

Arkitektur og underliggende teknologier

Svend Andreas Horgen og Mildrid Ljosland

07.02.2012

L restoffet er utviklet for faget LN393D Ajax

2. Arkitektur og underliggende teknologier

Resym : Vi skal n  g  i dybden p  de underliggende teknologiene bak Ajax. Vi starter med prinsippene for hvordan Ajax virker, og introduserer viktige teknologier: JavaScript, XHTML, CSS og DOM. Du vil i denne leksjonen f  se eksempel p  en tiln rmet ajax-l sning skrevet i DHTML, som i praksis er Ajax uten den asynkrone kommunikasjonen. Dersom du ikke har v rt s rlig borti JavaScript f r, s  les tilleggsleksjonen om JavaScript (egen leksjon).

2.1.	HVORDAN FUNGERER AJAX?.....	2
2.1.1.	Fors�k p� � definere Ajax.....	2
2.1.2.	Synkron vs asynkron kommunikasjon med tjeneren.....	4
2.1.3.	N�kkelel�teknologier i Ajax.....	6
2.2.	JAVASCRIPT – LIMET SOM HOLDER AJAX-TEKNOLOGIENE SAMMEN.....	7
2.2.1.	Kort om objekter i JavaScript.....	7
2.2.2.	Funksjoner i JavaScript.....	7
2.2.3.	Hendelser.....	9
2.2.4.	Syntaks for beslutninger, l�kker, matriser (arrays), operatorer++.....	9
2.2.5.	Eksterne JavaScript-filer.....	11
2.3.	DOCUMENT OBJECT MODEL (DOM).....	11
2.3.1.	Hva er DOM?.....	11
2.3.2.	Eksempel p� bruk av DOM i JavaScript.....	12
2.3.3.	HTML eller XHTML?.....	13
2.3.4.	Et lite eksempel p� DHTML.....	13
2.4.	KILDER.....	14

2.1. Hvordan fungerer Ajax?

Ajax som begrep og teknologi er nytt, men delteknologiene er velkjente og brukt gjennom en årrekke av webutviklere. Ajax er et samspill av teknologier. Det fordrer minst to ting for å lykkes: En må beherske de underliggende teknologiene, og en må forstå samspillet mellom disse. I denne leksjonen skal vi se på JavaScript, HTML/XHTML og DOM, og vise hvordan en dynamisk side fungerer. Neste leksjon går i dybden på den asynkrone kommunikasjonen som er så vital (og genial) for Ajax, og etter det ser vi på formater for å utveksle informasjon og bruk av CSS for å presentere informasjon.

2.1.1. Forsøk på å definere Ajax

Hva er egentlig Ajax? Vi kan tenke oss ulike innfallsvinkler for å forstå og definere Ajax:

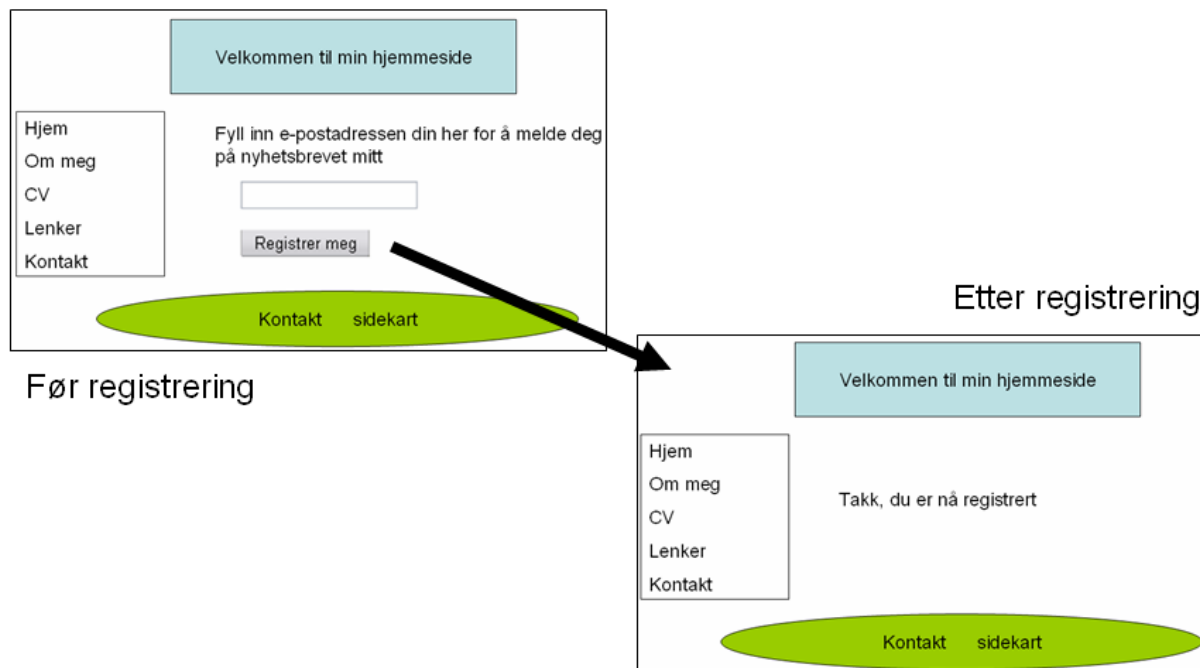
- Nettleseren inneholder en applikasjon, ikke innhold.
- Tjeneren sender over data, ikke innhold.
- Brukeropplevelsen innenfor applikasjonen kan være flytende og kontinuerlig.
- Ajax-koding krever disiplin og struktur.

