

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





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
Javascript kører runtime, hvor java bliver kompileret om til "byte kode"
 - Java is both a language and a platform
Javas kode bliver compiled ned til bytekode, som bliver aflæst af JVM (Java Virtual Machine).
 - General differences in language features.
JavaScript er primært brugt i forbindelse med webudvikling, og kan som udgangspunkt kun køres igennem en browser, medmindre man bruger hjælpeværktøjer som node.
Java er "strongly- "typed"" og kan køre på flere tråde.
Java er primært Objekt Klasse orienteret.
 - Blocking vs. non-blocking
Javascript er som udgangspunkt "Blocking" dvs at alt kode bliver lagt i "stacken" og bliver læst i en bestemt rækkefølge.
Hvorimod Java er "non-blocking" da vi kan køre kode parallelt fordelt på flere tråde.
- Explain generally about node.js, when it "makes sense" and *npm*, and how it "fits" into the node ecosystem
Node er en platform hvorpå man kan køre javascript kode udenom om browseren.
Node bruger en event-baseret og non-blocking model som gør at javascript kan bruges ved fullstack udvikling (backend + frontend)

NPM står for "Node Package Management" og bruges til at hente og holde styr på libraries og dependencies der gør det lettere at bygge større applikationer i javascript.
- 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.
Eventloop'et holder styr på callstacken og callback que'en.
Hvis callstack'en skulle være tom, tager eventloop'et det første event i callback que'en og flytter over i callstack'en

Når et event er blevet eksekveret i callstack'en er det færdigt og bliver fjernet fra stacken. Som forklaret ovenfor kan javascript kun eksekverer et event ad gangen, og er derfor som udgangspunkt blocking.

- What does it mean if a method in nodes API's ends with xxxxxxSync?
Det betyder at ens kode først bliver eksekveret når "sync" funktionen er færdiggjort. (blocking)
-  Explain the terms JavaScript Engine (name at least one) and JavaScript Runtime Environment (name at least two)
Javascript engine er et program der kan eksekvere javascript kode. Javascript kode compileres "runtime": altså at det bliver renderet når koden eksekveres. Chrome bruger V8 som engine og edge bruger chakra.
- Explain (some) of the purposes with the tools Babel and WebPack and how they differ from each other.  Use examples from the exercises.

Babel kan bruges til at transpilere kode der er skrevet i senere versioner af ES(ecmaScript) til tidligere versioner der kan bruges i flere forskellige browsere. Da man ikke altid kan forvente at alle browsere er udviklet lige langt.

Webpack er et værktøj som sørger at pakke større projekter til et bundle, og sikrer samtidigt at din kode bliver pakket i rigtig rækkefølge og med de rigtige referencer.

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

- Variable/function-Hoisting
Hoisting er en javascript mekanisme som gør at variabler og funktioner bliver deklareret som det første når koden eksekveres. Men det er ikke ensbetydende at de kan bruges før de bliver defineret.
Hoisting bruges altså til at fortælle at der på et tidspunkt i koden kommer til at være en variabel eller funktion med dette navn.

```
// Outputs: undefined
console.log(declaredLater);

var declaredLater = "Now it's defined!";

// Outputs: "Now it's defined!"
console.log(declaredLater);

//
```

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

For Javascript:

"this" referer til det objekt, det er tilknyttet til.

Hvis "this" bruges alene referere det derimod til det globale objekt

Hvis "this" bruges i en funktion refererer det også til det globale objekt.

Hvis "this" bruges i forbindelse med et event, så refererer "this" til elementet som modtager eventet.

For Java/.net :

"this" refererer til det bestemte objekt, som den pågældende metode er brugt til at eksekverer.

```
var person = {
  firstName: "test",
  lastName : "test",
  id      : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

- Function Closures and the JavaScript Module Pattern
"Closures" gør det muligt at en variabel der er defineret i et ikke globalt scope. Variablen kan derfor kun nå indenfor det lokale scope. Og når først den er blevet defineret genbruges.

