**Forskellen på Java og Javascript:**

Et program der er lavet i Java er nødt til at compile før det kan køre, mens et Javascript program ikke behøver at compile først. Dette er fordi Javascript er et scripted program.

Java laver applikationer som kører i en virtuel maskine eller browser, mens Javascript kode kun bliver kørt i en browser.

**Use strict:**

når man bruger "use strict" fortæller man at dette skal køres i strict format, hvor man foreksempel ikke kan bruge udeklerede variabler.

Det kan se således ud:

**Funtion (){**

**"use strict";**

**a=54;**

**}**

dette vil give en fejl da variable a ikke er blevet deklaret.

**Hoisting:**

hoisting er når man kan bruge en variable som man før deklere senere i sin kode Fx:

a="hej med dig"

function(){

console.log(a)

}

var a;

**This i Javascript vs java:**

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**

closures handler om at have styr på hvor ens scope ligger. Man kan have en variable som er set af alle funktioner og variabler som har scope i de funktioner de er blevet lavet i.

fx

var a = "hej":

function h(){

var b = "hej2"

function j(){

var c = "hej3"

}

Console.log(a);--- hej

Console.log(b);--- hej2

Console.log(c);--- error

};

Console.log(a);--- hej

Console.log(b);--- error

Console.log(c);--- error

**Module patterns:**

Med module patterns kan vi lave metoder som kun kan tilgås af objectet, eller kan tilgås ved hjælp af dens egne metoder.

Man kan lave en metode som er privat, men skal kan tilgås ved hjælp af en public metode.

fx:

var Module = (function() {

// Following function is private, but can be accessed by the public functions

function privateFunc() { ... };

// Return an object that is assigned to Module

return {

pubFunc: function() {

privFunc(); // publicFunc has direct access to privateFunc

}

};

}());

**Immediately-Invoked Function Expression :**

dette bruger vi til at få en function til at blive kørt med det samme. hvis man sætter en variable lig med function, så vil den allerede være blevet udførst når variablen kaldes. fx:

var a = (function(){

b=1;

return b;

}

**Javascript prototype:**

Javascript objekter nedarver altid fra en prototype. Alle objekter arver fra object.prototype, som indeholder nogle ting man kan gøre med objecter. Object.prototype har en constructor skelet som man kan bruge og tilføre ting til. FX:

* function Person(first, last, age, eyecolor) {  
      this.firstName = first;  
      this.lastName = last;  
      this.age = age;  
      this.eyeColor = eyecolor;  
  }  
  Person.prototype.name = function() {  
      return this.firstName + " " + this.lastName;  
  };

**Callback:**

En callback function er en function som bliver ”passed” til en anden function som et parameter. Den callback function er så kaldt (kørt) inden i den anden function.  
  
var names = ["kurt","ole","ib"];

function myforEach(arr,callback){

for(var i =0; i < arr.length; i++){

callback(arr[i]);

}

}

myforEach(names,function(name){

console.log(name);

});

**NPM:**

Npm er en package manager, hvor alle kan ligge deres forskellige koder op, som kan tilgås af andre. man kan så bruge npm install til at bruge disse packages i ens program. Npm gør det nemmere at vise dependencies.

**Node.JS**

node.js er en platform som er bygget på chrome's javascript. Det er et værktøj som gør det nemmere at lave store netværks applikationer. node-js bruger en event-driven, non-blocking i/O model, som gør det hurtigt og effektivt.

.