})();

myCalculator.increment()
console.log(myCalculator.getValue()) // 1
myCalculator.increment()
console.log(myCalculator.getValue()) // 2
```

# COMMONJS AND AMD

# COMMONJS AND AMD

- » Previous approaches encapsulate internals
- » Make Applications Modular
- » Define boundaries for functionality
  - » Similar to chapters in books
- » Scripts need to be loaded in correct order



# COMMONJS AND AMD ORDER OF SCRIPTS

- » finding the right order is tough
- » eg: backbone requires underscore.js
  - » underscore.js needs to be loaded before backbone
- » complexity of finding right

# COMMONJS MODULE

- » reusable piece of JavaScript
- » used in node.js
- » each file is own module with own context
- » `module.exports` exposes contents of a modules
- » prevents global namespace

# COMMONJS MODULE

```
// myCalculator.js
let value = 0

module.exports = {
  increment: function() {
    value += 1
  },
  getValue: function() {
    return value
  }
}

// -----
// app.js
const myCalculator = require('./myCalculator')

myCalculator.increment()
console.log(myCalculator.getValue()) // 1
myCalculator.increment()
console.log(myCalculator.getValue()) // 2
```

# AMD MODULE

- » Asynchronous Module Definition
- » CommonJS loads all modules synchronous
- » Browser blocks other JS execution until everything is loaded
- » modules can be loaded when needed
- » used in browsers which don't support es6 modules

```
define([
  'myModule', 'myOtherModule'], // define dependencies
  function(myModule, myOtherModule) { // callback will be executed once myModule/myOtherModule was loaded
    console.log(myModule.hello());
});
```

# UMD MODULE

- » Combines CommonJS und AMD modules
- » Sees which environment is available
  - » loads modules by either CommonJS or AMD
- » Work both on the server and on the client
- » When building a library use UMD as target

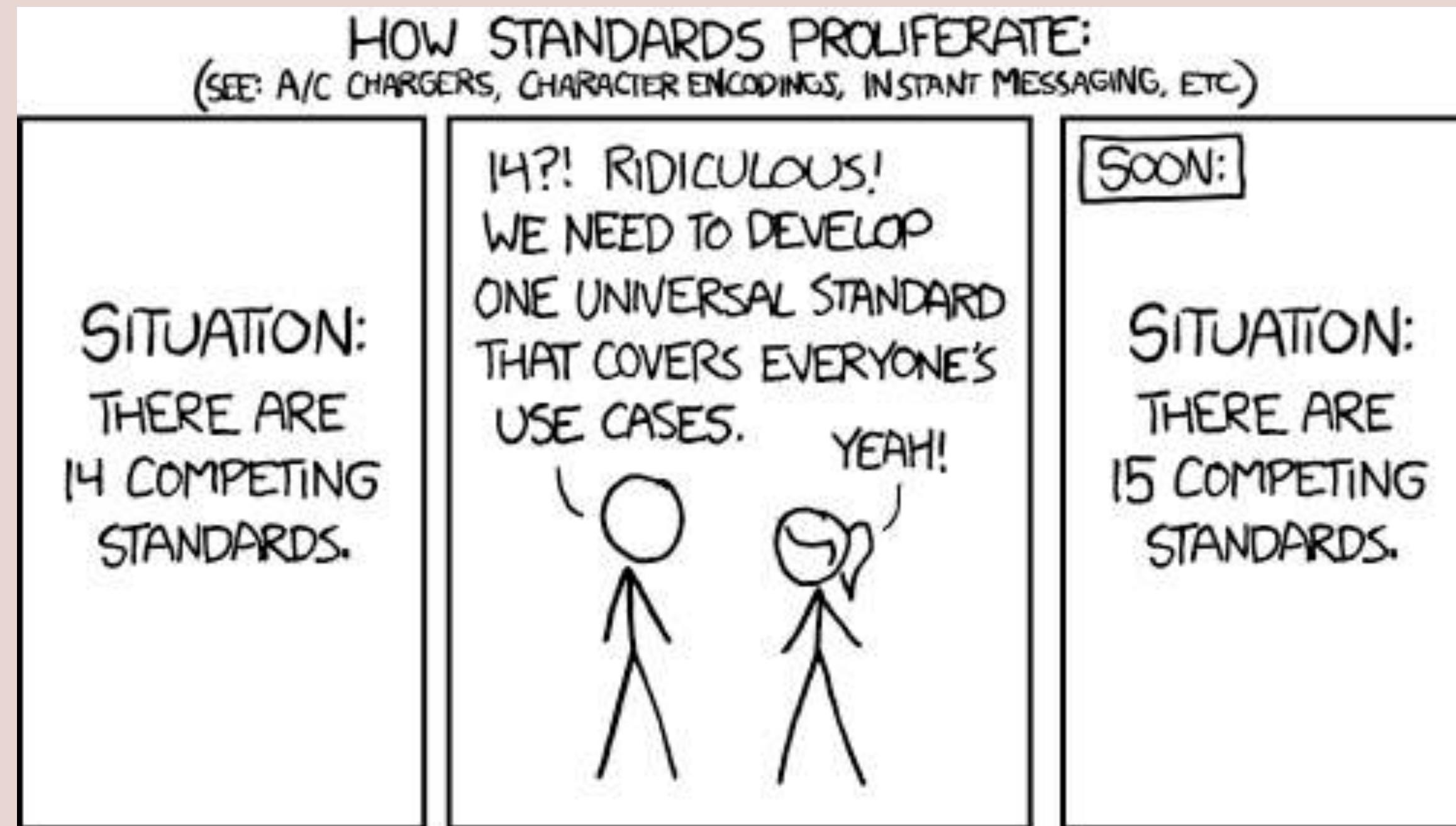
# UMD MODULE<sup>2</sup>