```
var counter = () => {
  var privateCounter = 0;
  function changeBy(val) {
    privateCounter += val
  }
  return {
    inc: () => {changeBy(1);},
    dec: () => {changeBy(-1);},
    value:() => {return privateCounter;},
  }
}
```

- User-defined Callback Functions (writing functions that take a callback)
En callback function er en function som givet videre ind i en anden function som et argument.

```
function greeting(name) {
  alert('Hello ' + name);
}

function processUserInput(callback) {
  var name = prompt('Please enter your name. ');
  callback(name);
}

processUserInput(greeting);
```

- Explain the methods `map`, `filter` and `reduce`
En map metode laver et nyt array med resultatet af en function for hvert element i et array
En filter metoder laver et nyt array for hvert element i et array som stemmer overens med filteret krav. (eksempelvis string på højst 3 karaktere.)

En reduce medtode laver et array om til én enkelt værdi ud fra et medgivet parameter.

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

```
var fs = require('fs');
var path = require('path');

function filterFiles(list, filter) {
  return list.filter(function(file) {
    return path.extname(file) == '.' + filter;
  });
};

module.exports = function(directory, filter, callback) {

  fs.readdir(directory, function(error, list) {
    if (error) {
      return callback(error);
    }
    var filtered = filterFiles(list, filter);
    return callback(null, filtered);
  });
};
```

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

```
//This is a regular function
hello = function() {
  return "Hello World!";
}

//This is an arrow function
hello = () => {
  return "Hello World!";
}
```

```
//rest parameters
function sum(...theArgs) {
  return theArgs.reduce((previous, current) => {
    return previous + current;
  });
}
console.log(sum(1, 2, 3));
// expected output: 6
console.log(sum(1, 2, 3, 4));
// expected output: 10
```

```
var person = {
  name: "Klaus",
  hobby: "football"
}
//Without ES2015
var name = person.name
var hobby = person.hobby
//With ES2015
const {name,hobby} = person
```

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

Fra ES6 er det blevet muligt at gøre brug af en speciel type funktioner. Class: som læner sig op af hvad vi fra Java verden kender i det objekt orienterede miljø.

Ligesom i Java, kan vi gøre brug af nedarvning til at implementere attributter på arvede klasser.

```
class Cat {
  constructor(name) {
    this.name = name;
  }

  speak() {
    console.log(`${this.name} makes a noise.`);
  }
}

class Lion extends Cat {
  speak() {
    super.speak();
    console.log(`${this.name} roars.`);
  }
}
```

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

ES6,7,8,ES-next and TypeScript

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

```
//Get any promises ES-Next function
Promise.any([myPromise(3), myPromise(4)]).then((res) => {
  console.log("Promise.any:");
  console.log(res.name);
  console.log();
});|
```

- 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

For Babel/ES6+Next kræves det at vi installerer npm, og vi opsætter en konfigurations-fil.

Både babel og Typescript kan transpile koden fra ES6 til tidligere versioner.

TypeScript kompilere hele projektet på én gang, hvor babel transpilere én file ad gangen.

TypeScript er type baseret, og Typescript skal kompileres til en JS-fil for at kunne køres i en browser.

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

```
//Generics
function reverseArr<T>(arg: any[]): any[] {
  return arg.reverse();
}
console.log(reverseArr<string>(["a","b","c"]));
console.log(reverseArr<number>([1,2,3]));
console.log(reverseArr<boolean>([true,true,false]));
console.log(reverseArr<number>(["a","b","c"]));

