

# JAVASCRIPT

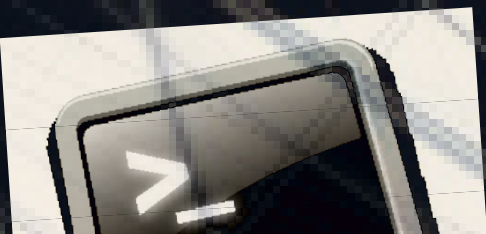
## NODE.JS

GRUNNLEGENDE PROGRAMMIERUNG

MODULER, EXPRESS

OBJEKTE, KLASSEN

DATABASER, JSON



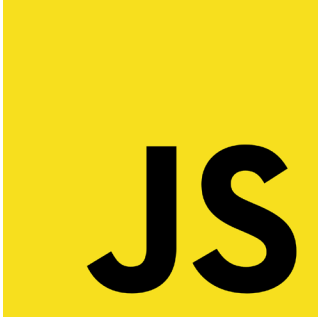
---

# Innhold

Introduksjon	4
Hva er Javascript?	4
Hva er Node.js?	4
Ressurser for å lære seg javascript	5
 Grunnleggende koding	 6
Skriving til console	6
Ulike typer variabler	6
Tallvariabel, desimaltall	6
Tabeller	7
Objekter	7
If og True/False	8
Løkker	8
For-løkker	8
Funksjoner	9
 Moduler	 12
To typer moduler	12
Node Package Manager, npm	14
Nodemon	15
 Webserver med Express	 17
Hva er express?	17
Hvordan installere, og starte det opp?	17
Hvordan sette opp en rute?	18
Input ved hjelp Skjema	20
Flere Skjema muligheter	21
 Server Side Rendering (SSR)	 22
SSR med Handlebars Template Engine	23
 Client Side Rendering	 30
Hvor legger vi javascript-koden	31

DOM - Document Object Model	32
Hendelser	33
Datalagring	34
Fil-lagring	34
Databasen Sqlite og Grafisk brukergrensesnitt	34
Databasen Sqlite og modul for koding i node.js	36
Dokumentasjon	40
Koding 2	41
Funksjoner som variabler	41
Sortering av objekter i array	41
Søke i en array	43
Klasser og objekter	44
Lesing og Skrivning til fil	45
Arrow Functions	46
Oppgaver	49
Oppgave 1 - Karractersystem	49
Oppgave 2 - BlackJack	50
Oppgave 3 - BakeKing	51
Oppgave 4 - Rollespill	52
Oppgave 5 - Fotball	52
Oppgave 6 - Vanlige funksjoner	53
Oppgave 7 - Arrow Funksjon	53
Oppgave 8 - Skrivning og lesing fil	54
Oppgave 9 - Guess the number	54
Oppgave 11 - Input/Output Mal til Express	55
Oppgave 10 - Påmeldingsapp	57
Om heldagsprøven 13. Desember	58
Obliger	62
Oblig 1 - Filmsystem	62
Oblig 2 - DOM	66
Oblig 3 - Express App	66

# Introduksjon



## Hva er Javascript?

En moderne nettside kan gjøre mer enn det en avis kunne i gamle dager. For eksempel kan du ved å trykke på knapper legge til en post i sosiale medier.

For å få til dette kjøres det kode i nettleseren din. Denne koden er skrevet med kodespråket Javascript.

## Hva er Node.js?



Viss du skriver inn en nettside-adresse i nettleseren din, sendes en forespørsel om å hente denne nettsiden til en datamaskin et eller annet sted i verden. Vi kaller datamaskinen der nettsiden ligger en server.

Viss nettsiden ligger lagret som en enkel html fil, kan denne da sendes tilbake til klienten (deg).

Noen ganger ønsker serveren å kjøre litt kode for å sette sammen nettsiden hver gang noen spør om å få nettsiden. Viss det er en nyhetsside, vil serveren kanskje hente alle de siste nyhetene, og sette disse sammen til en nettside, hver gang noen ønsker å se siden. For å få til dette må serveren kjøre en kode. Node.js gjør at vi kan skrive denne koden med samme kodespråket som kjøres i folks nettleser, nemlig javascript.

I tillegg til at Node.js brukes på serverene, er det en fin måte å bli kjent med programmering. Dette fordi vi kan kjøre koden enkelt og oversiktlig på en PC/MAC. I dette heftet vil vi bruke Node.js for å introdusere grunnleggende programmering.

# Ressurser for å lære seg javascript

- [Sololearn, Stegvis gjennomgang fra grunnleggende og oppover](#)
- [Eloquent JavaScript, Gratis bok om Javascript, Grundig gjennomgang, Anbefales](#)
- [Javascript materiale på ndla](#)
- [Objektorientert programmering i Javascript på ndla](#)
- [Codecademy – tekstbasert koding](#)
- [Khan Academy – Lag grafikk mens du lærer å kode](#)
- [freeCodeCamp – her kan du bli en sertifisert JavaScript-programmerer](#)
- [CodeHS - Steget opp fra blokkprogrammering](#)

## Installasjon på PC/MAC

### Steg 1 - Installere Visual Studio Code

Du kan skrive kode i et veldig enket tekst-redigeringsprogram som notepad på Windows, og TextEdit på Mac.

Et kanskje bedre valg er å bruke et tekst-redigeringsprogram som er laget for kodeskriving. Det finnes mange valg her. Et godt valg er å bruke Visual Studio Code fra Microsoft. Dette kommer til både Windows og Mac. Du kan laste dette ned her:

<https://code.visualstudio.com/>

### Steg 2 - Installere Node.js

På vanlige folks pc-er kjører javascript inne i nettleseren. Vi skal nå installere programmet node.js som gjør at vi kan kjøre javascript i kommandolinjen. Du laster ned og installerer programmet ved å gå inn på denne nettsiden:

<https://nodejs.org/en/> (Velg versjonen der det står "Current")

# Grunnleggende koding

## Skriving til console

```
console.log("Heil!")
```

## Ulike typer variabler

```
var variabel = "Dette er en gammel type variabel"  
let enNyVariabel = "Dette er en ny type variabel"  
const enKonstant = "Variabel som ikke kan endres"  
  
enKonstant = "Nytt innhold" //Fører til feilmelding  
variabel = "Nytt innhold"
```

Tre måter å lage variabler. let og const er det vi bruker vanligvis. Variabler som er laget med let kan endres, variabler laget med const kan ikke det.

## Tallvariabel, desimaltall

```
let tallVariabel = 1234  
tallVariabel = 1234.34
```

Du lager en let variabel med 1234 i verdi  
Du gir variabelen tallVariabel en ny verdi (1234.34)

### Å øke med 1

```
let tall = 1  
  
tall = tall + 1  
tall += 1  
tall++
```

Her ser man tre måter man kan bruke for å øke en variabel med 1.

# Tabeller

```
let liste = [1, 2, 3, 4]
console.log(liste[3])
```

Du lager en variabel som heter liste og inneholder 1, 2, 3 og 4.  
Du skriver ut 4 fra listen inn i konsollen.

```
let tabell = ["Børge", "Garpestad"]
console.log(tabell[0])
```

Først lager du en variabel som er en tabell og det console.log gjør er at den skriver ut det første navnet i tabellen. Dette er bestemt med tabell[0] som er i parentes, der 0 er første tingen i tabellen.

# Objekter

```
let person1 = {
  fornavn: "Hans",
  etternavn: "Hansen"
}

let person2 = {
  fornavn: "Truls",
  etternavn: "Trulsen"
}

let personer = [person1, person2]

console.log(personer[0].fornavn)
```

Først blir det opprettet to objekter som heter person1 og person2. Inne i objektet opprettes det fornavn og etternavn. Nå opprettes en liste som heter personer og den inneholder person1 og person2.

Da, printer vi ut person 1 fornavn fordi person [0], betyr første element i javascript, det betyr det skal printe ut person 1 fornavn.

# If og True/False

```
let sant = true
sant = false
```

Variabel "sant" først definert som true, etterpå endret til false

```
let etTall = 3
let endaEtTall = 3
let svar = etTall * endaEtTall

if (svar === 9) {
  console.log("Vi gikk inn i denne kodeblokken")}
else if ( svar === 3) {
  console.log("Nei, denne blokken")
} else {
  console.log("Standardsvaret")
}
```

3 plasseres i to variabler, og ganges sammen. Resultatet lagres i svar variabelen.

Det printes ut en melding som endrer seg etter hva svaret er.

## Løkker

### For-løkker

0  
1  
2  
3  
4  
5  
6  
7  
8  
9

```
for (let i=0; i<10; i++) {
  console.log(i)
}
```

Måte å kjøre log funksjonen 10 ganger. Tallene 0 til 9 blir skrevet ut.

0 hest  
1 geit  
2 gris  
3 hjort

hest  
geit  
gris  
hjort

```
let dyrTabell = ["hest", "geit", "gris", "hjort"]

for (var i=0; i<dyrTabell.length; i++) {
  console.log(i,dyrTabell[i]);
}

for (const dyr of dyrTabell) {
  console.log(dyr);
}
```

To måter å bruke en for-løkke til å skrive ut alt i en tabell.



# Funksjoner

```
function einFunksjon(fornavn, etternavn) {  
    console.log("Hei", fornavn, etternavn);  
}  
  
einFunksjon("Børge", "Garpestad");
```

Det blir definert en funksjon med navn "einFunksjon" med input variablene "fornavn" og "etternavn". Funksjonen logger til konsollen "Hei + fornavn + etternavn". Funksjonen blir deretter aktivert med inputene "Børge" og "Garpestad".

```
function kulFunksjon(navn) {  
    console.log(navn, "er kul")  
}  
  
for (let i = 0; i < 4569; i++) {  
    kulFunksjon("Børge")  
}
```

Denne kodesnutten definerer en funksjon som tar input "navn", deretter printer den "\${navn} er kul".

for (let i = 0...) er en for loop som kjører kulFunksjon 4569 ganger med input "Børge"

To viktige ord når man snakker om funksjoner er parameter og retur-verdi. Under forklares disse to begrepene.

## Parameter:

```
function printTotalLength(string1, string2) {  
    let length = string1.length + string2.length  
    console.log(length)  
}  
  
printTotalLength("Kasper", "Iversen")
```

Parameterene til funksjonen er her string1 og string2. Når vi bruker funksjonene, kan vi med andre ord sende inn data som to strenger.

## Retur-verdi:

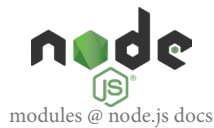
```
function getTotalLength(string1, string2) {  
  let length = string1.length + string2.length  
  return length  
}
```

```
let totalLength = getTotalLength("Kasper", "Iversen")
```

Denne funksjonen har to parametere som forrige eksempel. Disse muliggjør å sende data inn til funksjonen, i form av to `string(tekst)` variabler. I tillegg har denne funksjonen en retur-variabel, med datatypen `number`. Med andre ord, denne funksjonen returnerer et tall. Vi ser under funksjonen, at dette tallet blir lagret i variabelen `totalLength`.



