ExamQuestions 5

1)Explain the two strategies for improving JavaScript: ES6 (es2005) + ES7, versus Typescript. What does it require to use these technologies: In our backend with Node, in (many different) Browsers.

ES6 og typescript er et sprog som ikke kan læses af de fleste browsers(endnu i hvert fald). Derfor er man nød til at bruge en compiler som kan lave det om til et sprog som browsers kan forstå. Her kan man med god vilje lave det om til ES5 da det er et sprog mange nuværende browsers kan forstå

2) Provide examples and explain the es2005 features: let, arrow functions, this, rest parameters, de-structuring assignments, maps/sets etc.

Når man bruger 'let' er det for at begrænse sit scope. Man bruger det så det kun er tilgængelig i den function man arbejder med. det kan man foreksempel gøre hvis man gerne vil bruge det samme variable navn i flere functioner, og så er man også sikker på man ikke bruger den hvis man ikke skal have tilgang til den.

```
function showLet()

{

let testScope = "testing";

console.log(testScope)// Denne command vil virke da vi kender testScope

}

showLet();

console.log(testScope);// Dette vil ikke virke da testScope er deklaret lokalt i showLet functionen.
```

Når man benytter sig af en arrow funktion er det for at gøre det nemmere at læse. Det kan forkorte ens kode, og hvis man ikke behøver at navngive sin funktion er den nem at bruge. en arrow funktion kan se sådanne ud.

```
var test = (data => {

let result = data+10;

return result;

})
```

Og den vil så se sådan her ud som en normal function

```
var test = ( function(data)  {

let result = data+10;

return result;

})
```

når man bruger arrow funktioner så beholder THIS sit scope.

3) Explain and demonstrate how es2015 supports modules (import and export) similar to what is offered by NodeJS.

Hvis vi kigger på vores ovenstående eksempel med arrow funktioner. så vil man kunne eksport/import den på denne måde.

Eksport module.js

```
module.exports = {_test : test};
```

Import module.js

```
let importer = require("./Eksport module");

console.loG(importer (5));// ville udskrive 15
```

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

```
class Shape {

 constructor(color) { this._color = color; }

getArea() { return undefined; }

getPerimeter() { return undefined; }

toString() { return `this shape is of the color ${this._color}, it has an undefined area and perimeter!!`; }

getColor() { return this._color; }

setColor(color) { this._color = color; }

}


class Circle extends Shape {

constructor(radius, color) { super(color); this._radius = radius; }

getArea() { return undefined; }

getPerimeter() { return undefined; }

toString() { return `this circle is of the color ${this._color}, a radius of ${this._radius}. It has an undefined area and perimeter!!`; }

getRadius() { return this._radius; }

setRaduis(radius) { this._radius = radius; }

}
```

Når man vælger at arbejde med classer fungere arv ligesom i java, dog kan man arve direkte fra et objekt og ikke en klasse, det er lidt aftypisk for java.

5) Explain about Generators and how to use them to:
Implement iterables
Implement blocking with asynchronous calls

En generator kan se således ud:

```
function *foo() {
        yield 1;
        yield 2;
        yield 3;
        yield 4;
        yield 5;
}

var it = foo();
it.next(); //første run giver 1 next giver 2 osv.
```

Man bruger generators når man gerne vil have at ens funktion skal "vente". Det næste bliver ikke kørt for der er kaldet next(). Så på den måde kan man lave blockerende kald som venter.

6) Explain about promises in ES 6, Typescript, AngularJS including:
The problems they solve.

Promises løser et essentielt problem som “non-blocking” sprog/apis har nemlig, hvad gør man hvis man har en operation der SKAL eksekveres på et specifikt tidspunkt. Non-blocking adfærd er ok, når man bare vil lave alerts på en hjemmeside, skal man dog lave evt et http-request, hvor resultatet bruges I et databasekald (begge async operationer), bliver det lige pludseligt vigtigt, at du kan lave httprequestet først (hver gang). Promises kan da bruges på følgende måde (I forhold til ovenstående eksempel): “Lav et http-request og lov mig, at du vil gemme resultatet I følgende variabel, når du er færdig – der så kan bruges I DB kaldet” (er det underligt at snakke sådan til sin computer?). I korte træk tillader promises altså udvikleren, at lave asynkone kald I et singletrådet, non-blocking miljø.

Examples to demonstrate how to avoid the "Pyramid of Doom"

Examples to demonstrate how to execute asynchronous code in serial or parallel

Eksempel: vi ønsker en række tal fra en async operation, I en specifik rækkefølge, for at sikre det skal man normalt gøre følgende (psudokode

getNumber(funktion(array, callback) {

var asyncCall callback(asyncCall)

}

Hvis vi ville have say 6 tal, skulle vi da kalde denne funktion seksgange inde I den første, således at vi får nedestående eller lign:

```
func(){
        func(){
                func(){
                        func(){
                                func(){
                                }
                        }
                }
        }
}
```

I stedet kan man bruge pormises og gøre følgende (eks fra opgave), derfor med en smart lille promiseFactory:

let makeRandoms = function (callback) {

let p1 = promiseFactory(48);

let p2 = promiseFactory(40);

let p3 = promiseFactory(32);

let p4 = promiseFactory(24);

let p5 = promiseFactory(16);

let p6 = promiseFactory(8);

```
let result = {
 title: "6 secure randoms", randoms: []
}

 Promise.all([p1, p2, p3, p4, p5, p6]).then(arr => {
 result.randoms = arr; callback(result);
});
}
```

Ved at man gør det på denne måde kan man også være ligeglad om det er nr 1 eller n5 der bliver kørt først, for du kan selv vælge hvordan de vil blive skrevet ud, i dette tilfælde, p1 først og p6 til sidst(den måde de kom ind på).