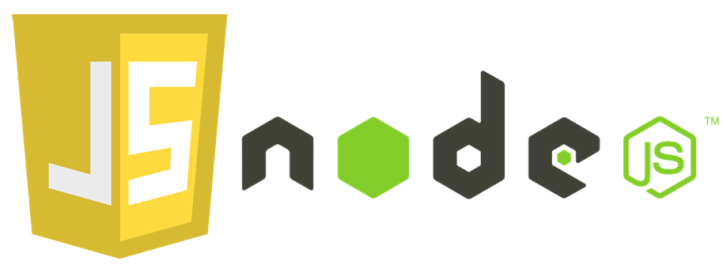
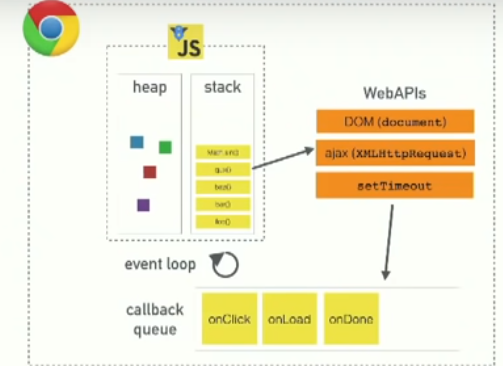
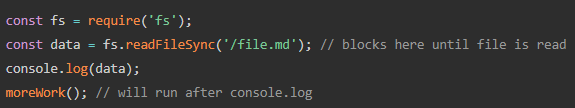
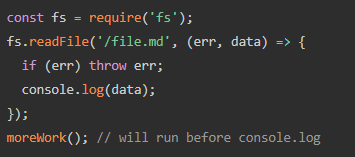
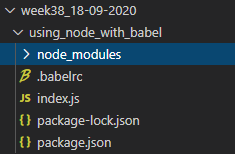
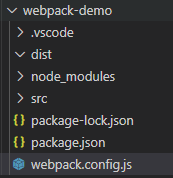
**Period-1 Vanilla JavaScript, Es-next, Node.js, Babel + Webpack and TypeScript**

Note: This description is too big for a single exam-question. It will be divided up into several smaller questions for the exam

Explain and Reflect:

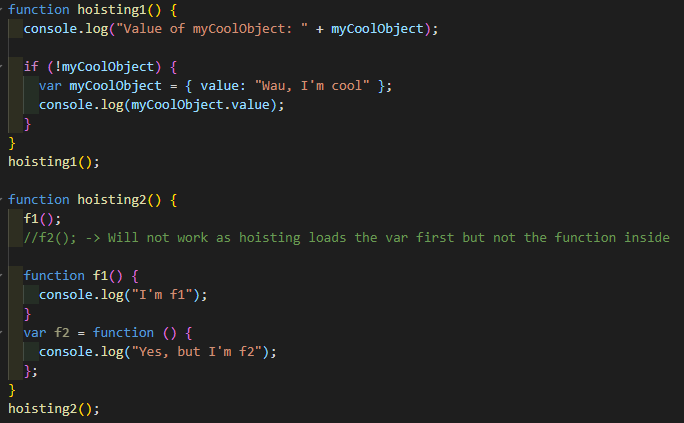
* Explain the differences between Java and JavaScript + node. Topics you could include:
  + that Java is a compiled language and JavaScript is a scripted language
  + Java is both a language and a platform.
  + Java is an Object-Oriented, general purpose programming language and class-based
  + General differences in language features.
  + Node JS is a runtime library and environment which is cross-platform and used for creating running JavaScript applications outside the browser. It is a free and open source and utilized for creating server-side JS applications. Node JS allows developers to execute their code on the server-side. It provides a faster way to write scripts that are scalable and light.
  + Node JS gives JavaScript the ability to interact with I/O (input/output) devices through its APIs, and connect with other external libraries written in various other languages.
  + Java is suited for complex webbased concurrency project where Node JS is best suited for small size projects.
  + Java dominate server-side interaction where Node JS developers can utilize it on client and server side.
  + Source code written in Java is compiled into bytecode and then bytecode is converted to machine code, ready to run on any platform whereas, with Node JS, source code written in a source file is JavaScript, Node JS interpreter will be used to interpret and execute JavaScript code.
  + Java uses the concept of multithreading with ease whereas Node JS does not use the concept of multi-threading like Java does.
  + CPU intensive task required huge computational power can be handled with Java whereas I/O bound operations like real-time chat, media streaming etc can be handled well with Node JS.

* Explain generally about node.js, when it “makes sense” and *npm*, and how it “fits” into the node echo system.
  + Node.js is an open source project designed to help you write JavaScript programs that talk to networks, file systems or other I/O sources.
  + NPM (Node package manager) is a package manager for Node.js. It is one of the largest online repository where we can install functionalities by writing “npm install NAME” and it will be added into our node-module folder and we can then import it into our project. It is an fast and easy (usually) plug-and-play. Packages can be found on <https://www.npmjs.com/> where there is also documentation and tutorials with examples.
* 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.  
  <https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/>
  + Node.js is a single-threaded application. So, we have the Event Loop is a que for how code is executed. When we load a page we wouldn’t want to load the button of the screen first with a “slow loader” so that it pauses the interaction so to say. We need to think of ways to block the right things and use non-blocking for others. When we for example want to show a map of data, we want the data to be retrieved first before we show a table.  
    
  + Blocking methods execute synchronously.  
    
  + Non-blocking methods execute asynchronously.  
    
* What does it mean if a method in nodes API's ends with xxxxxx**Sync**?
  + When a node API ends with Sync, it means that it is blocking. It will run the code synchronously with everything else that is on the stack.
* Explain the terms JavaScript Engine (name at least one) and JavaScript Runtime Environment (name at least two)
  + A JavaScript engine is a computer program that executes JavaScript code.
  + “V8” is the name of an open-source Java-Script engine, developed by the chromium project that is also used in Google Chrome.
  + Mozilla Firefox uses “[SpiderMonkey](https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey)”, “[Gecko](https://developer.mozilla.org/en-US/docs/Mozilla/Gecko)” and “[Quantum](https://blog.mozilla.org/blog/2017/11/14/introducing-firefox-quantum/)”. Safari uses “[WebKit](https://webkit.org/)” as their JS engines.
* Explain (some) of the purposes with the tools *Babel* and *WebPack and how they differ from each other*. Use examples from the exercises.
  + [Webpack](https://medium.com/ag-grid/webpack-tutorial-understanding-how-it-works-f73dfa164f01#:~:text=Webpack%20is%20a%20module%20bundler,standard%20for%20Web%20Applications%20today.) is a module bundler. It takes disparate dependencies, creates modules for them and bundles the entire network up into manageable output files. This is especially useful for Single Page Applications (SPAs), which is the defacto standard for Web Applications today.
  + [Babel](https://babeljs.io/docs/en/) is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments. One of the few things Babel can do is: Transform syntax, polyfill features that are missing in the target environment (through @babel/polyfill), source code transformations etc.
  + Differences:  
    Babel:  
    Our file directory:  
      
    Possibly to use new features of JavaScript code.  
    We can choose how it would run different in dev/prod.  
    Useable to convert different versions of JavaScript versions to older browser or environments versions.

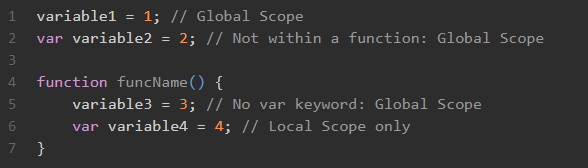
WebPack:   
Our file directory:  
  
We have a webpack.config file here where we can add what to export, our plugins, output name+directory and much more.

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

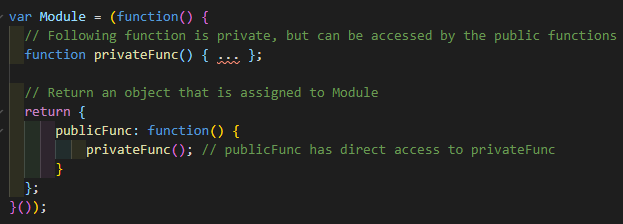
* Variable/function-Hoisting
  + Function declarations are completely hoisted, var declarations are also hoisted, but not assignments made with them.