```
(function (root, factory) {
  if (typeof define === 'function' && define.amd) {
    // AMD
    define(['myModule', 'myOtherModule'], factory);
  } else if (typeof exports === 'object') {
    // CommonJS
    module.exports = factory(require('myModule'), require('myOtherModule'));
  } else {
    // Browser globals (Note: root is window)
    root.returnExports = factory(root.myModule, root.myOtherModule);
  }
})(this, function (myModule, myOtherModule) {
  let value = 0

  return {
    increment: function() {
      value += 1
    },
    getValue: function() {
      return value
    }
  }
}));
```

<sup>2</sup> Writing a UMD module won't be part of the exam.

# NATIVE JS MODULES



# ES6 MODULES

- » CommonJS and UMD are not standardized by ECMA
  - » both standards emulate module pattern via JS
- » Built in modules with ES6 <sup>4</sup>
  - » compact and declarative syntax
  - » asynchronous loading
  - » server and browser

<sup>4</sup> source: <https://medium.freecodecamp.org/javascript-modules-a-beginner-s-guide-783f7d7a5fcc>

# ES6 MODULES<sup>3</sup>

The compatibility matrix illustrates the support for three key features of ES6 Modules across different environments:

- `export`: Available in Chrome 61, Edge 16, Firefox 60, Opera 48, Safari 10.1, and Node.js 13.2.0. Edge and Firefox have a note below them.
- `default keyword with export`: Available in Chrome 61, Edge 16, Firefox 60, Opera 48, Safari 10.1, and Node.js 13.2.0. Edge and Firefox have a note below them.
- `export * as namespace`: Available in Chrome 72, Edge 79, Firefox 80, Opera 60, Safari 11.0, and Node.js 12.0.0. Edge and Firefox have a note below them.

Icons represent the platforms: Desktop (monitor), Mobile (phone), and Server (server tower). A note at the top right says "Update compatibility data on GitHub".

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Android webview	Chrome for Android	Firefox for Android	Opera for Android	Safari on iOS	Samsung Internet	Node.js
<code>export</code>	61	16	60	No	48	10.1	No	61	60	45	10.3	8.0	13.2.0
<code>default keyword with export</code>	61	16	60	No	48	10.1	No	61	60	45	10.3	8.0	13.2.0
<code>export * as namespace</code>	72	79	80	No	60	No	No	72	No	51	No	11.0	12.0.0

<sup>3</sup> <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules> 01.11.2020

# ES6 MODULES NAMED EXPORTS

```
// counter.js
let value = 0

export function increment () {
  value += 1
}
export function getValue () {
  return value
}

// app.js
import { increment, getValue } from './counter.js'

increment()
console.log(getValue()) // 1
increment()
console.log(getValue()) // 2
```

# ES6 MODULES NAMED EXPORTS, IMPORT ENTIRE MODULE

```
// counter.js
// ...
```

```
// app.js
import * as myCalculator from './counter.js'

myCalculator.increment()
console.log(myCalculator.getValue()) // 1
myCalculator.increment()
console.log(myCalculator.getValue()) // 2
```

# ES6 MODULES

## DEFAULT EXPORT

```
// myModule.js
export default function myAmazingJSFunction () {
  console.log('FHS')
}

// app.js

import myAmazingJSFunction from './myModule.js'
import nameCanBeAnythingForDefaultExports from './myModule.js'

myAmazingJSFunction() // will log "FHS"
nameCanBeAnythingForDefaultExports() // will log "FHS"
```

# ES6 MODULES

## DYNAMIC IMPORT

- » import statement must be top level
- » conditional loading of scripts is not possible

```
if (Math.random()) {  
    import 'foo'; // SyntaxError  
}
```

# ES6 MODULES

## DYNAMIC IMPORT

- » dynamic import to rescue
- » adds ability to load scripts
  - » conditionally
  - » on demand
- » harder to analyze statically

# ES6 MODULES

## DYNAMIC IMPORT

```
if (loadCounter) {
  // counter won't be loaded when loadCounter === false
  const { increment, getValue } = await import('./counter.js')
  increment()
  console.log(getValue()) // 1
  increment()
  console.log(getValue()) // 2
}
```

# ES6 MODULES

## IMPORTING STYLES

```
// import a default export
import someModule from './my-module.js'

// import an entire module (excluding the default export)
import * as namedExports from './my-module.js'

// import an entire module (including the default export)
import defaultExport, * as namedExports from './my-module.js'

// import a named export
import { export1, export2 } from './my-module.js'

// alias a named export
import { export1 as aliasedExport1 } from './my-module.js'

// dynamic import
const { default, export1, export2 } = await import('./my-module.js')
```

# ES6 MODULES

## EXPORT<sup>5</sup>

```
export let myVar2 = '...';
export const MY_CONST = '...';

export function myFunc() {
    //...
}

export function* myGeneratorFunc() {
    //...
}

export class MyClass {
    //...
}
```

<sup>5</sup> [https://exploringjs.com/es6/ch\\_modules.html#sec-importing-exporting-details](https://exploringjs.com/es6/ch_modules.html#sec-importing-exporting-details)

# ES6 MODULES

## EXPORT<sup>5</sup>

```
const MY_CONST = 'MY_CONST'  
function myFunc() {}
```

```
// Export all at the end  
export { MY_CONST, myFunc };
```

```
// Export with a different name  
export { MY_CONST as FOO, myFunc as myAliasedFunc };
```

<sup>5</sup> [https://exploringjs.com/es6/ch\\_modules.html#sec-importing-exporting-details](https://exploringjs.com/es6/ch_modules.html#sec-importing-exporting-details)

# USING ES6 MODULES IN THE BROWSER

```
<!-- index.html -->
<script src="./index.js" type="module"></script>
<!--
<!-- required so the browser knows you're using modules -->

import { myApp } from './appliction.js'
// import application from a different module
// note .js extension is required in the browser

myApp()
```



# HOMEWORK

- » Due Date: 10.11. 8pm
- » can be done in pairs
- » hand-in via email  
`tmayrhofer.lba@fh-salzburg.ac.at`
- » email contains link to  
git repository
- » name of students who  
worked on the assignment

# HOMEWORK

- » We'll be building a quiz application
- » This assignment includes the game logic only (no ui)
- » setup
  - » clone Repository
  - » if you're having troubles let me know

# HOMEWORK

- » define and export a list of questions from `questions.js`
- » a question looks like this: `{ question: 'some question', correctAnswer: 'a', a: 'answer', b: '', c: '', d: '' }`
- » implement a function `askQuestion()` in `quiz.js`
- » this function returns a random question (without the `correctAnswer` property)

# HOMEWORK EXAMPLE USAGE IN INDEX.JS

```
// index.js
import { askQuestion, answerQuestion } from './quiz.js'

const question = askQuestion()
console.log(question)

/**
 * {
 *   question: 'Whats the best university?',
 *   a: 'Hagenberg',
 *   b: 'FHS',
 *   c: 'TU',
 *   d: 'JKU'
 * }
 */

const answer = answerQuestion(question, 'a')
console.log(answer ? 'correct' : 'incorrect')
```

# FEEDBACK

- » Questions:  
[tmayrhofer.lba@fh-salzburg.ac.at](mailto:tmayrhofer.lba@fh-salzburg.ac.at)
- » Feedback Link