//Class og inheritance
abstract class Shape {
  private _color:string;
  constructor(color:string){
    this._color = color;
  }
  abstract get area():number;
  abstract get perimeter(): number;
  toString(): string{
    return `Shape's color: ${this.color}, Area: ${this.area}, Perimeter: ${this.perimeter}`
  }
  get color(): string {
    return this._color
  }
  set color(color : string) {
    this._color= color
  }
}

class Circle extends Shape{
  _radius: number;

  constructor(color:string, radius: number ){
    super(color)
    this._radius = radius
  }

  get radius(): number {return this._radius;}
  set radius(radius:number) {this._radius = radius;}

  get area(): number {
    return (Math.PI * Math.pow(this._radius,2));
  }
  get perimeter(): number {
    return (2 * Math.PI * this._radius);
  }
}
```

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

Stage0 Strawman– Alle kan komme med et forslag til et nyt feature. Hvis forslaget skal komme videre herfra, skal det have nok opbakning, fra "champions/patrons"

Stage1 Proposal– Forslaget bliver presenteret, og skal nu være delvist implementeret for at det kan blive demonstreret til communityet.

Stage2 Draft– For at nå dette punkt skal alle specifikationer være dokumenterede, og en skitse lavet.

Stage3 Candidate– For at nå dette punkt skal forslaget være gennemtestet, ved skrevne tests. For at sikre at kvaliteten er i orden.

Stage4 Finished– Forslaget kan nu komme med i den næste release.

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")

```
function makeSecureRandom(size) {
  return new Promise((resolve, reject) => {
    crypto.randomBytes(size, function (err, buffer) {
      if (err) {
        reject(err);
      } else {
        resolve({ "length": size, "random": buffer.toString('hex') });
      }
    });
  });
}

module.exports = async function getSecureRandoms(values) {
  let promises = [];
  values.forEach(element => {
    let tmp = makeSecureRandom(element);
    promises.push(tmp);
  });
  return hexObject = {
    "title": "6 Secure Randoms",
    "randoms": await Promise.all(promises)
  };
}

getSecureRandomsPromise([48, 40, 32, 24, 16, 8])
```

- Example(s) that demonstrate how to execute asynchronous (promise-based) code in **serial** or **parallel**


```

function getPlanetforFirstSpeciesInFirstMovieForPerson(id) {
  let result = {};
  let url = "https://swapi.dev/api/people/" + id;
  myPromise(url)
    .then((data) => {
      result.name = data.name;
      return myPromise(data.films[0]);
    })
    .then((data) => {
      result.movieTitle = data.title;
      return myPromise(data.species[0]);
    })
    .then((data) => {
      result.firstSpecies = data.name;
      return myPromise(data.homeworld);
    })
    .then((data) => {
      result.homeworld = data.name;
      console.log(result)
    })
}

async function getPlanetforFirstSpeciesInFirstMovieForPersonAsync(id) {
  let url = "https://swapi.dev/api/people/" + id;
  let name = await myPromise(url)
  let movieTitle = await myPromise(name.films[0]);
  let firstSpecies = await myPromise(movieTitle.species[0]);
  let homeworld = await myPromise(firstSpecies.homeworld);

  let result = await {
    name: name.name,
    film: movieTitle.title,
    firstSpecies: firstSpecies.name,
    homeworld: homeworld.name
  }
  console.log(result)
}

getPlanetforFirstSpeciesInFirstMovieForPersonAsync(1)
getPlanetforFirstSpeciesInFirstMovieForPerson(1)

```

- Example(s) that demonstrate how to implement **our own** promise-solutions.

```

const createSecureString = (size) => {
  return new Promise((resolve, reject) => {
    crypto.randomBytes(size, (error, buffer) => {
      if(error) return reject(error);
      resolve({
        length: buffer.length,
        random: 'A buffer with ' + buffer.length + ' hex-characters'
      });
    });
  });
};

```


- Example(s) that demonstrate error handling with promises

Se ovenfor

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

Async/Await er egentlig bare promises i et pænere format. Det har samme funktionalitet, da en async metode returnere et promise, og await egentlig bare "afventer promise"

Provide examples to demonstrate

- Why this often is the preferred way of handling promises
- ***SE BESVARELSE OVENFOR*** (Example(s) that demonstrate how to execute asynchronous (promise-based) code in **serial** or **parallel**)
- 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