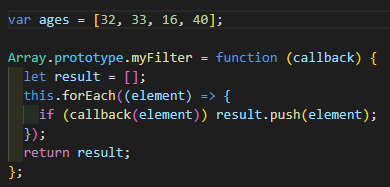
* *this* in JavaScript and how it differs from what we know from Java/.net.
  + In JavaScript this always refers to the “owner” of the function we’re executing, or rather to the object that a function is a method of.
  + In Java, this refers to the current instance object on which the method is executed.
* Function Closures and the JavaScript Module Pattern
  + Closures are a construct of the JavaScript language. Within JavaScript all variables are accessible from the global scope except variables that are declared within a function using the var keyword.



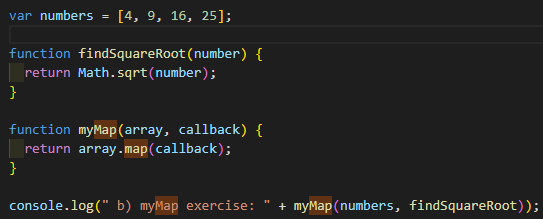
* + Knowing what we know about closures, we can create objects using the module pattern. By returning an object or variable and assigning it to a variable outside of the function, we can expose whatever we wish to the outside world, so we can have both public and private methods.



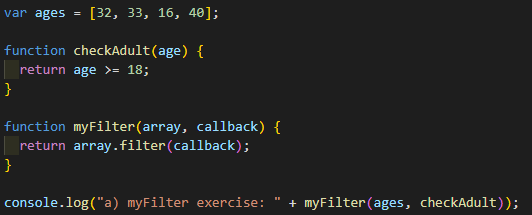
* User-defined Callback Functions (writing functions that take a callback)



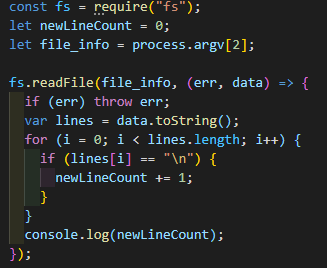
* Explain the methods map, filter and reduce
  + Map: Takes a list of something and displays it a certain way according to the map function you’ve made.



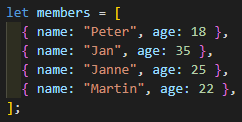
* + Filter: Takes an input and checks with a function that the parameter return “true” and gives the new input back.



* Provide examples of user-defined reusable modules implemented in Node.js (learnynode - 5)  
  (MAKE IT MODULAR)



* Provide examples and explain the es2015 features: let, arrow functions, this, rest parameters and destructuring objects.
  + Let: Declares a block-scoped local variable, optionally initializing it to a value.



* + Arrow functions: is an alternative from traditional function expression. This can also in some situation make some code work that otherwise [wouldn’t work](https://javascript.info/arrow-functions) with as a traditional function.

  
  
Generally, a traditional function is easier to read. Where Arrow functioning is more compact.

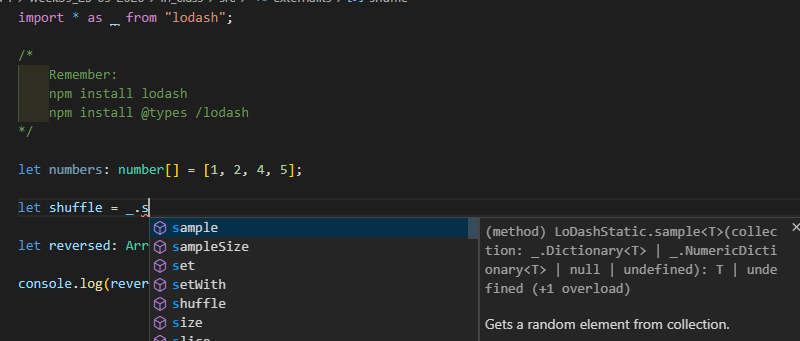
* + This: In JavaScript this always refers to the “owner” of the function we’re executing, or rather to the object that a function is a method of.



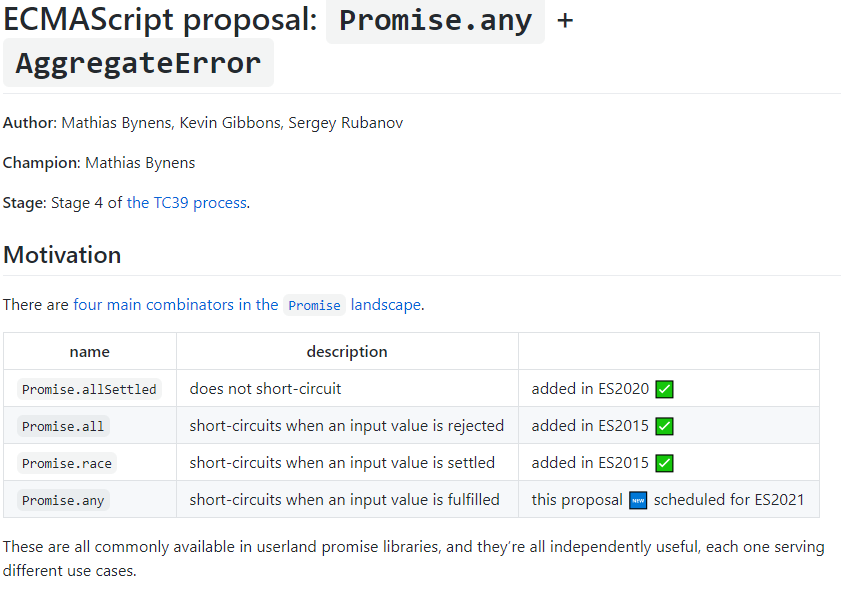
* + [Rest parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters):
  + [Destructuring objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment): Is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects into distinct variables.
* Provide an example of ES6 inheritance and reflect over the differences between Inheritance in Java and in ES6.
* Explain and demonstrate, how to implement event-based code, how to emit events and how to listen for such events
  + Week36\_02-09-2020/nodeServer.js

**ES6,7,8,ES-next and TypeScript**

* Provide examples with es-next, running in a browser, using Babel and Webpack
  + Babel: week38\_18-09-2020/using\_node\_with\_babel
  + Webpack: week38\_18-09-2020/webpack-demo
* Explain the two strategies for improving JavaScript: Babel and ES6 + ES-Next, versus Typescript. What does it require to use these technologies: In our backend with Node and in (many different) browsers
  + TypeScript helps us with targeting specific EMCA versions, translates it into working JavaScript it is also “closer” to Java as we can put types on “everything”, add optional things and much more. Easier to find and fix the errors straight away while coding.
* Provide **examples** to demonstrate the benefits of using TypeScript, including, types, interfaces, classes and generics
  + \week39\_25-09-2020\empty-project\src\interfaces.ts
* Explain how we can get typescript code completion for external imports.
  + We have to import it in order to get code completion.  
    \week39\_25-09-2020\in\_class\src\external.ts



* Explain the ECMAScript Proposal Process for how new features are added to the language (the TC39 Process)
  + [ECMA TC39](https://tc39.github.io/process-document/). The committee handles making ECMAScript better and easier to use. The TC39 process has 5 stages to get the specifications in the next release.
  + Stage 0: Strawperson  
    Allows input into the specification
  + Stage 1: Proposal  
    Make the case for the addition, describe the shape of a solution, identity potential challenges.
  + Stage 2: Draft  
    Precisely describe the syntax and semantics using formal spec language
  + Stage 3: Candidate  
    Indicate that further refinement will require feedback from implementations and users
  + Stage 4: Finished  
    Indicate that the addition is ready for inclusion in the formal ECMAScript standard

  
<https://github.com/tc39/proposal-promise-any>

**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")
  + \week37\_11-09-2020\exercises\ex2.js
* Example(s) that demonstrate how to execute asynchronous (promise-based) code in **serial** or **parallel**
  + \week37\_11-09-2020\exercises\ex2.js
  + \week37\_11-09-2020\exercises\ex3.js
* Example(s) that demonstrate how to implement **our own** promise-solutions.
  + \week37\_11-09-2020\exercises\ex3.js
  + \week38\_18-09-2020\using\_node\_with\_babel\index.js
* Example(s) that demonstrate error handling with promises
  + <https://javascript.info/promise-error-handling>

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

* We use async/await over normal promises is to make the code easier to read, there isn’t 1000 .then(s) making a call back hell

Provide examples to demonstrate

* Why this often is the preferred way of handling promises
  + \week37\_11-09-2020\exercises\ex2.js
* Error handling with async/await
  + \week37\_11-09-2020\exercises\ex1\_tester.js

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