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

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 a scripted language
  + Java is both a language and a platform
  + General differences in language features.
  + Blocking vs. non-blocking

SVAR: JavaScript bruges mest til client side scripts for at gøre hjemmesider mere levende, Java er både mange flere ting.

Java skal kompileres.

Java er objektorienteret, JS er et objektorienteret scripting sprog.

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

SVAR: Node bruger asynkron programmering, hvilket betyder, at den ikke venter på, at et job er færdiggjort før, at den begynder på en ny task.

Npm: package manager for Node.js. En package indeholder alle de filer, der er brug for i et modul. Moduler er JS biblioteker. Pakker placeres i mappen ”node\_modules”

* 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.

SVAR: JavaScript kører single-threaded – der sker kun én ting ad gangen.

Event loopet checker kontinuerligt call stack’en for at se, om der er en funktion, som skal køres. Imens tilføjer den enhver funktion, den finder til call stacken og eksekverer dem.

Blocking: Er, når noget må vente på, at noget andet er færdigt før, at det kan udføres. Ting, der kan blokere er synkrone netværkskald og uendelige loops.

Nonblocking:

Callback queue: Det er her, den asynkrone kode kommer hen og venter på at blive eksekveret.

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

SVAR: Asynkrone metoder i Node er nonblokerende, men nogle af dem har lignende metoder, der blokerende, altså synkrone, og de ender med Sync, f.eks. readFileSync.

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

SVAR:

JavaScript Engine er et computerprogram, der eksekverer JS. Det oversætte soure kode til machine kode og eksekverer det. Alle kendte web browsere har en. V8 (udviklet af Google i 2008 til deres Chrome browser), SpiderMonkey (Mozilla), JavaScriptCore (Apple).

Node.js er et JavaScript Runtime Environment, som giver adgang til filsystemer, netværk og andre ting, som normalt ikke kan ses i web browsere. Et runtime environment er en slags container, som indeholder biblioteker, web apis osv.

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

SVAR:

Babel: En slags oversætter, der konverterer JS til ES5, som kan køres i enhver browser. Kan både bruges i frontend og backend

Webpack: Et værktøj, der kompilerer JS moduler, også kaldet en module bundler. Den omdanner en mængde filer til én fil, som kan køre appen. Bruges mest i frontend.

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

* Variable/function-Hoisting

SVAR: Hoiste betyder at løfte op.

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

SVAR:

* Function Closures and the JavaScript Module Pattern

SVAR:

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

SVAR:

* Explain the methods map, filter and reduce

SVAR:

**Map:** Ændrer hver en værdi i et array. Har en callback som første argument. Et nyt array skabes

**Filter:** Filtrerer visse værdier ud af et array. Har også en callback som første argument. Et nyt array skabes

**Reduce:** Tager alle elementerne i et array og reducerer dem til én værdi, f.eks. summen.

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

SVAR:

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

SVAR:

**Let:** I modsætning til var, som er funktions-scoped, er scope for let og const blokken.

**Arrow functions:** Funktioner uden “function” som keyword, når de defineres. Man behøver heller ikke at skrive ”return”, og hvis der kun er én parameter, behøves parentes ikke.

**This:** I en normal funktion refererer this til det objekt, der kalder funktionen. I en arrow funktion refererer this til ejeren af funktionen

**Rest parameters:**

**Destructuring objects:** Er, når man tager properties fra et objekt og binder dem til nye variabler.

const name = hero.name;

const realName = hero.realName;

// is equivalent to:

const { name, realName } = hero;

**Arrays:**

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

SVAR:

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

SVAR:

ES6,7,8,ES-next and TypeScript

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

SVAR:

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

SVAR:

* Explain how we can get typescript code completion for externaimports.

SVAR:

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

SVAR:

**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 implement **our own** promise-solutions.
* Example(s) that demonstrate error handling with promises
* Example(s) that demonstrate how to execute asynchronous (promise-based) code in **serial** or **parallel**

SVAR: Promises løser problemet med callback hell, hvor mange asynkrone funktioner er pakket ind i hinanden, og det ligner et juletræ, som er svært at overskue.

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

SVAR: Ordet ”async” før “function” betyder, at funktionen altid returnerer et promise. Await får JS til at vente indtil promise’t er løst. I mellemtiden kan JS engine sagtens løse andre jobs. Med andre ord, er det en anden måde, at skrive ”promise.then” på.

Provide examples to demonstrate

* Why this often is the preferred way of handling promises

SVAR:

* Error handling with async/await

SVAR:

* Serial or parallel execution with async/await.

SVAR: