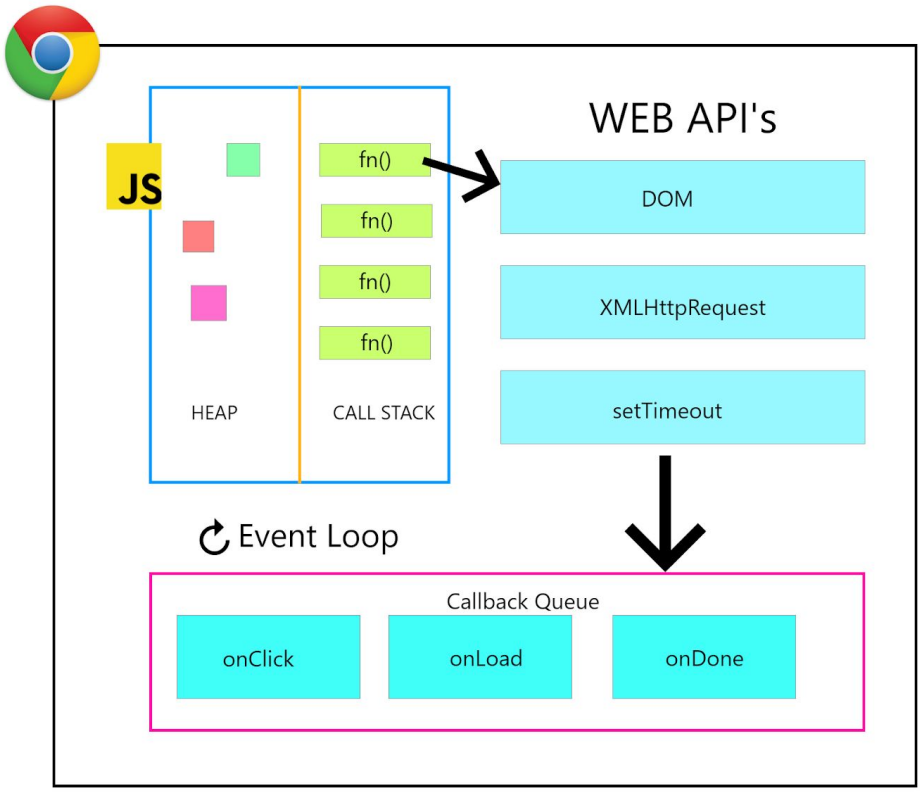
Matias Bue Koefoed og Emil Gotthelf Tranberg Hansen

Period-1 Vanilla JavaScript, es2015/15.., Node.js, Babel + Webpack and TypeScript

Explain and Reflect:

- Explain the differences between Java and JavaScript and Java and node. Topics you could include:
    - that Java is a compiled language and JavaScript a scripted language
    - Java is both a language and a platform (node and javascript combined (kind of))
    - Node is not a language like Java, it is only a runtime
    - Blocking vs. non-blocking (javaScript by ITSELF is single-threaded)
- Explain generally about node.js, when it “makes sense” and *npm*, and how it “fits” into the node echo system.(if you want to run JS outside a browser then you need node. Node makes sense to use any time you don’t work directly with scripts in the browser)(npm is comparable to maven, it enables building larger scaling projects)
- Explain about the Event Loop in JavaScript, including terms like: blocking, non-blocking, event loop, callback que and "other" API's. Make sure to include why this is relevant for us as

developers.



WebAPI are functions in JS provided by the browser or node.

Event loop checks if the JS stack is empty and then returns first callback in the callback queue

heap is the place where objects are stored when we define variables

The callstack is a list of function call in JS

The event loop video: https://www.youtube.com/watch?v=8aGhZQkoFbQ

- Explain the terms JavaScript Engine (name at least one) and JavaScript Runtime Environment (name at least two)(V8 in chrome, Chakra in old edge or spidermonkey in firefox)(JSRE are browsers or node)
- Explain (some) of the purposes with the tools *Babel* and *WebPack and how they differ from each other*. Use examples from the exercises.

Explain using sufficient code examples the following features in JavaScript (and node)

Matias Bue Koefoed og Emil Gotthelf Tranberg Hansen

- Variable/function-Hoisting
  - Variables and functions are declared initially at runtime but not initialized
- `this` in JavaScript and how it differs from what we know from Java/.net.
  - this in JS can refer to both objects and functions. this then refers to the function closure scope.
- Function Closures and the JavaScript Module Pattern
  - A closure is a feature in JavaScript where an inner function has access to the outer (enclosing) function's variables
- User-defined Callback Functions (writing your own functions that take a callback)
  - A function can be passed as an argument to another function
- Explain the methods `map`, `filter` and `reduce`
  - map example, +1 to all elements in a numbers array
  - filter example, find all even numbers
  - reduce example, return the sum of the numbers
- Provide examples of user-defined reusable modules implemented in Node.js (learnyounode - 6)
- Provide examples and explain the es2015 features: `let, arrow functions, this, rest parameters, destructuring objects and arrays, maps/sets` etc.
  - let: The let statement declares a block scope local variable, optionally initializing it to a value.
  - arrow functions:An arrow function expression is a syntactically compact alternative to a regular function expression, although without its own bindings to the this....
  - this: It has different values depending on where it is used:
    - In a method, this refers to the owner object.
    - Alone, this refers to the global object.
    - In a function, this refers to the global object.
    - In a function, in strict mode, this is undefined.
    - In an event, this refers to the element that received the event.
    - Methods like call(), and apply() can refer this to any object.
  - rest parameters: function f(...[a,b,c]) return a + b + c
  - destructuring: const {func1, func2} = import(funcs)
  - sets: set finds unique value combinations
    - let mySet = new Set()
    - mySet.add(1) // Set [1]
    - mySet.add(5) // Set [1, 5]
    - mySet.add(5) // Set [1, 5]
  - maps: work like in java with [key, value]
- Provide an example of ES6 inheritance and reflect over the differences between Inheritance in Java and in ES6.
- Explain and demonstrate, how to implement your own events, how to emit events and how to listen for such events
  - `myEmitter.on('DOS', function listener (data) {`
  - `  console.log(data);`
  - `});`

- myEmitter.emit('DOS', {url,timeBetweenCalls: deltaTime});

ES6,7,8,ES-next and TypeScript

- Provide examples with es-next, running in a browser, using Babel and Webpack
- Explain the two strategies for improving JavaScript: Babel and ES6 (es2015) + ES-Next, versus Typescript. What does it require to use these technologies: In our backend with Node and in (many different) Browsers
- Provide a **number of examples** to demonstrate the benefits of using TypeScript, including, types, interfaces, classes and generics

  TS video: https://www.youtube.com/watch?v=0ChtcZmb3dI

- Explain the ECMAScript Proposal Process for how new features are added to the language (the TC39 Process)
  - Stage 0: Strawperson — to allow input into specifications
  - Stage 1: Proposal — make the case for the addition, describe the solution and identify the potential challenges
  - Stage 2: Draft — describe the syntax and semantics using formal spec language
  - Stage 3: Candidate — states that further refinement will need feedback from implementations and users
  - Stage 4: Finished — states that the addition is ready for inclusion in the formal ECMAScript standard

Callbacks, Promises and async/await

Explain about (ES-6) promises in JavaScript including, the problems they solve, a quick explanation of the Promise API and:

- Example(s) that demonstrate how to avoid the callback hell (“Pyramid of Doom")
- Example(s) that demonstrate how to execute asynchronous (promise-based) code in **serial** or **parallel**
- Example(s) that demonstrate how to implement **our own** promise-solutions.
- Example(s) that demonstrate error handling with promises

Explain about JavaScripts **async/await**, how it relates to promises and reasons to use it compared to the plain promise API.

Provide examples to demonstrate

- Why this often is the preferred way of handling promises
- Error handling with async/await
- Serial or parallel execution with async/await.

Se the exercises for Period-1 to get inspiration for relevant code examples