- ***Explain the differences between Java and JavaScript + node. Topics you could include:***
    - ***that Java is a compiled language and JavaScript a scripted language***

    Java code needs to be compiled into machine code to run in the java engine, while javascript can run directly in a javascript engine.

    - ***General differences in language features***

    Java code must be compiled, and JavaScript code is all-text.
    Each language requires different plug-ins.
    JavaScript code is run on a browser only(if used vanilla, without tools like Node), while Java creates applications that run in a virtual machine or browser.
    Java is an OOP (object-oriented programming) language, and JavaScript is specifically an OOP scripting language.

    - ***Blocking vs. non-blocking***

    Java is a multi-threaded programming language, allowing for multiple processes to run at the same time, while javascript is single-treaded. The difference is, that it is in javascript very possible to block the flow, by having a bit of code taking a long time or wait for external factors, and not allowing the next code to run, unless the use of promises/async methods are used.

    **Variables**: In javascript, variables are typeless, meaning that a variable can both is declared as var/let, and can hold both Boolean values, strings, numbers, ext. Types like string and number do exist in javascript, but only as values, and not as variables, meaning that it can be both freeing and difficult to work with variables and values in javascript.

    **Main class:** In javascript, you can run any .js file on it's own, and if it requires other .js files, it can import/require them itself, so while you will normally want to start a program from a specific file, you can run it each file separately, unlike in java.

- ***Explain generally about node.js, when it "makes sense" and npm, and how it "fits" into the node echo system.***

Node is a runtime environment used by javascript, in order to interact with the local system, for instance when we need additional files on the computer to be used, or when we want the program to write to another file. The npm is the package manager, responsible setting up and keeping track of the node environment as it is install into the system.

- ***Explain about the Event Loop in JavaScript, including terms like; blocking, non-blocking, event loop, callback queue and "other" API's. Make sure to include why this is relevant for us as developers.***

The Event loop consist of, as a minimum example, the stack, a webapi, and the callback queue. When a function is called with async or setTimer for instance, it is given to the webapi to keep track of, and when it's done/the time runs out, it is placed upon the callback queue, where it will go in the back of the line. Whenever the stack is empty, it will take the first object in the callback queue, and put onto the stack. As such, when code runs on the stack, it is blocking the stack, when it is handled outside the stack, and put into the event loop, it becomes non-blocking. This allows for faster code, since we don't have to wait for potential slow blocking code, that depends on outside assistance, like fetching from an external api. This can also be used to make sure that code runs after the stack has been emptied from what else was on the stack at the time.

- ***What does it mean if a method in nodes API's ends with xxxxxxSync?***

It would mean that it is a function that do NOT use a async method, and would as such be a blocking method, blocking the JS-event-queue (the stack)

- ***Explain the terms JavaScript Engine (name at least one) and JavaScript Runtime Environment (name at least two)***

A Javascript engine (v8 for chrome, spiderMonkey for firefox) is the program parsing the javascript code, and the part that runs the code from the top and down, and look for syntax errors, turning the js-code into machine code.
A javascript runtime enviorment (node, chrome, firefox) is what provides some objects to js to interact with, so it can interact with objects outside it's own imidiet scope. This also includes the outside elemtents like the webapi and callback queue.

- ***Explain (some) of the purposes with the tools Babel and WebPack and how they differ from each other. Use examples from the exercises.***

Webpack is a module that help organize and compile js code from the core version andimports from other modules. Babel can be used to compile modern js-code into earlier versions, so it can be used in runtime-environments that might not use the newest version of js. You can target the versions of js that should be compiled to.

Example for WebPack is in Week4/Webpack-demo, and for Bable is in Week4/Babel, for instance the .babelrc file.

