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

- **Explain the differences between Java and JavaScript + node**

Oprindeligt blev Java og JavaScript udviklet til at tjene to forskellige formål; java er designet som et programmeringssprog til at bygge standalone applikationer, hvor javascript kan betegnes som et scripting sprog som bruges specifikt til at interagere med web teknologier, primært HTML.

Dette oprindelig formål, kan komme til udtryk på flere måder:

- Compiled vs. Interpreted:
  Java kompileres til bytecode og kører på virtual machine. Dvs. source koden omdannes til machine code og kopien af machine koden overføres til klienten. Klienten har ikke adgang til sourcekoden.

- Interpreted/scripting: Source koden sendes præcis som den er til klienten, som fortolker/læser koden linje for linje.

- Fordelen ved compiled sprog er, at source koden beskyttes, men ikke så effektiv hvis den skal kører på cross platform, grundet forskellen på arkitektur, hardware, os-systemer.

Blocking vs non-blocking:
Blocking er synkront, som referer til udførelse der blokerer for anden udførelse, indtil køen er udført.
Non-blocking er asynkront.

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

Ligesom Java har set eget runtime miljø med JVM – java virtual machine, har Javascript også et miljø udenfor browseren. Node.js. Den bruger Chromes v8 engine. NPM(node package manager) er pakker som Node kan bruge. Dette gør det også muligt, for at bruge javascript til at lave fullstack udvikling.

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.

<u>Callback :</u>

callback function er en funktion der gives til en anden funktion som parameter/argument .

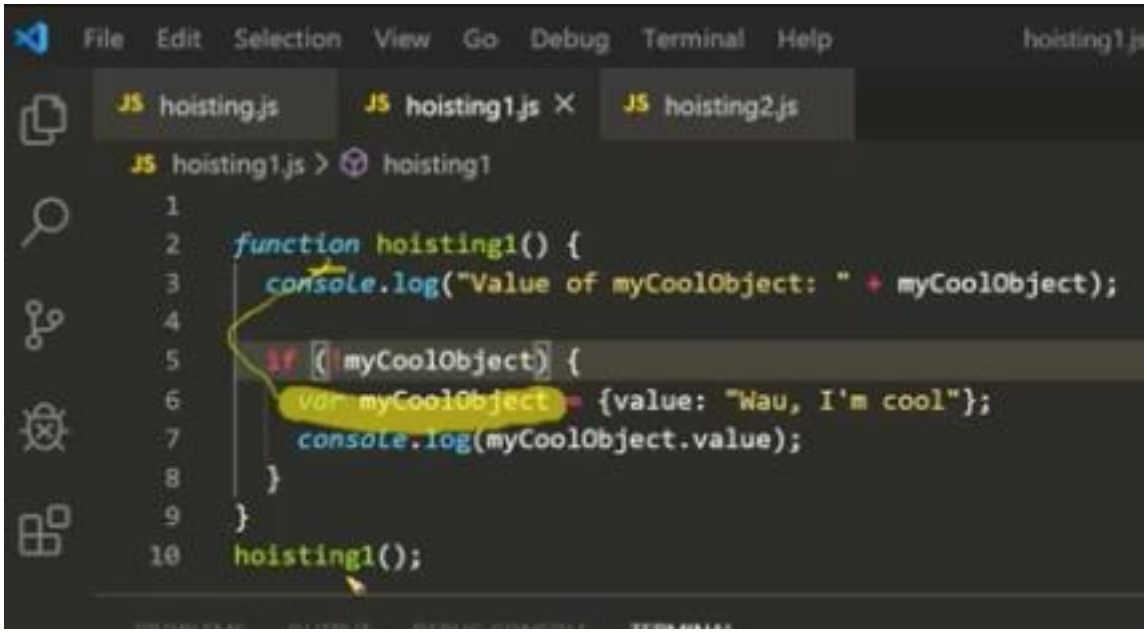
Hoisting:

Scope af koden. Hvornår og hvordan du kan få adgang til en variabel.

issaFunc();

function issaFunc() {
console.log('I'm a function');

vi kan få adgang til issafunc før den er deklaret , på grund af hoisting, "at løfte op", selvom den ikke er en global variabel.



Når var deklares, løftes den op. Compiler/fortolker vil tage erklæringen og lægge det op. Kun erklæringen (var mycoolobject) og ikke værdien.

Let giver block scope(så gør ikke det samme).

Closures:

Specielt objekt der kombinerer to ting: en funktion og miljøet som miljøet blev skabt i.
Closure is when a function remembers and continues to access variables from outside its scope, even when the function is executed in a different scope.

Linje 10 kan stadig udskrive ”Mozilla”, selvom vi er udenfor scopet. Den har stadig adgang til variablen.