# Moduler



En modul er et lite stykke kode, som ligger i en fil, og som kan brukes i en annen fil.

Når vi gjør jobber med et stort kode-prosjekt blir det mye kode. Dersom all denne koden havner i samme fil blir det veldig uoversiktlig. En måte å løse dette problemet er å spre koden utover flere filer. De ulike filene kaller vi moduler.

I en fil/modul skriver vi gjerne flere funksjoner. Vi kan velge i denne filen hvilke funksjoner som skal kunne hentes utenfra.

I en annen fil henter vi inn de funksjonene som ble gjort tilgjengelig. Vi kan nå bruke disse funksjonene, men vi trenger ikke rote til filen vår med all koden, den ligger lagret i en annen fil.

## Eksempel:

module.js

```
function printTotalLength(string1,string2) {  
  length = string1.length+string2.length  
  console.log(length)  
}  
exports.printTotalLength = printTotalLength
```

main.js

```
let funksjoner = require('./module.js')  
funksjoner.printTotalLength("Hans", "Hansen")
```

## To typer moduler

Node støtter bruk av to forskjellige moduler; ES6 moduler og CommonJS moduler.

I filen package.json kan en legge inn hvilken modul type js filer skal bli behandlet som . Ligger det "type": "module" i package.json blir js filene behandlet som es6 moduler. Motsatt, dersom det står "type": "commonjs" blir js filene behandlet som common js moduler.

Dersom en bruke filendelsene .mjs (es6 module) og .cjs (common JS module) styrer filendelsen hvilken modultype filene blir filene behandlet som.

## ES6 modules

module.mjs:

```
function sayHi(name) {  
  console.log("Hi", name, "im a ES6 Module")  
}  
export {sayHi}
```

main.mjs

```
import * as funksjoner from './module.mjs'  
  
funksjoner.sayHi("Borge")
```

## Common JS modules

module.cjs:

```
function sayHi(name) {  
  console.log("Hi", name, ", I'm a CommonJS Module")  
}  
exports.sayHi = sayHi
```

main.cjs

```
let funksjoner = require('./module.cjs')  
funksjoner.sayHi("Borge")
```

## Bruk av common js i en es6 module

module.cjs:

```
function sayHi(name) {  
  console.log("Hi", name, ", I'm a CommonJS Module")  
}  
exports.sayHi = sayHi
```

main.mjs

```
import funksjoner from "./module.cjs"  
  
funksjoner.sayHi("Borge")
```

# Node Package Manager, npm

Med Node Package Manager (npm) kan man gjenbruke kode som andre har skrevet før. Når en jobber med et prosjekt, er det nyttig å bare skrive koden som er helt spesiell for akkurat det prosjektet, og unngå å bruke tid og ressurser på å skrive kode som allerede finnes.

## Installasjon

npm er allerede installert sammen med node, så ingenting ekstra trengs å installeres. For å sjekke at programmet finnes på maskinen kan man kjøre følgende kommando:

```
npm - v
```

## Initialisere prosjekt der npm skal brukes

For å sette opp et prosjekt til å bruke npm pakker man kommandoen "npm init". Dette må gjøres i mappen til prosjektet. Enkleste måten å vite at man er i rett mappe er å åpne mappen med "Open Folder" i Visual Studio, for deretter å åpne terminalen.

```
npm init
```

Du kan trykke enter-tasten for å bruke standardverdier på alle instillingene.

Du kan nå observere at du har fått nye filer i mappen din.

## Installere en pakke

Første pakken vi installerer heter chalk. Den gjør at vi kan få fargefull output..

Vi installerer denne med kommandoen "npm install chalk"

```
npm install chalk
```

Observerer nå en ny mappe med kode i. Filene i denne mappen skal ikke redigeres på.

## Bruke pakken

For å bruke akkurat denne pakken må du gjøre følgende:

1. Legge til "type": "module", i filen package.json. Se bilde til venstre.
2. Legge til følgende kode på toppen:

```
import chalk from 'chalk';
```

```
{ } package.json > { } scripts
1  {
2    "type": "module",
3    "name": "leksjoner",
4    "version": "1.0.0",
5    "description": ""
```

3. Legge til denne koden et sted:

```
console.log(chalk.green.bold.inverse("Hello!"))
```

### **NB!**

Noen pakker skal ikke brukes med "import" kodeordet, men heller kodeordet require. Når "require" pakker brukes skal man heller ikke ha "type": "module" i package.json filen. Man kan ikke bruke import og require i samme fil.

## Nodemon

Med nodemon starter programmet ditt automatisk på ny, hver gang du trykker lagrer. Du slipper med andre ord å starte programmet på ny i terminalen hver gang du har gjort endringer.

Installer nodemon ved å kjøre følgende kommando:

```
npm install nodemon -g
```

-g gjør at installasjonen blir global, det vil si at den installeres slik at den kan brukes i alle node prosjektene dine. Package.json, og package-lock.json blir ikke endret.

### **Får du følgende feil?**

-> C:\Users\\*\*\*\*\AppData\Roaming\npm\nodemon.ps1 is not digitally signed error:

Fiks: Slett filen C:\Users\\*\*\*\*\AppData\Roaming\npm\nodemon.ps1

Alterniv løsning: Run: powershell as admin, and "Set-ExecutionPolicy Unrestricted

Du kan nå kjøre programmet ditt med kommandoen nodemon, istedenfor kommandoen node.

Viss du skal kjøre filen app.js, blir da kommandoen du skal bruke:

```
nodemon app.js
```

Nodemon starter på ny hver gang en .js fil lagres. Vi kan endre hvilke filtyper nodemon skal overvåke for nye endringer med. Senere i kurset skal vi blant annet jobbe med .ejs filer. Kommandoen vi da kan gjøre er følgende:

```
nodemon app.js -e js,ejs
```

# Typiske problem med moduler:

//Typiske problem:

## **js filen ligger feil sted**

Viss filen du kjører ligger på i feil mappe vil ting ikke fungere. Filen skal vanligvis ligge i en mappe som har mappen "node\_modules" inni seg.

## **js filen har feil fil-endelse**

-Filen du skal kjøre må være en .js fil.



# Webserver med Express

## Hva er express?

<http://expressjs.com/>

Express er en samling kode (npm modul) som tilbyr funksjonalitet som brukes i web og mobile applikasjoner. Vi kommer blant annet til å se hvordan express gjør at vi kan sende en webside til brukeren når brukeren ber om det. Vi vil også se på hvordan express gjør at brukeren kan sende data inn til node programmet vårt.

## Hvordan installere, og starte det opp?

Viss vi begynner på et helt nytt prosjekt, som skal bruke express, trenger vi først å initialisere npm, og dermed få laget en fil kalt package.json. Sørg for å være i korrekt mappe i terminalen, før du kjører følgende kommando:

```
npm init -y
```

(-y gjør at standard svar blir brukt på alle spørsmålene en ellers ville fått)

Vi skal nå ha en fil kalt package.json i mappen vår, og vi kan installere express.

```
npm install express
```

**Vi lager så en mappe kalt server-code.** Her skal vi legge server-koden vår. Inni serverCode legger vi hoved-filen for prosjektet vårt som vi lkaller **server-app.js**. Her legger vi inn følgende kode for å starte opp express.

```
//Hente inn modulen
const express = require('express')

//Starte en express applikasjon, og lagre den i app
const app = express()
```

```

  node_modules
  server-code
  JS server-app.js
  {} package-lock.json
  {} package.json
```

# Hvordan sette opp en rute?

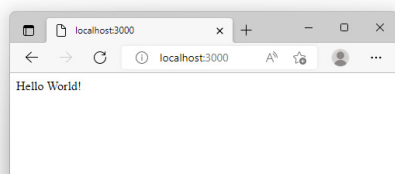
Vi kaller en nettside på express-webserveren din en rute/route.

## Rute til top-mappen

Vi kan først sette opp ruten for når du ikke skriver inn noen mappenavn eller filnavn i det hele tatt. Altså, dersom du bare skriver inn domenet, for eksempel bt.no

Vi lager først funksjonen som skal kjøre, når noen spør etter hoved-nettsiden vår. Etterpå sender vi denne inn som en callback funksjon ved hjelp av funksjonen get.

```
//Callback funksjon for når noen åpner root-mappen
//på web-serveren vår
function rootRoute(request, response) {
  response.send("Hello World!")
}
//Sende callback funksjonen inn i Express, slik at
//funksjonen blir trigget når noen skriver inn kunn
//domenet vårt
app.get('', rootRoute)
```



Vi trenger til slutt å starte express opp, slik at den lytter til innkomende forespørsler/requests.

```
app.listen(3000, () => {
  console.log('Server is up on port 3000')
})
```

Vi har nå laget en web-applikasjon. Vi skal nå starte programmet, vi åpner da **terminal** og skriver følgende:

```
node .\server-code\serverApp.js
```

Vi kan nå skrive inn følgende i nettleseren:

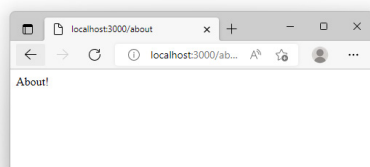
http://localhost:3000/

og skal få opp en nettside med teksten "Hello World"

## Eksempel på rute til adressen /about

Her er kode for å få opp en nettside på adressen localhost:3000/about

```
function aboutRoute(request, response) {
  response.send("About")
}
app.get('/about', aboutRoute)
```



Når du nå åpner adressen http://localhost:3000/about skal du få opp en nettside med teksten About.

# Hvordan tilby statisk innhold

Statisk innhold er ting som alltid er likt, uansett om vi endrer data i for eksempel en database.

En nettside som facebook.com er ikke statisk for eksempel, denne endrer seg hele tiden når ny data blir lagt til.

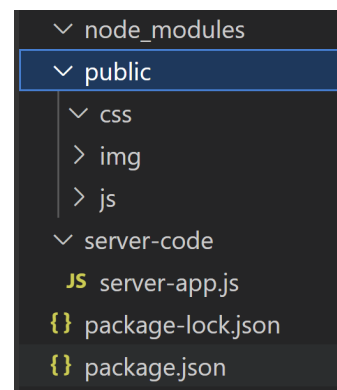
Det av innhold som skal være statisk legger vi i en mappe vi kaller public.