Punktene over er kraftige i mening – de uttrykker forståelsen for hva Ajax i essens er, og gjennom å ha denne forståelsen kan vi også bedre forstå hvordan Ajax fungerer. Særlig de to første krever refleksjon. Her følger en kort forklaring.

Applikasjon, ikke innhold: Hva er forskjellen på innhold og en applikasjon? Innhold er det du ser, mens applikasjonen er det som lager innholdet. Innholdet kommer (helt eller delvis) fra, og styres av, Ajax-applikasjonen. Applikasjonen er altså sentral som innholdstilbyder. Det du ser på websiden er et resultat av JavaScript-koden som ligger bak og styrer.

Data, ikke innhold: På en tradisjonell webside hvor du kan klikke på noe, må hele siden lastes på nytt. Tjeneren leverer alt innholdet på nytt hver gang. Det som vises kan være nesten likt forrige side, men også helt forskjellig. Ofte vil det på et nettsted være nesten likt utseende på sidene som utgjør nettstedet. Det meste vil i så fall være unødvendig å laste inn på nytt, siden det bare er en liten del som er forandret fra forrige tilstand. Topptekst, bunntekst, menyer og kanskje andre elementer vil være de samme, mens bare en liten del av informasjonen skal byttes ut.

Informasjonen som skal byttes ut krever altså ikke at helt nytt innhold sendes, men at *nye data* erstatter de gamle dataene i et visst område eller på nye steder, med mye av det gamle innholdet inntakt. Dette er illustrert i Figur 1. Dersom tjeneren kan overlevere disse dataene uten å endre det gamle innholdet, er vi kommet langt. Mengden transport reduseres. Dette er mulig med asynkrone forespørsler, og Ajax-løsninger avhenger av slike for å fungere. Det som vises i nettleseren kan oppdateres uten at resten av siden lastes på nytt. For å få til dette må sidens objektmodell (DOM) manipuleres, og nettopp det kan gjøres med JavaScript. Vi skal i denne leksjonene se nærmere på både JavaScript og DOM, og legger i så måte et viktig grunnlag for videre programmering med Ajax.



Figur 1: Eksempel på hvor lite innhold som endres når knappen trykkes.

En rik brukeropplevelse: Tradisjonelt har web fokusert på visning av tekst, bilder, lenker og skjema (forms). Lenker (og bilder) kan klikkes. Skjema tillater brukeren å oppgi informasjon som kan sendes til tjeneren gjennom blant annet tekstbokser, nedtrekkslister, radioknapper og sjekkbokser. Det du ser på en slik webside, er et bilde av tiden, én av mange tilstander i din interaksjon med nettstedet. For å gå til neste tilstand, må du som regel klikke på en knapp/lenke for å sende informasjon til tjeneren, som så kan sende deg til neste tilstand (etter påfølgende ventetid mens siden lastes på nytt).

HTML-skjema tilbyr et tynt utvalg av brukergrensesnittkontrollere sammenliknet med en vanlig skrivebordsapplikasjon. Med Ajax er det mulig å implementere dra-og-dropp-funksjonalitet, samt introdusere en rekke mer avanserte grensesnittkomponenter. Det er likevel viktig å merke seg at det ikke er gitt at mer avanserte grensesnitt automatisk øker brukeropplevelsen.

Med Ajax er det som tidligere nevnt mulig å fjerne ventetiden, og sømløst hente og sy inn ny informasjon bak kulissene. Dette åpner for helt nye måter å gjennomføre utfylling av for eksempel skjema. I stedet for ett standardskjema, kan en tenke seg skjema som endres dynamisk avhengig av hva som skrives inn/velges underveis, eller Ajax-løsningen kan automatisk fylle inn implisitt informasjon. Skriver du et postnummer, kan for eksempel tilhørende poststed vises automatisk. Vet du ikke postnummeret, skjer det motsatte, og for å hjelpe med valg av poststed, kan antall alternativer vises i en dynamisk generert nedtrekksliste som oppdateres kontinuerlig mens du skriver. Brukeren får hjelp av applikasjonen, brukeren kan ha en mer flytende og kontinuerlig interaksjon med websiden enn før.

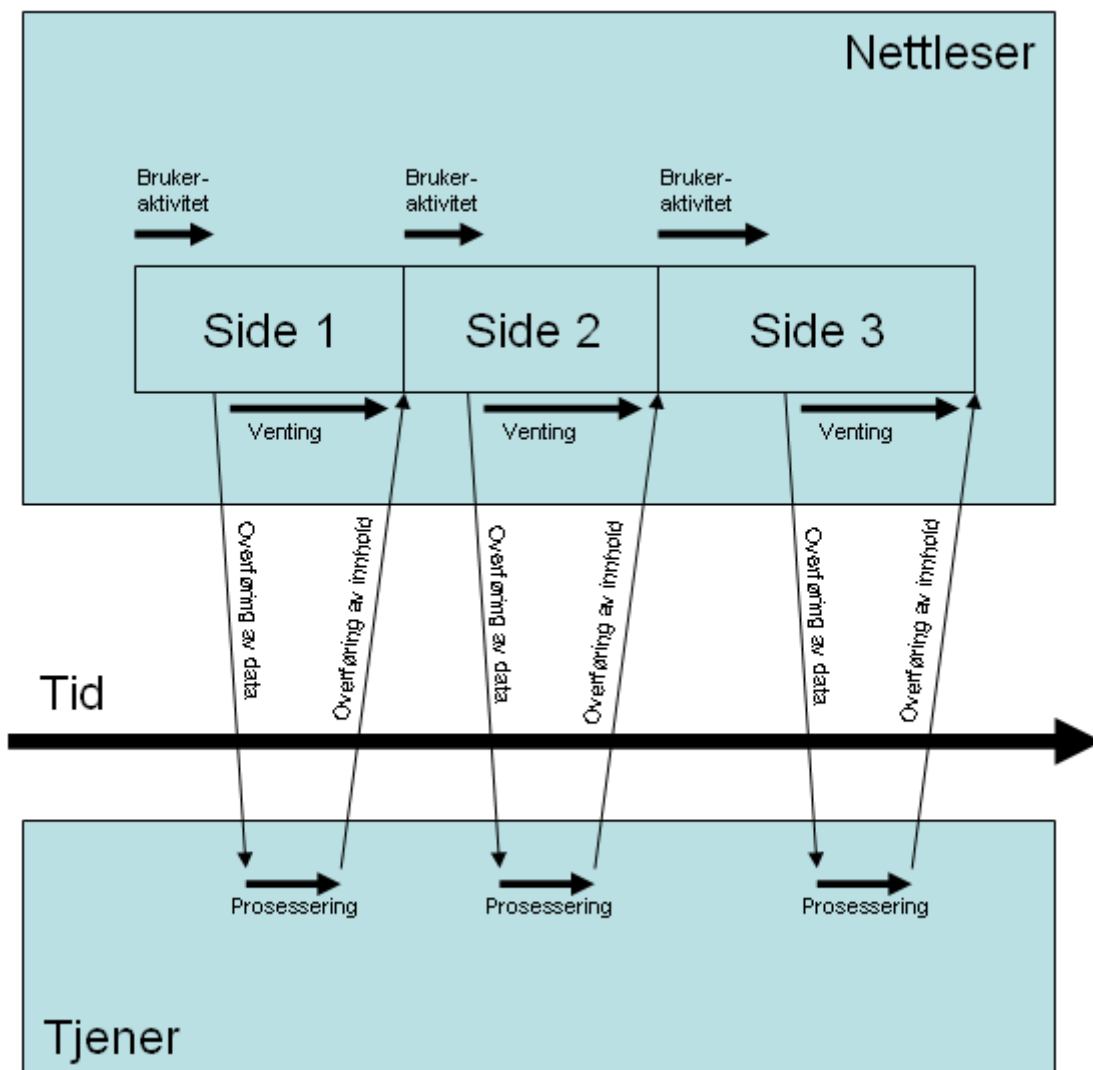
Disiplin og struktur: JavaScript anses ofte for å være et onde på web, siden mange websider bruker JavaScript for å åpne vinduer (popup) med reklame og liknende. Resultatet er at nyere nettlesere tilbyr å blokkere kjøring av script eller popup-vinduer. Mange utviklere anser JavaScript som et for enkelt scriptspråk til å kunne utføre profesjonell utvikling.