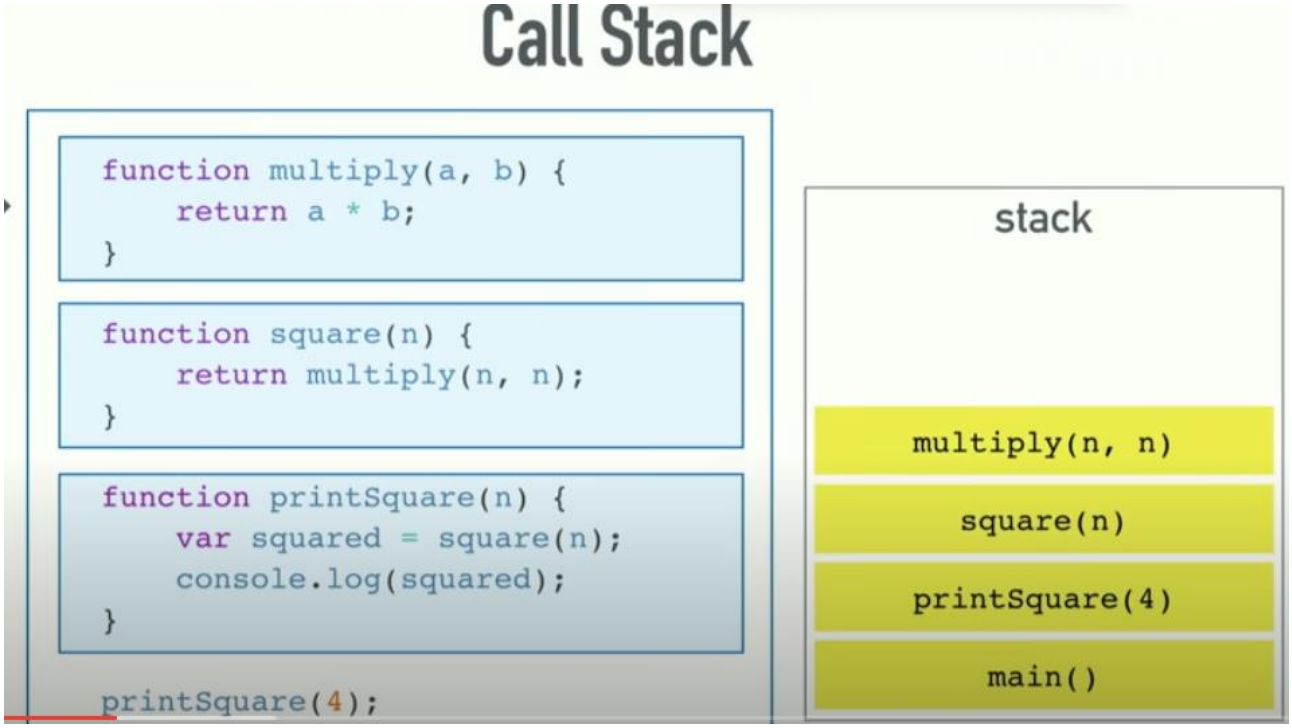
<u>This:</u>

Det objekt der står foran .(dot) er this. Ligesom java med andre ord.

Event loop:

JavaScript er single threaded, som betyder at den kan gøre en ting af gangen. One thread ==one call stack == one thing at a time.

Et stykke kode af gangen.



Return fjerner fra stack så når multiply return osv. fjernes de fra stacken.

Blocking: ”kode der er langsom”. Network requests kan være slow f. eks.

Vi burger ikke threads i javascript, med single thread og derfor bliver vi bare nødt til at vente til den er færdig med et network request, vi har ikke en måde at håndtere det på ellers.

Men hvorfor er det er problem? Fordi vi kører kode i browsers. Browser bliver blokeret/stopper indtil request er klar. Call stack har ting på den der skal køres / så den stopper.

Hvad er løsningen?

Asynkront callbacks. Men hvad gør det? Her kommer concurrency og event loopet i billedet.

Browser har web api’er. DOM, ajax, setTimeout. Browser starter en timer, når man bruger settimeout, når den er færdig smider den callback i task queue, og så kommer den i event loop.

Eventloop kigger på stack og task queue, hvis stack er tom, så smider den callback fra task queue i stack.

Callbacks kan være to ting: en funktion der kalder en anden funktion, eller et asynkront callback, som skubbes i callback queuen på et tidspnkt.

Node:
Node eller Node.js crossplatform runtime envoirnment for at køre javascript udenfor en browser. Bruges tit til backend og Api’er.

Forskellen mellem var, let og const:

Var er globalt og funktions scope. Let og const er blocke scoped (virker kun i en block).
Brug let og cons.