## Mappen public er fin å bruke til:

- Stilark
- Bilder
- Javascript kode
- Statiske Html-sider (Sider som ikke endres pga. dataendringer)

Anbefalt mappe-struktur i denne mappen:

- css -> Mappe for stilark
- img -> Mappe for bilder
- js -> Mappe for klient-side javascript



Vi skriver følgende kode, for å gjøre det som ligger i denne mappen, tilgjengelig på webserveren vår.

```
const path = require('path')
const publicDirectoryPath = path.join(__dirname, "../public")
app.use(express.static(publicDirectoryPath))
```

(På linje 2 i koden over skriver vi ../public. De to prikkene .. er fordi vi må gå opp en mappe fra der koden ligger (server-app.js), for å finne mappen public.)

## Eksempel på bruk av public-mappen

Vi ønsker å lage html-siden hjelp.html som bruker stilarket styles.css

Vi legger hjelp.html i public mappen og styles.css i css mappen.

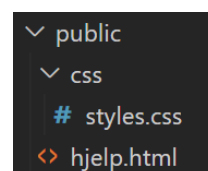
For å linke til stilarket fra html-filen får vi denne koden:

```
<link rel="stylesheet" href="css/styles.css">
```

Vi starter så express-appen med følgende kode

```
node .\server-code\serverApp.js
```

Du skal nå kunne åpne følgende nettside i nettleseren:  
<http://localhost:3000/hjelp.html>



# Input ved hjelp Skjema

Vi kan ha et skjema på nettsiden, og få data fra skjema inn i node-appen våre. Koden og fremgangsmåten som vises her, krever at du allerede har en [fungerende express applikasjon](#).

For at express skal forstå skjemaene våre, må vi sette den til å bruke en spesifikk standard. Det gjør vi slik.

```
app.use(express.urlencoded({ extended: true }));
```

Vi lager så et skjema, for eksempel dette:

Beskjed:

```
<form action="/sendInn" method="POST">
  <label> Beskjed: </label>
  <input type="text" name="message" required>
  <button>Send beskjed</button>
</form>
```

Vi lager så en funksjon(også kalt handler) som skal kjøre på serveren, når inndata fra skjemaet kommer inn på adressen /sendInn.

```
//Funksjon (handler) for å håndtere et skjema
function formHandler(request, response){
  console.log("Innkommende body: " + request.body)
  console.log("Input-feltet 'message': "+request.body.message)
  //Sender innsendereen videre til topp-domenet vårt
  response.redirect("/")
}
```

Vi knytter så funksjonen/handleren opp med post-adressen /sendInn.

```
app.post('/sendInn', formHandler)
```

```
Innkommende body: [object Object]
Input-feltet 'message': Hei på deg
```

Du kan nå teste skjemaet, og se at det du skriver der, blir logget av handleren til console.

En videre utvikling av dette, vil være å lagre det som kommer inn i en fil, eller i en database.

# Flere Skjema muligheter

```
<form action="/sendInn" method="post">
```

```
<label for="navn">Navn:</label>
<input type="text" name="navn" id="navn">
<br>
```

```
<label for="epost">Epost:</label>
<input type="email" name="epost" id="epost">
<br>
```

```
<label for="fodt">Dato:</label>
<input type="date" name="dato" id="dato">
<br>
```

```
<label for="gave">Velkomstgave:</label><br>
<input type="radio" name="gave" value="Radio">Radio
<input type="radio" name="gave" value="TV">TV
<input type="radio" name="gave" value="PC">PC
<br>
```

```
<label for="valg">Valg:</label>
<br>
<input type="checkbox" name="valg[]" value="Laks">Laks
<input type="checkbox" name="valg[]" value="Grøt">Grøt
<input type="checkbox" name="valg[]" value="Pasta">Pasta
<br>
```

```
<button> Send inn </button>
```

```
</form>
```

Navn: Epost: Dato: Velkomstgave:  
☒ Radio ☐ TV ☐ PC

Radio button:  
Man kan bare trykke på  
en ting. Det du velger  
blir lagret i variabel med  
navnet gave.

Valg:  
☒ Laks ☒ Grøt ☐ Pasta

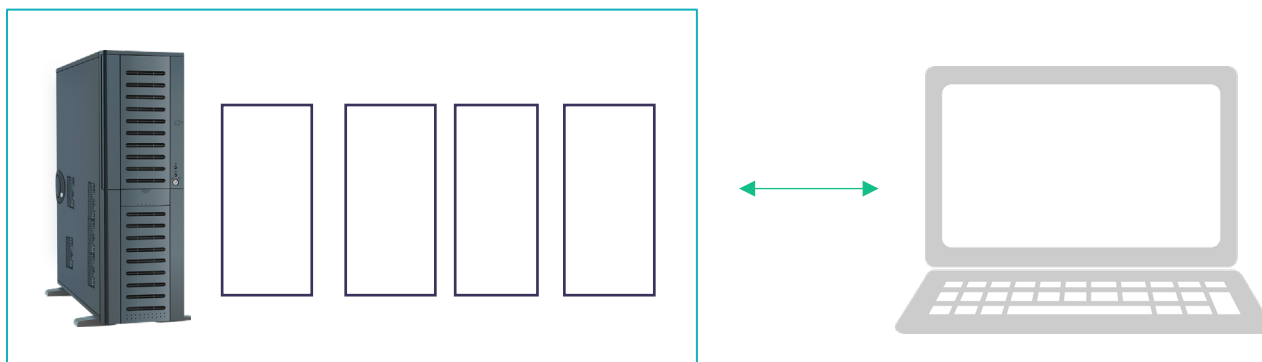
Checkbox:  
Her kan man velge flere.  
På grunn av [] blir det du  
velger sendt inn som en  
tabell med navnet valg.

# Server Side Rendering (SSR)

Med Server Side Rendering menes å sette sammen html-filen på serveren, før den sendes til klienten. Vi vil se på hvordan vi kan lage dynamiske websider, ved å sette data fra variablene våre, inn sammen med html-kode.

Vi kan lage dynamisk innhold ved å skrive kode i callback funksjonene som vi sendte til `app.get()` slik som forklart under kapittelet "[Hvordan sette opp en rute?](#)".

Når man lager større nettsider blir dette veldig rotete. Vi kan istedenfor bruke noe som kalles en template engine. Det finnes mange ulike template engines. To populære er Ejs og Handlebars. Ejs har mye funksjonalitet, men ser kanskje litt mer rotete ut en handlebar. Vi vil først vise hvordan en bruker Handlebars (hbs).



Server Side Rendering

Html og for eksempel data fra databaser settes sammen til en ferdig nettside på serveren, før denne sendes til klienten.

# SSR med Handlebars Template Engine

Handlebars er ikke laget for node.js, men ved hjelp av npm pakken hbs, blir handelbars også tilgjengelig i node.js.

```
npm install hbs
```

I filen app.js stiller vi inn express til å bruke hbs som template engine med denne kommandoen:

```
app.set('view engine', 'hbs')
```

Vi får senere bruk for å stille inn hbs, vi må derfor inkludere hbs på toppen av app.js slik:

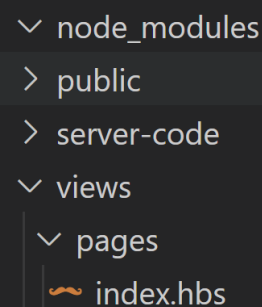
```
const hbs = require('hbs')
```

## Hvor vi lagrer våre dynamiske sider

Vi lager så en mappe der alle nettside som skal være dynamiske skal plasseres. Mappen kaller vi views. I denne mappen lager vi en mappe som heter pages. Her plasserer vi alle de dynamiske nettsidene vi skal ha.

Vi stiller inn express, slik at den vet om denne mappen slik:

```
const viewsPath = path.join(__dirname, '../views/pages')
app.set('views', viewsPath)
```



```
node_modules
public
server-code
views
  pages
    index.hbs
```

## Sende en variabel inn på de dynamiske sidene

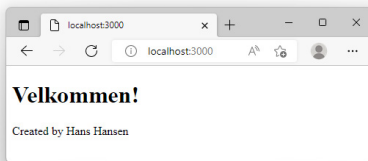
Vi ønsker å sende innholdet i en variabel, inn i en dynamisk nettside.

Vi lager filen index.hbs inni mappen pages, og legger inn følgende innhold:

```
<!DOCTYPE html>
<html>
<head>
</head>

<body>
  <h1> {{ title }} </h1>
  <p>Created by {{ author }}</p>
</body>
</html>
```

Vi endrer så på callback funksjonen vi laget for ruten til top-mappen på webserveren vår, slik at den blir slik:



```
function rootRoute(request, response) {  
  response.render('index.hbs', {  
    title: "Velkommen!",  
    author: "Hans Hansen"  
  })  
}  
app.get('/', rootRoute)
```

Vi har altså endret fra å bruke funksjonen `response.send` til funksjonen `response.render`. Første parameter i `render` er hvilken av filene i `pages` mappen som skal vises. Neste parameter er et objekt som inneholder den dynamiske dataen som skal brukes på nettsiden.

Viss vi nå kjører koden vår, og går inn på nettsiden `localhost:3000`, skal vi få opp en nettside med tittelen `Velkommen!` og teksten `"Created by Hans Hansen"`

### Sende en tabell inn på de dynamiske sidene

Vi ønsker å sende en liste med data inn på nettsiden vår. Listen vi vil sende inn er denne:

Interests:

- cars
- piano
- dance

Vi gjør dette ved å først legge til tabellen i objektet som sendes til `render`-funksjonen.

```
function rootRoute(request, response) {  
  response.render('pages/index', {  
    title: "Velkommen!",  
    author: "Hans Hansen",  
    interests: ["cars", "piano", "dance"]  
  })  
}  
app.get('/', rootRoute)
```

Vi legger så til følgende i filen `index.hbs` i mappen `pages`.

```
<ul>  
  {{#each interests}}  
  <li> {{this}} </li>  
  {{/each}}  
</ul>
```



```

  ✓ node_modules
  > public
  > server-code
  ✓ views
    ✓ pages
      🍷 index.hbs
    ✓ partials
      🍷 footer.hbs
      🍷 head-tag.hbs
      🍷 header.hbs
  {} package-lock.json
  {} package.json

```

## Gjenbruk av HTML-kode

Når vi lager et nettsted, vil mange av nettsidene dele html-kode. Her er eksempel på deler som ofte er lik:

- Det som står i head-taggen
- Det første i body-taggen. (Meny o.l). Headeren med andre ord.
- Det siste i body-taggen. (Kontaktinfo o.l). Footeren med andre ord.

For å slippe å skrive lik kode flere ganger, lager vi egne filer, for html-koden som skal vises flere steder. Disse filene plasserer vi i en mappe vi kaller partials, som ligger inni mappen views.

Vi må fortelle hbs, hvor denne mappen er. Det gjør vi slik:

```
const hbs = require('hbs') //Viss du ikke har lagt denne til enda
const partialsPath = path.join(__dirname, '../views/partial')
hbs.registerPartials(partialsPath)
```

(Husk at du må ha med const path = require('path') for å bruke path)

Vi begynner med å lage filene head.hbs, header.hbs og footer.hbs. Vi fyller disse filene med følgende innhold.

### head-tag.hbs:

```
<title> My awesome site</title>
```

### header.hbs:

```
<hr> <h1> {{ title }} </h1> <hr>
```

### footer.hbs:

```
<hr> <p>Created by {{ author }}</p> <hr>
```

Når vi har sider liggende i partials mappen, kan vi inkludere dette i en annen side med koden:

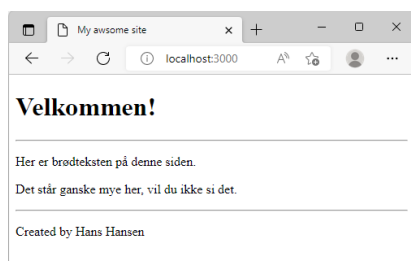
```
{{ >FILNAVN }}
```

Vi redigerer index.hbs nå til å se slik ut:

```
<!DOCTYPE html>
<html>
<head>
  {{>head-tag }}
</head>

<body>
  <header>
    {{>header }}
  </header>
  <p> Her er brødteksten på denne siden.</p>
  <p> Det står ganske mye her, vil du ikke si det.</p>
  <footer>
    {{>footer }}
  </footer>
</body>

</html>
```





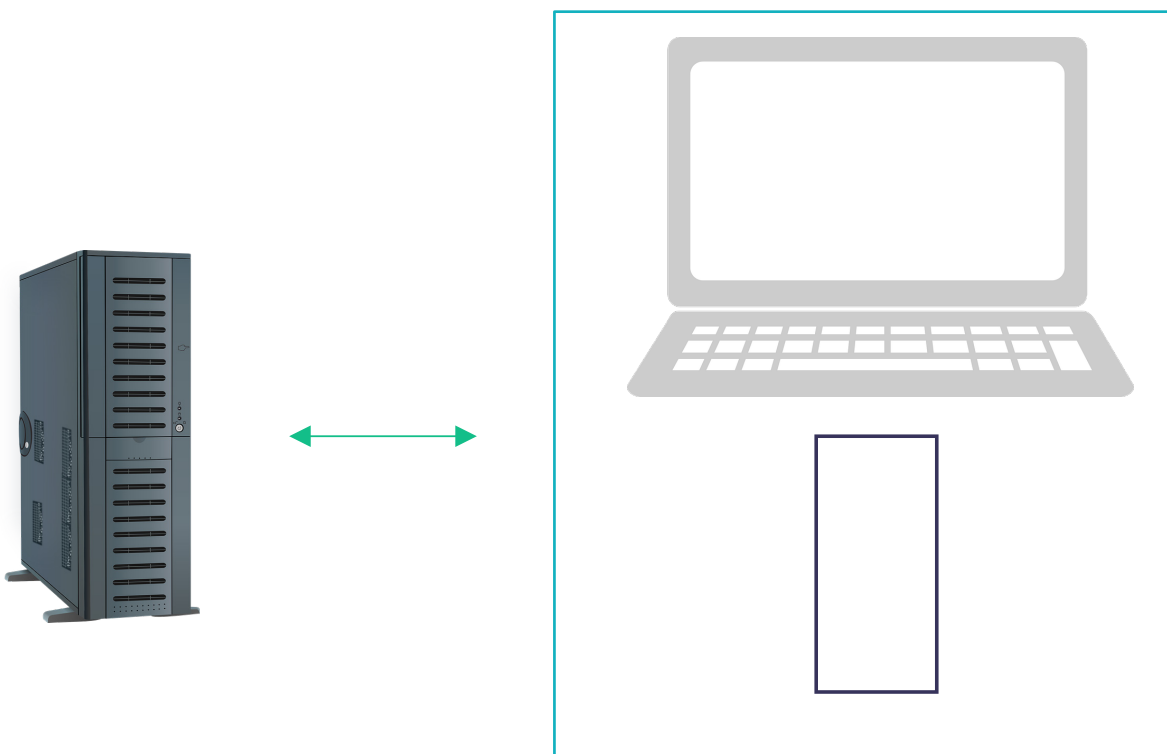






# Client Side Rendering

I dette kapitlet vil vi se på hvordan vi kan sende data fra node applikasjonen vår, til kode i nettleseren. Koden i nettleseren vil så bruke dataen til å bygge en nettside, som vises for brukeren. Vi vil først se på javascript funksjoner i nettleseren som er nyttige for å få dette til.



Client Side Rendering:  
Data (tabeller/objekter) sendes over nettverket, og settes sammen med html for å lage en nettside på klienten

# Hvor legger vi javascript-koden

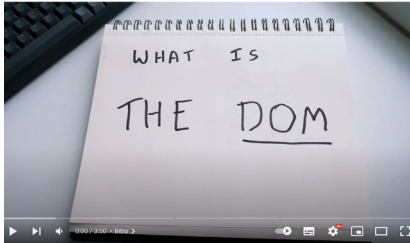
Vi kan skrive koden sammen med html-filen, da må den ligge inni en `<script>` tag

```
<script> alert("Kode kjørt i script tag") </scrip>
```

Vi kan også skrive koden i en egen fil. Da legger vi inn denne koden i head-delen.

```
<script src="/js/script.js" defer></script>
```

defer gjør at koden ikke kjører fører siden er ferdig lastet ned



[Film som forklarer hva DOM er](#)

# DOM - Document Object Model

Når dataen sendes til nettleseren fra serveren, ønsker vi å bruke denne til å endre html og css på en htmlside. I Javascript får vi tilgang til et objekt som heter document. Når vi endrer på dette, endres html-en og css-en på siden.

Her er kodesnutter vi ofte får bruk for:

Et

## Hente et html-element

```
// Henter en div, med attributten id="minDiv"  
// og lagrer denne i en variabel.  
let minDiv = document.getElementById("minDiv")
```

## Endre teksten inni et html-element

```
// Setter teksten inni minDiv til "Ny tekst"  
minDiv.innerText = "Ny tekst"
```

## Endre css-en til et html-element

```
//Endrer css-egenskaper til et element  
minDiv.style.color = "red"
```

## Lage et nytt html-element

```
//Lager et nytt p-element og setter teksten inni  
//til "Nytt avsnitt"  
let nyttElement = document.createElement("p")  
nyttElement.innerText = "Nytt avsnitt"
```

## Legge et html-element inni et annet

```
//Legger et element, inn i et annet.  
//Her blir en <p> tag lagt inni en <div> tag  
minDiv.appendChild(nyttElement)
```



# Hendelser

Viss vi har en knapp som skal få javascript kode til å kjøre, kan vi gjøre det slik

```
<button id="button">Klikk her</button>
```

Vi kan så skrive følgende kode, for å få en funksjon til å kjøre når noen klikker på knappen

```
let button = document.getElementById("button")
button.addEventListener("click", function () {
  alert("Du trykket på knappen")
})
```

# Datalagring

## Fil-lagring

Fil lagring er en teknikk som brukes for å lagre data på en hard-disk eller annen lagringsenhet. I en node.js-applikasjon kan det være nyttig å bruke fil lagring for å lagre informasjon som trengs for å huske brukerinnstillinger, logge hendelser, eller for å lagre data som er blitt generert av applikasjonen.

## Databasen Sqlite og Grafisk brukergrensesnitt

En database er et system for lagring og organisering av data, og er et viktig verktøy for mange webapplikasjoner.

Når en webapplikasjon kjøres på en node.js-server, kan den bruke en database til å lagre og hente informasjon som er nødvendig for å fungere, for eksempel brukerinformasjon eller data som er blitt lagt inn av brukerne.

Vi vil her se på bruk av databasen Sqlite. Dette er en liten database som ikke krever noe konfigurasjon for å brukes. Dette gjør det enkelt å komme i gang med å lagre data i en strukturert form, uten å måtte gjøre mye forarbeid eller forstå altfor mye om databaser.



### Opprette en database med tabeller

Vi vil bruke et program som heter sqlite Studio for å opprette databaser og tabeller med et grafisk brukergrensesnitt.

Programmet kan [lastes ned her](#).

ÅpneSQLite Studio og lag en ny database. Du kan for eksempel gjøre dette ved å høyreklikke i panelet til høyre, og velge Database -> Add database. Plasser databasen i prosjekt-mappen din.

Høyreklikk på den nye databasen og velg "Create Table". Skriv så inn tabell-navnet, og dobbelklikk i "kolonne-vinduet" for å legge til en ny kolonne.

Når du legger til kolonner kan du skrive inn navn, velge datatype og om det skal være primærnøkkel, fremmednøkkel. Under valg-knappen kan du stille inn til Auto Increment. Merk og valget NotNULL som gjør at det blir obligatorisk med verdi i kolonnen.

Her er eksempel med tabellen "medlemmer" som har kolonnene id, fornavn, etternavn og e-post:

Structure		Data	Constraints	Indexes	Triggers	DDL									
database2		Table name: medlemmer				<input type="checkbox"/> WITHOUT ROWID					<input type="checkbox"/> STRICT				
	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value					
1	id	INTEGER								NULL					
2	fornavn	TEXT								NULL					
3	etternavn	TEXT								NULL					
4	e-post	TEXT								NULL					

Husk å trykke på den grønne "hake-knappen" for at tabellen og kolonnene du har laget skal bli lagret i databasen.

## Skrive inn data

Når du har åpnet en tabell, kan du klikke på "Data", for så å klikke på den grønne "pluss-knappen" for å legge til data.

Structure

Data

Constraints

Indexes

Triggers

DDL

Grid view

Form view

1

	id	fornavn	etternavn	e-post
1	NULL	Hans	Birgers	h@bir.no
2	NULL	Birger	Hansen	b@han.no

Husk å trykke på den grønne "hake-knappen" for at dataen du har skrevet inn skal bli lagret i databasen.

# Databasen Sqlite og modul for koding i node.js

En modul for å skrive kode som kommuniserer med sqlite databasen vår er better-sqlite3. Denne er blant annet anbefalt fordi den er enkel i bruk.

Vi installerer modulen slik:

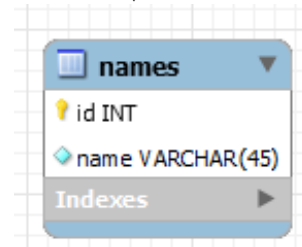
```
npm install better-sqlite3
```

For å koble til database-filen database.db skriver vi denne koden

```
const sqlite3 = require('better-sqlite3')
const db = sqlite3('database.db', {verbose: console.log})
```

Viss ikke filen database.db finnes, blir en ny fil laget. Vi skriver verbose... for at logg av hva som skjer skal skrives til console.

Videre viser vi eksempel på hvordan vi henter, setter inn, sletter og oppdaterer data i en database ved hjelp av modulen better-sqlite3. I dette eksempelet antar vi at vi har en database med en tabell, names:



## Hente alle radene i names (all)

```
const sql = db.prepare('SELECT * FROM names')
const rows = sql.all()

for (const row of rows)
  console.log(row)
```

## Hente en spesiell rad i names (get)

```
const sql = db.prepare('SELECT * FROM names WHERE id = ?')
const firstResult = sql.get(1)
console.log(firstResult)
```

Her henter vi kun raden som har id=1

## Legge til en rad

```
const sql = db.prepare('INSERT INTO names (name) VALUES (?)');  
  
const info = sql.run("Birger")  
console.log("Amount changes done: " + info.changes)  
console.log("lastInsertRowID: " + info.lastInsertRowid)
```

Her setter vi inn en rad, der kolonnen name får verdien Birger

## Slette en rad

```
const sql = db.prepare("DELETE FROM names WHERE id=(?)");  
  
const info = sql.run("7")  
console.log("Amount changes done: " + info.changes)  
console.log("lastInsertRowID: " + info.lastInsertRowid)
```

Her sletter vi raden som har id=7

## Oppdatere en rad





# Dokumentasjon

En viktig del av dokumentasjonen til det du utvikler, er å kommentere koden din på en god måte.

En relevant måte å gjøre dette på er å bruke oppmerkingsspråket kalt jsdocs.

Dersom du bruker denne standarden vil for eksempel Visual Studio Code forstå dokumentasjonen din, og vise denne eksempelvis når du eller andre bruker funksjonene du har lagret.

Når du bruker denne standarden finnes det også verktøy som gjør at du enkelt kan generere html-kode som dokumenterer all koden din.

Under vises noen eksempler på hvordan funksjoner skal dokumenteres etter jsdocs standarden:

```
/**
 * Legger til en karrakter i karrakteroversikten
 * @param {number} grade En tallkarrakter. Skal være
heltall mellom 1 og 6. IV ikke støttet
 */
function addGrade(grade) {
    grades.push(grade)
}
```

```
/**
 * Henter siste karrakter som er lagret
 * @returns {number} Returnerer siste karrakter som
er lagret
 */
function getLast() {
    return(grades[grades.length-1])
}
```

I eksemplene over, er datatypen til paramtereret til funksjonen og returverdien av typen number. Dersom det er tekst vil en skrive string istedenfor number her.

<https://jsdoc.app/>



# Koding 2

## Funksjoner som variabler

I Javascript kan man lagre en funksjon som en variabel, og kjøre denne med å bruke variabelnavnet og (). Se eksempel 1a under.

Man kan og sende funksjoner som parameter inn i andre funksjoner, se eksempel 1b under.

Eksempel:

Output:

Yes

Output:

Yes

Yes

```
//Eksempel 1a:
let printYes = function () {
  console.log("Yes")
}

console.log("Kjører funksjon lagret i printYes en gang:")
printYes()

//Eksempel 1b:
let runTwice = function(functionToRunTwice) {
  functionToRunTwice()
  functionToRunTwice()
}
console.log("Kjører funksjon lagret i printYes to ganger:")
runTwice(printYes)
```

I eksempel 1a blir en funksjon lagret i variabelen printYes, før den kjøres under.

I eksempel 1b blir det laget en funksjon som skal motta en funksjon som paramter. Funksjonen som kommer inn som parameter blir kjørt to ganger. På sluttten av eksempelet sender vi funksjonen printYes inn i funksjonen runTwice. printYes blir derfor kjørt to ganger.

## Sortering av objekter i array



Viss vi skal sortere et objekt i en array, kan vi ikke bare skrive arraynavn.sort(). sort vet ikke hvilke verdier i objektet vi skal sortere etter først.

Viss vi lager en funksjon som klarer å bestemme rekkefølgen på bare to objekter, kan vi sende denne funksjonen inn i sort funksjonen. På den måten kan vi sortere på objekter.

Denne funksjonen skal ta inn to objekter, a og b.

- Viss a skal være først, skal den returnere -1.
- Viss b skal være først skal den returnere 1.

- Viss det ikke er noe forskjell på dem, sorteringsmessig, skal det returneres 0.

For å forstå hvordan vi gjør dette, viser vi først på et eksempel der vi ser på hvordan vi kan finne rekkefølgen på de to tekstene "Hund og Ape".

```
console.log("Hund" < "Ape")
```

Output:

false

Vi sjekker her om Hund "er mindre"/"kommer før i alfabetet". Vi får false til svar, siden Ape skal komme først.

```
console.log("Hund" > "Ape")
```

Output:

true

Her sjekker vi om Hund kommer etter Ape. Vi får true, siden dette stemmer.

Vi kan bruke koden fra Hund og Ape eksempel når vi skal lage vår sammenligningsfunksjon for to objekter. La oss tenke at vi har følgende objekter i tabellen vår som vi ønsker sortert på etternavn.

```
{
  firstname: "Truls",
  lastName: "Trulsen",
  age: 97
}
```

Vi kan lage følgende funksjon, for å sammenligne to objekter på etternavn:

```
function sortLastName(first, second) {
  const firstLastName = first.lastName.toLowerCase()
  const secondLastName = second.lastName.toLowerCase()

  if (firstLastName < secondLastName) {
    return -1
  } else if (firstLastName > secondLastName) {
    return 1
  } else {
    return 0
  }
}
```

Vi kan nå sende denne funksjonen til sort, for å sortere en array med denne typen objekter.

```
personer.sort(sortLastName)
```

Vi kan nå sjekke med console.log at tabellen vår er sortert korrekt.

## Søke i en array

Viss man har en tabell/array, ønsker man ofte å søke for å finne en spesiell "ting" i tabellen.

### Eksempel med for-løkke:

Vi tenker oss at vi har en array med dyr.

En enkel måte å søke i arrayet, er bare å bruke en for løkke slik:

```
let dyrTabell = ["Hest", "Ku", "Ape"]
let searchItem = "Ku"

for(let i=0;i<dyrTabell.length;i++) {
  if (dyrTabell[i] === searchItem) {
    console.log("Funnet dyret,", dyrTabell[i], "i tabellen")
  }
}
```

### Eksempel med find-funksjonen

Alternativt kan man bruke find-metoden.

Når man skal bruke find-funksjonen trenger man å sende inn en funksjon som tester en ting i tabellen opp mot det en søker etter. Parameteret i denne funksjonen er en ting i tabellen.

Viss vi for eksempel søker etter Ape, trenger vi at funksjon returnerer true dersom parameteret er lik "Ape", ellers false.

Under lager vi en array med dyr, en variabel som inneholder det vi søker etter, og en funksjon som sjekker det vi søker etter opp mot en ting i tabellen.

```
let dyrTabell = ["Hest", "Ku", "Ape"]
let searchItem = "Ape"
function checkExist(dyr) {
  if (dyr === searchItem) {
    return true
  }
  return false
}
```

Vi kan nå sende denne funksjonen inn i find funksjonen, for å se om "Ape" finnes i listen.

```
const funnet = dyrTabell.find(checkExist)
console.log(funnet)
```

Viss vi finner "Ape" i tabellen, blir "Ape" lagret i variabelen funnet. Viss vi ikke finner "Ape", blir ingenting lagret i variabelen, og console.log vil printe undefined.

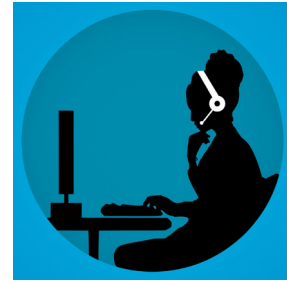
## Callback



Viss du for eksempel ringer Elkjøp for å klage på et kjøleskap, kan det være lang kø. Elkjøp kan da tilby deg å ringe deg tilbake (call back) så fort det blir din tur. Med andre ord: Når en hendelse inntreffer, hendelsen at det blir din tur, da iverksettes operasjon ring tilbake til deg..

I programmering har vi ofte at en funksjon skal kjøre, når en spesiell hendelse inntreffer. Vi kaller funksjonen for en callback funksjon.

Under vises et eksempel der vi lager en nedtelling på 5 sekund. Vi har hendelsen at 5-sekunders nedtellingen er ferdig, når denne hendelsen inntreffer kjører vi callback funksjonen `congrats`.



```
function congrats() {  
  console.log("Congrats, you managed to wait 5 seconds!");  
}  
  
setTimeout(congrats, 5000);
```

Output:

Congrats, you...

## Klasser og objekter

En god [gjennomgang av klasser finner du på ndla](#).

# Lesing og Skrivning til fil

Vi bruker en modul kalt fs for å lese og endre filer på datamaskinen. fs står for filesystem.

```
const fs = require('fs')
```

1. Først hvordan vi leser tekst fra en fil, for så å skrive teksten tilbake til fil.
2. Etterpå et eksempel på hvordan vi kan bruke json for å lagre data (Tabeller og objekter) på samme måten.

## 1 - Skrive tekst til fil

```
tekst.txt
1  Tekst som lagres i fil
```

Koden til høyre lagrer tekst i en fil kalt tekst.txt

```
let tekst = "Tekst som lagres i fil"
fs.writeFileSync('./tekst.txt', tekst, "utf-8")
```

## 1 - Hente tekst fra fil

```
Output:
Tekst som lagres i fil
```

```
let tekst = fs.readFileSync("./tekst.txt", "utf-8")
console.log(tekstfraFil)
```

## 2 - Skrive variabel/array/objekt til fil

```
{} data.json > ...
1  {"tabell1":["testdata1","testdata2"]}
```

Koden til høyre, lager en json-tekst av objektet data, og lagrer teksten i filen data.json

```
let data = {
  "tabell1" : ["testdata1", "testdata2"]
}
// Vi gjør nå om dataen til en tekst-streng, som følger
// Json-standarden:
const dataInJsonFormat = JSON.stringify(data)
fs.writeFileSync('./data.json', dataInJsonFormat, "utf-8")
```

## 2 - Lese variabel/array/objekt fra fil

```
Output:
```

```
{ tabell1: [ 'testdata1', 'testdata2' ] }
```

```
// Lese fra fil
const dataInJsonFormat = fs.readFileSync("./data.json", "utf-8")
// Vi gjør nå om tekst-strengen dataInJsonFormat, om til
// vanlig "variabel/array/objekt":
let data = JSON.parse(dataInJsonFormat)
console.log(data)
```

# Arrow Functions

Vi har tidligere sett at vi kan lage funksjoner slikl

```
let sayHello = function() {  
  console.log("Hello")  
}
```

Av og til ønsker vi å lage en slik funksjon, uten å gi den et navn, siden den skal sendes direkte inn som et parameter i en annen funksjon. Funksjonen vi da lager kalles Anonymous. Vi kan lage et eksempel, at programmet vårt skal si hallo etter to sekund, vi bruker da setTimeout() funksjonen.

```
setTimeout(  
  function () {  
    console.log("Hello")  
  },  
  2000)
```

I Javascript kan vi forenkle dette enda mer, ved å lage en arrow funciton. Da bytter vi ut function () med () =>

```
setTimeout( () => {  
  console.log("Hello")  
},  
2000)
```

Når koden i funksjonen bare er på en linje, kan vi forenkle enda mer.

```
setTimeout( () => console.log("Hello"), 2000)
```

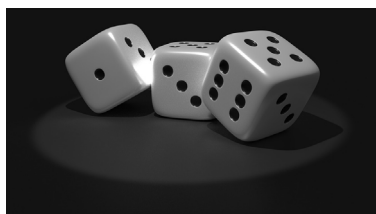
NB! Funksjonene over er ikke identisk. Det er forskjeller, men de er ikke viktig for oss enda.







# Oppgaver



## Oppgave 1 - Karractersystem

Lag en modul som gjør det mulig å enkelt holde styr på alle karakterene dine.

I felleskap gjør vi det slik at modulen kan:

Legge til en karakter

Skrive ut alle karakterene dine

Vi bruker modulen i en annen fil. Tester at det fungerer å legge til karakterer, og å skrive alle ut.

Du finner koden for dette [her](#):

### Del 1 - Grunnleggende

Du skal så utvide karakter-modulen slik at den kan gjøre følgende:

- Skriv ut funksjonen skal også skriver ut hvor mange karakterer som er lagret.
- Kunne fjerne karakteren som ble sist lagt til (undo)
- Kunne sortere karakterene fra størst til minst, og skrive ut resultatet
- Kunne sortere karakterene fra minst til størst, og skrive ut resultatet
- Skrive ut den beste karakteren
- Skrive ut den dårligste karakteren
- Kunne fjerne den største karakteren
- Kunne fjerne den minste karakteren
- Regne ut gjennomsnittet av alle karakterene
- Hente den største karakteren (tips: return)
- Hente den minste karaktere (tips: return)

### Del 2 - Videre utvikling

- Legge til kontroll på data som lagres, slik at modulen tåler ugyldig input.
- utvid, slik at ikke bare tall lagres, men navn med karakter. Tips: Lagre objekter, istedenfor bare tall.

Når du har fått til at karakter og navn henger sammen, kan du lage funksjoner for følgende:

- Endre karakter, når en oppgir et fagnavn
- Slette karakter, når en oppgir et fagnavn
- Lag en grafisk fremstilling av karakterene dine, som printes til console.
- Legg på farger, for å skille gode karakterer, fra ikke fullt så gode. Tips: Bruk chalk

## Oppgave 2 - BlackJack

Lag et program som gjør at en kan spille blackjack.

Du trenger blant annet følgende:

- En funksjon for å lage en tabell som inneholder en kortstokk. Du kan lage det som en enkel tabell slik: [Hjerner 1, Hjerner 2 osv..] eller du kan lage hver celle som objekter. Bruk iallefall løkker for å lage tabellen.
- En funksjon for å trekke et visst antall kort fra kortstokken og plassere over i en spillers eller dealers hånd. Husk at kortene som blir trukket, skal forsvinne fra kortstokken...
- En funksjon for å beregne verdien av kortene en spiller/dealer har på hånden.
- En funksjon for å gjennomføre et spill.  
Anbefaler modulen prompt-sync for å få brukeren til å gi input til programmet.  
I spill funksjonen kan du sette opp kortstokk, og hånden til spiller og dealer. Videre må du sjekke om dealer får blackjack allerede ved starten. Vider kan du lage en løkke som går helt til noen vinner. For hver ny omgang, får du valget om du vil trekke flere kort, eller om du ikke vil spille lenger.

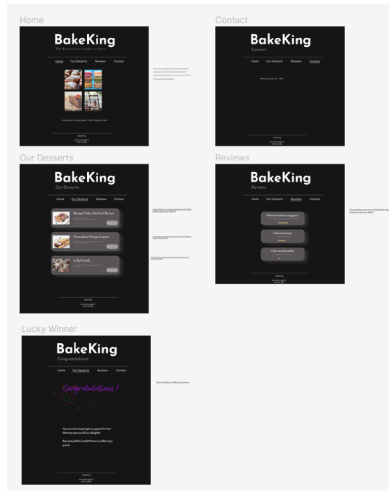


Dersom du har lært om klasser og objekter, kan du gjerne prøve å programmere med objektorientert.

## Oppgave 3 - BakeKing

I denne oppgaven skal du bygge nettstedet BakeKing ved hjelp av Node og Express

[Prototype til nettstedet finner du i Figma her. Her er det og noen css-tips.](#)



### Nettstedet skal inneholde følgende nettsider:

- Fremside
- Dessert side
- Reviews side
- Kontakt side
- Du er vinner side

### Data på nettstedet skal lagres i en json fil

Eksempel på data som lagres i filen:

- Tittelen på siden
- Adressen i footeren
- Telefonnummer i footeren
- Reviewseene
- Dessertene

### Du er vinner siden:

- Denne siden skal lastes i 50% av tilfellene fremsiden åpnes

### Videre utfordringer:

- Legg til små animasjoner. For eksempel at linkene vokser litt når du tar musepeker over dem.
- Gjør det mulig å legge inn reviews
- Lage admin-side for de ansatte på Bakeking, slik at de kan godkjenne reviews, og legge inn nye desserter.
- Legg inn en animasjon fra lottie-files i bakgrunnen av hele siden når du vinner.
- 



## Oppgave 4 - Rollespill

På NDLA finnes en oppgave som går på å kode klasser som kunne blitt brukt i et rollespill. Du finner oppgaven [her](#).



## Oppgave 5 - Fotball

Lag kode som kan simulere en fotbalkamp.  
Koden skal kunne lagre spillere og lag.

En spiller skal blant annet lagres med navn, og treffsikkerhet.  
En spiller skal kunne gjøre et skudd, og programmet skal kunne finne ut om spilleren treffer eller ikke.

Et lag skal lagres med tabell over alle spillerene, navnet på laget, og navnet på hjemme stadion. Laget skal kunne arrangere en kamp, og estimere hvem som vinner i møte med en motstander.



Her er et forslag til hvordan du kan organisere koden:

### Klasse for spiller

- Med metode for skudd

### Klasse for lag

- Med metode for kamp

Til slutt skal du kunne teste at en spiller skyter, og se om han scorer eller ikke.

Du skal og kunne sette opp to lag, og la dem spille mot hverandre, og se hvem som vinner.

Du kan utvide programmet videre for å simulere på en bedre måte. Kanskje det er flere variabler du kan lagre om spillere/lag for å gjøre det mer realistisk?

## Oppgave 6 - Vanlige funksjoner

a) Lag en funksjon du kaller sayHello, som tar inn parameteret navn, og skriver ut teksten "Hello " etterfulgt av navnet.

Med andre ord:

Dersom en kjører denne koden:

```
sayHello(Birger)
```

skal en få denne outputen i console:

```
Hello Birger
```

b) Lag en helt vanlig funksjon som du kaller getFirstFive. Funksjonen skal ta inn en tabell som parameter, og returnere de første fem elementene.

Med andre ord:

Dersom en kjører denne koden:

```
let input = ["A","B","C","D","E","F","G","H"]
let output = getFirstFive(Birger)
console.log(output)
```

skal en få denne outputen i console:

```
["A","B","C","D","E"]
```



() => { ... }

## Oppgave 7 - Arrow Funksjon

I denne oppgaven skal du lage en funksjon som skriver ut "Hallo Hallo"

Lag funksjonen som en anonym [arrow funksjon](#) ved å lage den som et argument til parametrene til en setTimeout funksjon. Kjør den med fem sekunder delay, ved bruk av timeout.

## Oppgave 8 - Skrivning og lesing fil

a) Skriv følgende [data til fil](#) som json-data.

```
let person = {  
  name: "Birger"  
  age: 23  
}
```



b) Les dataen du skrev til fil ved å kode i en annen fil. Print navn og alder til console. Outputen skal se slik ut:

```
Navn: "Birger"  
Alder: 23
```

## Oppgave 9 - Guess the number

I denne oppgaven skal du lage et spill der brukeren kan gjette på et tilfeldig tall.

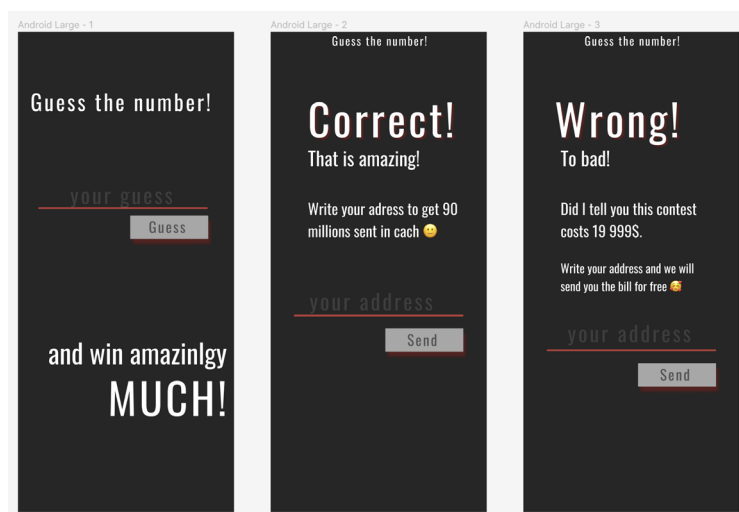
### Ved korrekt svar:

Dersom brukeren gjetter korrekt, skal han få en gratulasjonsmelding, og mulighet til å sende inn adressen sin.

### Ved feil svar:

Dersom brukeren gjetter feil, skal han få en melding om at det var feil, og mulighet til å sende inn adressen sin.

[Her er en Figma fil med layout du kan bruke.](#)



Ekstra utfordring:

Lag programmet i en express-aplikasjon, og gjør det slik at adressene folk sender inn, lagres i to forskjellige filer:

- regningAdresser.json
- premieAdresser.json

# Oppgave 11 - Input/Output Mal til Express

**Add content here**  
Message

Skjemaet trenger bare å ha et input felt.

Denne oppgaven går ut på at du får prøvd deg på å sette sammen et nettsted som har:

- Et skjema for lagring av data. Dataen skal lagres i en fil på serveren
- Visning av data som er lagret på fil på serveren

Å ha laget et slikt nettsted, og kunne bruke det som utgangspunkt for andre oppgaver, vil være en stor fordel.

Du kan selv velge hva "hensikten" til nettstedet skal være. Her er noen forslag:

- Quotes bibliotek
- Chattroom

Dataen som lagres trenger bare å være enkle tekst-strenger. Du trenger med andre ord bare et felt i skjemaet ditt. På nettstedet skal det finnes to ulike utvalg av dataene. For eksempel kan du ha en nettside der all data vises, og en annen der bare de fem siste dataene vises.



## Her er teknologier som er relevant å bruke:

(Se i innholdsfortegnelsen for hjelp)

- Node
- Express
- Handlebars
- Innput til Express ved hjelp av html-form
- HTML
- CSS
- Client Side Javascript (Spesielt for å få til utfordring under)

## Utfordring:

- Bruk CSS til å gjøre siden flott, både på desktop og telfon
- Utvid skjemaet til flere felter, og lagre objekter istedenfor tekst-strenger.
- Bruk Client Side Javascript til å få til at data oppdateres automatisk, når andre brukere legger til data. Hint: Fetch og setTimeout
- Legg til en søk funksjon. Hint Query string.
- Bytt ut funksjonene som lages for rutene dine, med anonyme arrow funksjoner.
- Lag en modul for funksjonene som omhandler skriving og lesing fra fil. Et forslag er at funksjonene her kan brukes direkte inn i app.get metoden.





## Oppgave 10 - Påmeldingsapp



Se for deg at du skal arrangere en veldedighetsstream på skolen. Du vil gjerne vite hvem som skal være med, men er for kul til å bruke Google Forms.

Ta utgangspunkt i følgende github prosjekt:

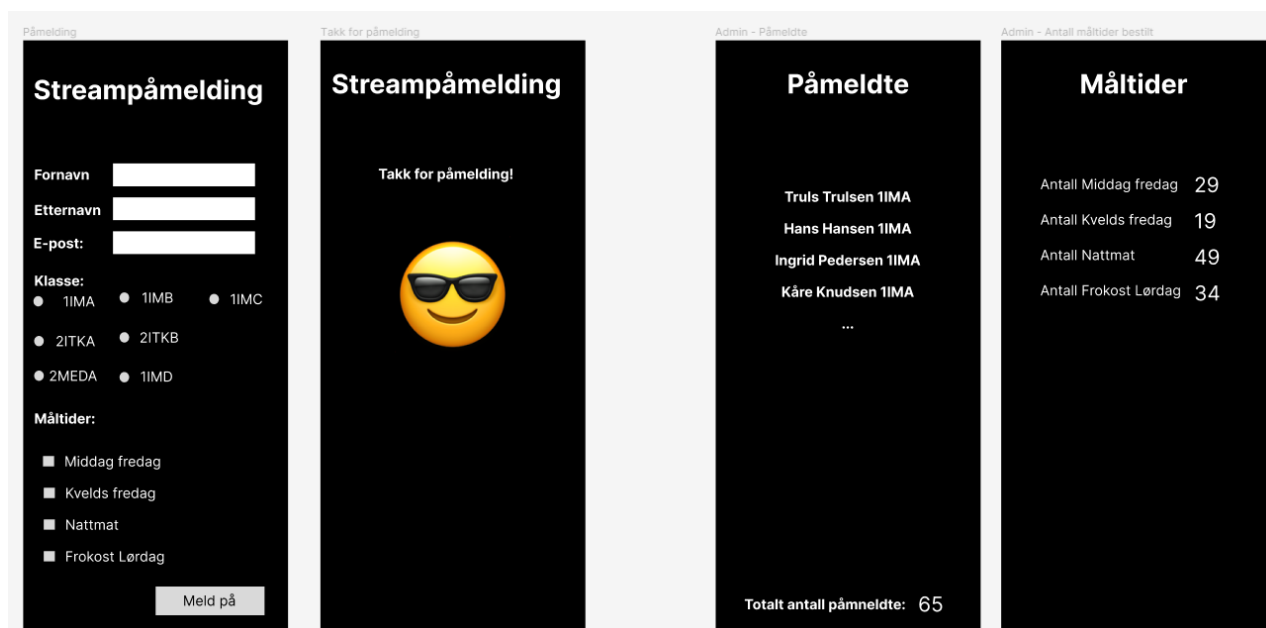
<https://github.com/boggarp/Utganspunkt-Express-og-Handlebars>

Lag appen. Du skal ha med:

- Et påmeldingsskjema
- Takk for påmelding-side
- Adminpanel med liste over alle påmeldte, og antall måltider bestilt.

Påmeldte lagres i en json-fil ved hjelp av modulen fileStorage som ligger i github prosjektet.

[Her er figma-skisse du kan bruke som utgangspunkt.](#)



Denne oppgaven er et eksempel på hva du skal kunne gjennomføre og forklare på heldagsprøven. På baksiden ser du mer om hvordan heldagsprøven blir.

# Om heldagsprøven 13. Desember

Det vi gjør idag blir en forsmak på hva vi gjør på heldagsprøven 13. Desember.

På heldagsprøven får du en lignende oppgave, og skal designe en løsning, og kunne forklare denne muntlig ved blant annet å gå gjennom koden din.



**Her er eksempel på spørsmål du kan måtte svare på:**

**Hva er forskjelle på node.js og Javascript i nettleseren?**

-> Vis og forklar kode du har laget som kjører på serveren..

-> Vis og forklar kode du har laget som kjører på klienten..

**Hva er en rute i express? Vis og forklar en du har laget.**

**Hva er forskjellen på get og post?**

Hvor bruker du disse to i applikasjonen din?

**Hvordan kan man få data inn fra en klient, via et skjema? Og så lagre dette. ..**

Vis i koden din og forklar.

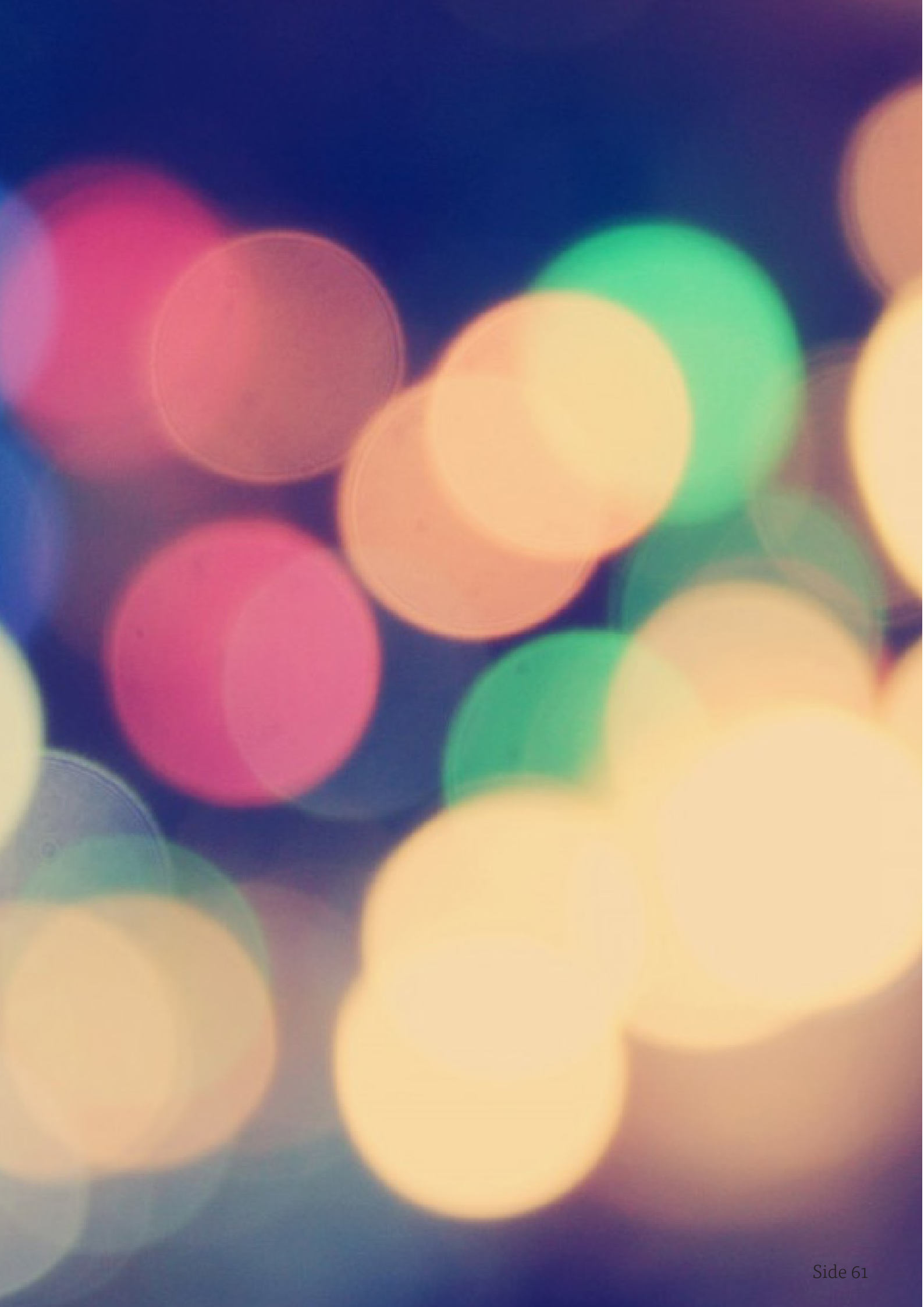
**Hvordan kan man vise data man har lagret på serveren, på en nettside.**

Vis i koden din og forklar.









# Obliger

## Oblig 1 - Filmsystem

I denne obligen skal du lage kode som gjør det mulig å holde oversikt over filmer. De første oppgavene trener deg i grunnleggende koding. De neste oppgavene trener deg i strukturere koden din ved hjelp av en javascript modul.

Hver oppgave skal lagres og leveres som en egen fil. Altså task1.js, task2.js osv.



### Task 1 - Variabler, console, array, for-løkke

#### a) Lag to tekst variabler.

Den første variabelen skal ha navnet programTitle. Her skal du lagre verdien "MyMovieApp, MMA".

Den andre variabelen skal ha navnet programDesc. Her skal du lagre verdien "App for keeping track off great movies"

Disse to variabelene skal lages slik at de ikke kan endres på et senere tidspunkt.

#### b) Skriv en melding til console

I meldingen skal innholdet i de to variablene fra oppgave a vises, sammen med litt passende tekst.

#### c) Lag en tall variabel

Lag en variabel som heter insertCounter. Denne skal få verdien 0. Denne variabelen skal vi senere få til å vise hvor mange filmer som noen gang har blitt lagt inn i programmet. Når vi sletter en film fra systemet, skal altså ikke denne reduseres.

#### d) Lag tabell (array)

Lag en tom tabell med navnet movies.

#### e) push metoden og øke variabel med 1

Bruk push metoden til å legge til fire film-navn i tabellen movies. Mellom hver gang du legger til en film, skal du øke verdien av variabelen insertCounter med 1.