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

- ***Variable/function-Hoisting***

Code Examples under Week01/magicOfCallbacks/hoisting.js

- ***`this` in JavaScript and how it differs from what we know from Java/.net.***

Code Examples under Week01/magicOfCallbacks/thisInJavaScript.js

- ***Function Closures and the JavaScript Module Pattern***

Code Examples under Week01/magicOfCallbacks/closures.js

- ***User-defined Callback Functions (writing functions that take a callback)***

Code Examples under Week01/magicOfCallbacks/myPrototype.js

- ***Explain the methods `map`, `filter` and `reduce`***

Code Examples under Week01/magicOfCallbacks/myFilter.js, myMap.js & reduceMethod.js

- ***Provide examples of user-defined reusable modules implemented in Node.js (learnynode - 6)***

Code Examples under Week01/learnyounode/make-it-module.js & mymodule.js

- ***Provide examples and explain the es2015 features: `let, arrow functions, this, rest parameters, destructuring objects and arrays, maps/sets` etc.***

Code Examples under OtherExamples/es2015_features

- 🟨 ***Provide an example of ES6 inheritance and reflect over the differences between Inheritance in Java and in ES6.***

Code Examples under otherExamples/inheritance.js.
*Java follows class based inheritance—a top down, hierarchical, class-based relationship whereby properties are defined in a class and inherited by an instance of that class (one of its members). In JavaScript, inheritance is prototypal—all objects can inherit directly from other objects. Hierarchy is accomplished in JavaScript by assigning an object as a prototype with a constructor function.*
***Need to study closer, so far taken from:*** *[https://www.upwork.com/resources/java-vs-javascript-what-is-the-difference](https://www.upwork.com/resources/java-vs-javascript-what-is-the-difference)*

- ***Explain and demonstrate, how to implement event-based code, how to emit events and how to listen for such events***

Code Examples under Week02/assignment2.js & dosDetector.js

ES6,7,8,ES-next and TypeScript

- ***Provide examples with es-next, running in a browser, using Babel and Webpack***

Look to Week04/babel, focus in the .babelrc and package.json files, and point out how the target can be changed to es-next. Take inspiration from here [https://www.youtube.com/watch?v=C2PDAGCrk_g](https://www.youtube.com/watch?v=C2PDAGCrk_g)

- ***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***

The strategies for these are to be able to write newer/smarter/better js code, that can take from the newest versions of js, and is then able to compile it down into any earlier version of javascript. This allows a programmer to focus on writing in one version of js, and should have it work in most/all versions after compiling. Both methods require the installation of the relevant modules, to be writing in their own file-type, and then to be compiled into javascript afterwards.

- ***Provide examples to demonstrate the benefits of using TypeScript, including, types, interfaces, classes and generics***

Look to Week05/exercises/exe1/interfaces.js

- 🟨 ***Explain the ECMAScript Proposal Process for how new features are added to the language (the TC39 Process)***

The process have 4/5 stages, and is meant to develop the newest version of javascript each year with the most relevant and thoughtly build tools available. Anyone can apply to this process, but each stage is designed to ensure that only relevant updates are considered. Each stage has conditions to be able to move the proposal to the next stage.
**Stage 0:** Strawman. Anyone can make this, as it is when the proposal is in a sketch format. To move forward, the proposal needs champions/patrons that recommend it for consideration.
**Stage 1:** Proposal. This is presented as a proposal, and now needs working implementations to demonstrate to the comity.
**Stage 2:** Draft. This will likely be included at some point. It needs to have all it's specs ready to be a candidate.
**Stage 3:** Candidate. This is a 'finished' draft, that now require to have tests written and passed to ensure it's quality, before moving on to the final stage.
**Stage 4:** Finished. This will be included in the next version of js.

**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")***
  Look in Week3/Exercises/Exercise2.js

- ***Example(s) that demonstrate how to execute asynchronous (promise-based) code in serial or parallel***
  Look in Week3/Exercises/Exercise3.js

- ***Example(s) that demonstrate how to implement our own promise-solutions.***
  Look in Week3/Exercises/Exercise1.js, assignement B. The focus is on newPromise(resolve,reject)

- ***Example(s) that demonstrate error handling with promises***
  Use Week3/Exercises/error-handling-examples.js

***Explain about JavaScripts async/await, how it relates to promises and reasons to use it compared to the plain promise API.***
Async/Await is ‘just’ promises used thought a nicer syntax, but serv the same functionality. An async method returns a promise, and the part of the method using await is what the promise is waiting for. Anything using one of the variables that had await Infront of it is then treated as if it was part of a .then method following that await code.

Provide examples to demonstrate

- ***Why this often is the preferred way of handling promises***
  Look in Week3/Exercises/Exercise2.js

- ***Error handling with async/await***
  Use Week3/Exercises/ error-handling-examples.js

- 🟨 ***Serial or parallel execution with async/await.***
  Look in Week3/Exercises/Exercise3.js