Google og andre store aktører har med sin satsning på Ajax generelt og JavaScript spesielt, fått utviklere verden over til å revurdere gamle fordommer mot språket.

JavaScript kan bygges inn i eksisterende websider gradvis. Utgangspunktet kan være en vanlig webside som JavaScriptifiseres jevnlig til å få stadig mer avanserte egenskaper. Det er krevende å utvikle gode Ajax-applikasjoner, uansett om en starter med dynamikk fra bunnen av eller tar utgangspunktet i eksisterende statiske sider. Struktur, en god praksis for koding og nøye planlegging er nødvendig på samme måte som i andre programmeringsspråk dersom en skal lage større Ajax-applikasjoner. Du må likevel ikke rømme bort allerede nå dersom du ikke anser deg som noe programmeringssess og synes dette høres skummelt ut! Gode Ajax-løsninger kan utvikles av alle, og ikke alle Ajax-løsninger trenger å være komplekse og velorganiserte. I dette kurset vil vi ta en gradvis tilnærming fra det enkle mot det mer avanserte.

2.1.2. Synkron vs asynkron kommunikasjon med tjeneren

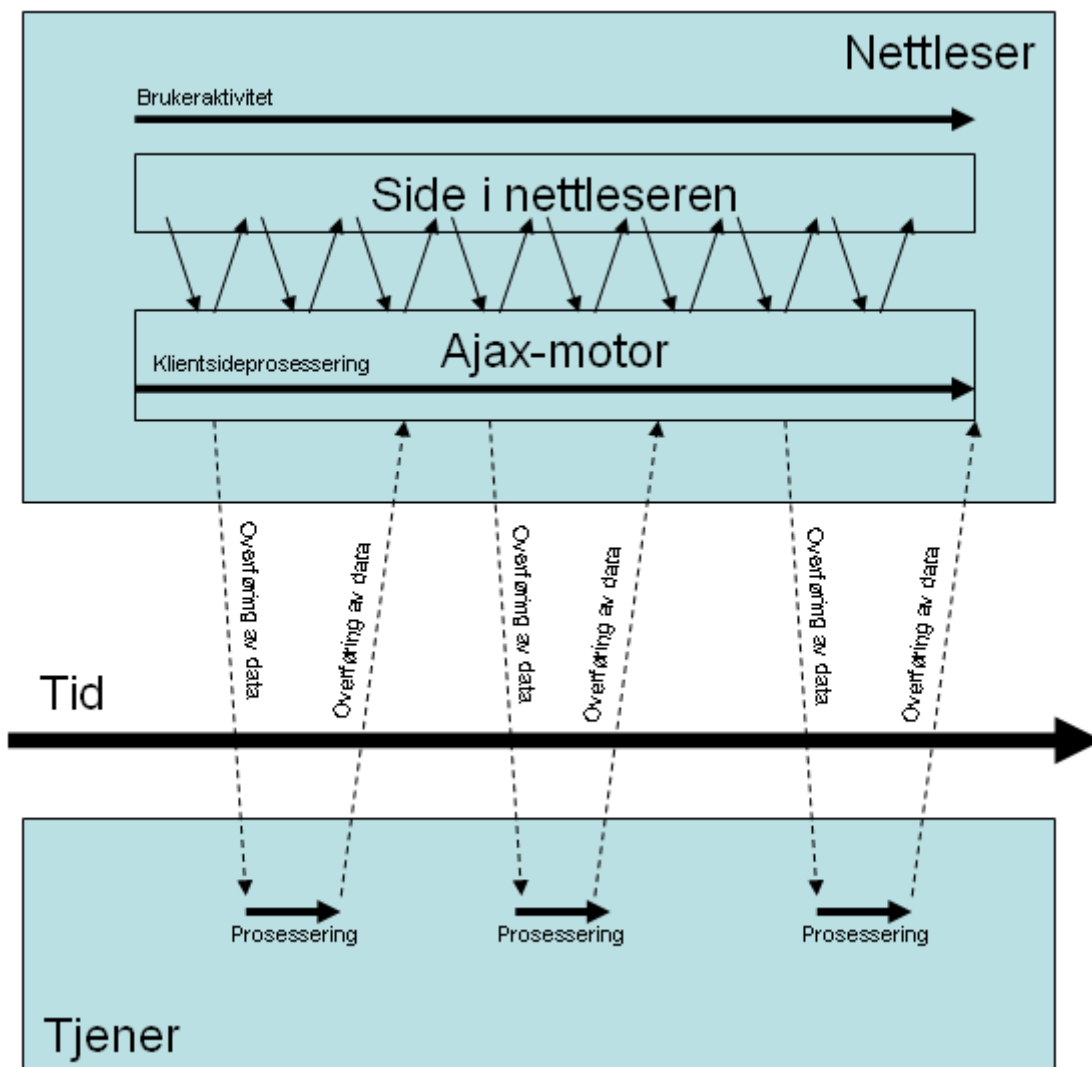
Til nå har vi snakket om at nøkkelen til suksess ligger i asynkron kommunikasjon med tjeneren. Hva innebærer egentlig dette?