#### f) Sorter

Bruk en passende array funksjon til å passe på at alle filmene er sortert alfabetisk i tabellen.

**g) Pop**

Fjern det siste elementet i tabellen, ved å bruke array-funksjonen pop.

**h) For løkke**

Bruk en for-løkke til å skrive ut navn på alle filmene  
Utskriften skal se slik ut (med andre filmnavn kanskje..):



```
Film 1 : Behind every closet lies a car  
Film 2 : Every secret has a bad knee  
Film 3 : The dark side of a frog
```

**i) Bruk av variabler**

Skriv ut en melding som viser hvor mange elementer som ligger i tabellen nå. Skriv også ut hvor mange filmer som noen gang har blitt registrert i systemet ved å bruke variabelen insertCounter. Du skal med andre ord få en utskrift som ser slik ut:

```
Amount movies stored in system: 3  
Amount movies ever entered into system: 4
```

**Husk:** Lagre filen nå som task1.js. Begynn så på task2.js. task2.js skal starte som en tom fil.

**Task 2 - Objekter**

(Denne oppgaven starter som en tom fil)

**a) Lag et objekt**

Lag et objekt med navnet film1, som inneholder følgende tre variabler:

- titel
- lengthMin
- rating

Gi variablene følgende verdier:

- The dark side of a frog
- 113
- 5.6

**b) Bruk et objekt**

Skriv ut til console informasjonen som ligger lagret i objektet.  
Utskriften skal se slik ut:



```
Titel: The dark side of a frog  
Length in minutes: 113  
Rating: 5.6
```

**c) Endre objekt**

Endre ratingen til 6.9. Skriv ut filmen på ny slik som i oppgave b.



**Husk:** Lagre filen nå som task2.js. Begynn så på task3.js.  
task3.js skal starte med koden fra task1.js.

### Task 3

I denne oppgaven skal du endre koden fra oppgave 1. Istedenfor å lagre kun filmnavn, skal du lagre objekter, som består av filmnavn, lengde i minutter, og karakter. Fjern koden for sortering og sletting av film.

#### a) Lage objekter

Endre koden der du bruker push metoden for å legge til filmnavn. Her skal du istedenfor lage fire objekter, og pushe dem inn i tabellen isteden.

#### b) Bruke objekter

Endre koden der film-infoen skrives ut. Utskriften skal bli slik:

```
Film 1 : The dark side of a frog, 113min, rating 5.6  
Film 2 : The elefant in the hallway, 96min, rating 6.7  
Film 3 : Behind every closet lies a car, 46min, rating 7  
Film 4 : Every secret has a bad knee, 97min, rating 2.9
```



**Husk:** Lagre filen nå som task3.js. Begynn så på task4.js.  
task4.js skal starte med koden fra task3.js.

### Task 4 - Funksjoner

I denne oppgaven skal du endre på koden fra Task 3.

#### a) Funksjon for innsetting

Lag en funksjon for å sette inn filmer. Velg passende parameter. Endre koden deretter slik at funksjonen brukes for å sette inn filmer i tabellen, istedenfor koden du har skrevet tidligere. Tips: Inni funksjonen kan du lage objektene, for så å putte dem inn i tabellen.

#### b) Funksjon for utskrift

Lag en funksjon for utskrift av filmene lagret i systemet, sammen med infoen om antall filmer i systemet, og antall filmer noen gang er registrert. Endre så koden slik at denne funksjonen brukes for å skrive ut info om filmene, istedenfor koden du har skrevet tidligere.

**Husk:** Lagre filen nå som task4.js.



## Task 5 - Modul

I denne oppgaven skal du endre koden fra task4, slik at koden spres utover to filer. Du skal med andre ord lage en modul, og bruke denne modulen i en annen fil.

Modulen skal du kalle movieRegister, og lagre i filen movieRegister.cjs

### a) - Lag modulen

Lag modulen movieRegister i filen movieRegister.cjs.

Modulen skal inneholde tabellen for lagring av filmer, og funksjonene for innsetting og utskrift. Eksporter funksjonene slik at de blir tilgjengelig i den andre filen.



### b) Bruk modulen

Lag filen app.js.

Hent inn modulen fra oppgave a.

Legg til filmene som du la inn i task4, og skriv dem ut på samme måte, bare ved bruk av modulen.

## Ekstra utfordringer

- Kommenter alle funksjonene ved hjelp av [JSDocs standarden](#).
- Legg til funksjon i modulen for å slette filmer basert på filmnavn.
- Legg til funksjon i modulen for å sortere filmene basert på tittel
- Legg til funksjon i modulen for å sortere filmene basert på karakter
- Legg til funksjon i modulen for å sortere filmene basert på lengde.
- Gjør det mulig å legge til rating på filmer, og at ratingen som vises er gjennomsnittet.
- Søk deg opp på klasser, og lag det slik at filmer, sjanger og rating blir egne klasser..
- Gjør det mulig at en film kan ha mange sjangere.
- Knytt deg opp mot IMDB/Rotten Tomato etc. apiet, for at rating hentes herfra.



## Oblig 2 - DOM

Se oppgave i Teams

## Oblig 3 - Express App

"Skjema for lagring"

"Vise data"

Mulige ting: Todo app, Eller noe annet fornuftig: (Som innvolverer lagring og henting av data...)

I denne obligen skal du lage et nettside som bruker følgende teknologi.

- Figma (Skisse av nettstedet)
- Node
- Express
- Handlebars, eller annen template engine
- Data fra json file i første omgang.
- CSS og HTML
- Javascript som kjører på klienten.

addEventListener - Legge til en funksjon, som kjører når en hendelse skjer. (Callbackfunksjon)

# Ekstra

# SSR med Ejs Template Engine

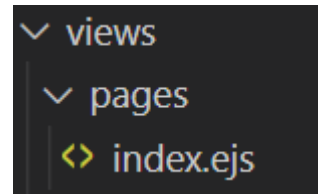
Ejs er et alternativ til Handlebars. Du trenger ikke bruke dette viss du er fornøyd med Handlebars.

Ejs installeres slik

```
npm install ejs
```

I filen app.js stiller vi inn express til å bruke ejs som template engine med denne kommandoen:

```
app.set('view engine', 'ejs')
```



## Hvor vi lagrer våre dynamiske sider

Vi lager så en mappe der alle nettside som skal være dynamiske skal plasseres. Mappen kaller vi views. I denne mappen lager vi en mappe som heter pages. Her plasserer vi alle de dynamiske nettsidene vi skal ha.

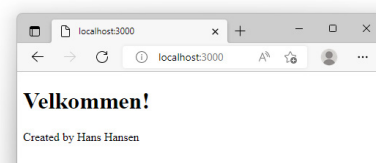
## Sende en variabel inn på de dynamiske sidene

Vi ønsker å sende innholdet i en variabel, inn i en dynamisk nettside.

Vi lager filen index.ejs inni mappen pages, og legger inn følgende innhold:

```
<!DOCTYPE html>
<html>
<head>
</head>

<body>
  <h1> <%= title %></h1>
  <p>Created by <%= author %></p>
</body>
</html>
```



Vi endrer så på callback funksjonen vi laget for ruten til top-mappen på webserveren vår, slik at den blir slik:

```
//Callback funksjon for når noen åpner root-mappen
//på web-serveren vår
function rootRoute(request, response) {
  response.render('pages/index', {
    title: "Velkommen!",
    author: "Hans Hansen"
  })
}
//Sende callback funksjonen inn i Express, knyttet
//til root-mappen ved hjelp av tom streng ''
app.get('', rootRoute)
```

Vi har altså endret fra å bruke funksjonen `response.send` til funksjonen `response.render`. Første parameter er hvilken av filene i `views` mappen som skal vises. Neste parameter er et objekt som inneholder den dynamiske dataen som skal brukes på nettsiden.

Viss vi nå kjører koden vår, og går inn på nettsiden `localhost:3000`, skal vi få opp en nettside med tittelen `Velkommen!` og teksten `"Created by Hans Hansen"`

## Sende en tabell inn på de dynamiske sidene

Vi ønsker å sende en liste med data inn på nettsiden vår. Listen vi vil sende inn er denne:

Interests:

- cars
- piano
- dance

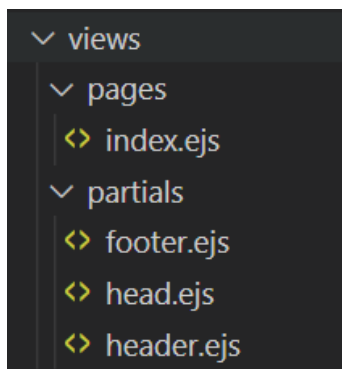
Vi gjør dette ved å først legge til tabellen i objektet som sendes til `render`-funksjonen.

```
function rootRoute(request, response) {
  response.render('pages/index', {
    title: "Velkommen!",
    author: "Hans Hansen",
    interests: ["cars", "piano", "dance"]
  })
}
app.get('/', rootRoute)
```

Vi legger så til en `for`-løkke i `index.ejs` i mappen `pages`.

Merk at taggene `<%= %>` fører til utskrift av data, mens taggene `<% %>` bare fører til at javascript kode blir kjørt.

```
<ul>
  <% for (let i=0;i<interests.length;i++) { %>
    <li> <%= interests[i] %> </li>
  <% } %>
</ul>
```



## Gjenbruk av HTML-kode

Når vi lager et nettsted, vil mange av nettsidene dele html-kode. Disse delene her, vil ofte være lik:

- Det som står i `head`-taggen
- Det første i `body`-taggen. (Meny o.l). Headeren med andre ord.
- Det siste i `body`-taggen. (Kontaktinfo o.l). Footeren med andre ord.

For å slippe å skrive lik kode flere ganger, lager vi egne filer, for html-koden som skal vises flere steder. Disse filene plasserer vi en mappe vi kaller `partials`, som ligger inni mappen `views`.

Vi begynner med å lage filene head.ejs, header.ejs og footer.ejs.

Vi fyller disse filene med følgende innhold.

head.ejs:

```
<title> My awesome site</title>
```

header.ejs:

```
<hr> <h1> <%=title %></h1> <hr>
```

footer.ejs:

```
<hr> <p>Created by <%=author %></p> <hr>
```

Når vi har sider liggende i partials mappen, kan vi inkludere dette i en annen side med koden:

```
<%- include('../partials/FILNAVN') %>
```

Vi redigerer index.ejs nå til å se slik ut:

```
<!DOCTYPE html>
<html>
<head>
  <%- include('../partials/head') %>
</head>

<body>
  <header>
    <%- include('../partials/header') %>
  </header>

  <p> Her er brødteksten på denne siden.</p>
  <p> Det står ganske mye her, vil du ikke si det.</p>

  <footer>
    <%- include('../partials/footer') %>
  </footer>

</body>
</html>
```

