**Forskelle mellem JS og Java**

Java er et OOP programmerings sprog mens JS er et OOP scripting sprog.

Java laver koder som kører i en virtual maskine eller browser, mens JS kun kører koden på browseren.

Java kode skal compiles mens JS kode er alt i text.

De skal have forskelle Plugins.

**Forskellen mellem Compiler language og Scripting Language**

I et compiler sprog skal du compile først hvor et Scripting language ikke skal.

**Forklar Strict**

Med strict kan du ikke bruge udeclarede variabler

a = 5 // sat før declareret

console.log(a); // prøver at bruge før declareret

var a; // declareret efter console

Strict vil stoppe dette.

**Hoisting**

Hoisting er når du har adgang til en variable eller function uden det er declareret fx;

b = 2;

function a ()

{

console.log(b)

}

a();

var b;

**Forklar This i JS**

this i java er en reference til det nuværende object der er blevet instansieret af classen.

I javaScript er this normalt en reference til det object som har ejerskab over metoden men det ændre sig alt efter hvordan functionen er kaldet.

console.log(this.document === document); // dette er true.

**Closures**

En closure er object som combinere to ting:

En function og Miljøet som functionen var lavet i. Miljøet indeholder alle lokale variabler som var i scopet da closuren var lavet.

variable1 = 1; // Global Scope  
**var** variable2 = 2; // Not within a function: Global Scope  
  
**function** funcName() {  
 variable3 = 3; // No var keyword: Global Scope  
 **var** variable4 = 4; // Local Scope only  
 console.log(variable1);  
}  
funcName();  
  
console.log(variable3); // returnerer 3  
console.log(variable4); // returnerer ReferenceError: variable4 is not defined.

**Closures - The Module Pattern**

Du kan ikke i JS lave private functioner men du kan bruge closures til at løse dette.

var modularpattern = (function() {

var sum = 0 ;

return {

add:function() {

sum = sum + 1;

return sum;

},

reset:function() {

return sum = 0;

}

}

}());

Nu kan du så kun bruge de funktioner som er lagt ind i variablen modularpattern til at lave noget om og ikke ændre på etc sum = 5 men at skrive;

modularpattern.add();

**Immediately-Invoked Function Expressions (IIFE)**

IIFE er indikere typisk at fuktionen vil blive kørt med det samme, og variablen vil indeholder resultatet af functionen, ikke funktionen den selv.

Kan redde tid fra andre som leder efter funktioner som bliver brugt andre steder.

**var** spaceworm = (**function**(){

**var** lel = 1337; **return** lel;  
}());  
console.log(spaceworm); // a returnere leet.

**JavaScript Prototype**

Alle JavaScript objekter har en prototype som også er et objekt.

Alle javascript objekter nedarver deres egenskaber og metoder fra deres prototype.

Prototype er basically et skelet som du kan bruge til at lave et nyt objekt med de basalle objekter og adde / slette fra objektet senere. Fx;

function Person(first, last) {

this.firstName = first;

this.lastName = last;

}

var myFather = new Person("ASD", "DSA");

Person.prototype.nationality = "English";

hvis du så kalder myFather.prototype.nationality returnere den så English.

**User defined Callback Functions**

En callbacket funktion er en fuktion som bliver "passed til en anden funtkionen som et parameter", den callback funktion er så kaldt "kørt" inden i den anden funktion som derefter returnere en værdi på funktionen.

**Node.js og NPM**

npm er en node.js package manager. Som navner betyder så kan du bruge det til at installere programmer, også hvis du bruger det i udvikling kan det være nemmere at specificere dependencies.

Node.js er en platform som er bygget på chrome's js runtime for at nemt bygge et hurtigt og scalertbart netværks application.

Node.js bruger et event-driven, ikke blockerende I/O model der gør den simpelt og effektivt, perfekt til data intensive nutids applicationer der kører på flere enheder.

**Event Loop**

The Node.js event loop runs under a single thread, this means the application code you write is evaluated on a single thread. Nodejs itself uses many threads underneath through libuv, but you never have to deal with with those when writing nodejs code.

Every call that involves I/O call requires you to register a callback. This call also returns immediately, this allows you to do multiple IO operations in parallel without using threads in your application code. As soon as an I/O operation is completed it's callback will be pushed on the event loop. It will be executed as soon as all the other callbacks that where pushed on the event loop before it are executed.

**Provide examples of user defined reusable modules implemented in Node.js**

Man laver sin egen funktion som et node.js modul som man derefter kan require (ligner lidt java library etc scanner)

Fx;

var scan = require("scanner")