Figur 2: Synkron kommunikasjon mellom klient og tjener.

I den tradisjonelle klient-tjener-modellen som ligger til grunn for tradisjonelle webløsninger, svarer tjeneren på forespørsel fra klienten. De fleste forespørsler er brukerinitierte, for eksempel ved at en lenke klikkes eller en skjema knapp trykkes. Mens en forespørsel pågår, må brukeren vente på neste side, uten å kunne gjøre noe i mellomtiden. Dette er illustrert i Figur 2 (modifisert utgave basert på illustrasjonene til Jesse James Garrett (<http://adaptivepath.com/publications/essays/archives/000385.php>)).

Dersom en Ajax-motor lastes i stedet til å begynne med, kan denne fungere som mellomledd mellom klient og tjener gjennom hele økten. Som vist i Figur 3, vil brukeren få tilnærmet ingen ventetid når dataoverføringen kan skje bak kulissene. Legg merke til at det blir opp til Ajax-motoren å bestemme hva som skal skje i ulike tilfeller. Av og til vil venting kanskje ikke være til å unngå på grunn av store datamengder som skal overføres eller liknende. I så fall kan Ajax-motoren vise et roterende timeglass eller liknende for å gjøre brukeren oppmerksom på at her er det transport av data på gang. Siden må uansett ikke lastes på nytt.



Figur 3: Asynkron kommunikasjon mellom klient og tjener.

Med Ajax-motoren mener vi den JavaScript-koden som du har laget og som blir mellomleddet mellom tjeneren og brukeren. Det er altså du som må programmere Ajax-

motoren. Dersom det fins kode for å håndtere klikk på et bilde, så utføres koden hver gang klikket skjer. Hvis ikke, skjer ingenting. All makt til programmereren!

2.1.3. Nøkkelelleteknologier i Ajax

Dersom du har holdt på med webutvikling tidligere, kjenner du kanskje til DHTML, eller dynamisk HTML. Med DHTML kan en bruke JavaScript og CSS til å manipulere utseendet på en allerede innlastet webside, vi sier at nettleserens DOM (Document Object Model) kan manipuleres dynamisk på klientsiden. Det nye med Ajax er den asynkrone kommunikasjonen med tjeneren. Dette tillegget (asynkronitet) gjør det mulig å lage en helt ny type applikasjoner på web.

Ajax er en samling av flere viktige teknologier:

- **HTML/XHTML:** Markeringsspråk som vanlige websider er skrevet i. Du kan få til det samme i XHTML som i HTML. Forskjellen er at XHTML har en strengere syntaks. Alle tagger må være i små bokstaver, attributter må ha anførselstegn rundt seg og alle elementer må avsluttes. Det vil si at `` ikke er gyldig XHTML-kode, mens `` er gyldig. En kortform for såkalte "tomme" elementer, er ``
- **JavaScript:** et scriptspråk som kan kjøres av nettleseren. Koden ligger sammen med vanlig HTML/XHTML-kode i taggen `<script>` (eller den lenkes inn fra en ekstern fil).
- **CSS:** Står for Cascading Style Sheets og er et språk som beskriver hvordan HTML/XHTML-kode (eller evt. andre markeringsspråk) skal presenteres. Det er lett å lage design som er konsistente over mange sider med CSS. Det var forresten nordmannen Håkon Wium Lie (Chief Technology Officer i Opera Software) som foreslo og stod i spissen for utviklingen av CSS-standard. Stilsett kan også endres i nettleseren ved hjelp av JavaScript, altså etter at siden er mottatt fra tjeneren.
- **DOM:** Står for Document Object Model og er en måte å representere strukturert data på. Strukturen kan defineres ved hjelp av XML, XHTML (som er HTML som følger XML-reglene) eller HTML/DHTML. JavaScript kan operere mot nodene i DOM-treet, for eksempel legge til, endre, slette, vise/skjule og liknende. Dermed er det mulig å endre innholdet på dokumentet etter at den er mottatt fra tjeneren.
- **XMLHttpRequest:** Et objekt som gjør asynkron kommunikasjon med tjeneren mulig. Data kan mottas i form av XML, men også som ren tekst og andre formater. Navnet er derfor ikke helt beskrivende for oppførselen.

Vi antar at HTML (evt. XHTML) er kjent fra før, og går nå kort gjennom basis JavaScript og DOM. Det forventes ikke at du behersker disse teknologiene fra før, men grunnleggende programmering bør du kjenne til (if, while, funksjoner). Det nye du må lære i denne leksjonen, er syntaksen til JavaScript og særegenhetene til språket. Du vil lære mye JavaScript utover i kurset, og alt du trenger blir gjennomgått underveis, så om du aldri har sett JavaScript før er et helt i orden. Vår første gjennomgang må bli kort og overordnet, siden hovedfokuset vårt i dette kurset er på Ajax (noe mer enn bare vanlig JavaScript).

Det er ikke nødvendig å bruke stilsett i Ajax-løsninger, men siden den grafiske presentasjonen i løsningen blir mye bedre med CSS enn uten, vil du ofte se CSS-kode i Ajax-applikasjoner i dette kurset. Du får en kort gjennomgang i denne leksjonen, med referanse til mer

informasjon på nett. Forkunnskaper i CSS er ikke noe krav i dette kurset, men en får penere løsninger ved å beherske CSS, så vi vil delvis komme inn på enkel CSS-bruk underveis.

Etter at vi har sett på JavaScript, CSS og DOM, er det mulig å lage DHTML-løsninger. DHTML står for Dynamisk HTML, og er i praksis det samme som Ajax, men uten den asynkrone kommunikasjonsbiten. En kan altså si at Ajax er DHTML kombinert med asynkron kommunikasjon med tjeneren. Din første ekte Ajax-løsning får du først se i og med gjennomgangen av XMLHttpRequest-objektet i neste leksjon: opprettelse, validitetssjekk, åpning, sending, mottak, metoder og egenskaper, for å nevne noe.

Her er en kort oppsummering av JavaScript. For en grundigere gjennomgang, se egen leksjon om JavaScript. Det anbefales at du leser denne før du fortsetter, med mindre du har kjennskap til JavaScript fra før.

2.2. JavaScript – limet som holder Ajax-teknologiene sammen

JavaScript er et *interpretert språk*, det betyr at koden tolkes linje for linje i motsetning til kompilerte språk. Det gir større feilmargin – noe av koden kan være feil, men feilmeldinger kommer ikke før den aktuelle setningen kjøres. Det er ikke nødvendig å deklarere variabler, og en trenger ikke tenke over datatyper, selv om JavaScript internt skiller ulike datatyper fra hverandre. En og samme variabel kan sågar veksle mellom ulike datatyper gjennom koden. Det er ikke nødvendig, men likevel lov, å bruke semikolon bak hver setning. Alt dette gjør terskelen for å programmere språket mindre, men språket har en rekke avanserte muligheter som dekker de fleste behov.

Det er opp til brukeren å slå av/på JavaScript, og med JavaScript avslått vil ikke våre Ajax-løsninger virke (vi kommer tilbake til håndtering av slike situasjoner senere i kurset). Det går også an å bruke andre scriptspråk enn JavaScript, for eksempel VBScript. Det språk som brukes må støttes av nettleseren. JavaScript er godt utbredt (implementert i de vanligste nettleserne) og stadig mer brukt av webutviklere.

2.2.1. Kort om objekter i JavaScript

Det fins en rekke innebygde objekter i JavaScript. Ett av disse er navigator. Et objekt kan ha metoder og egenskaper. Metodene gjør noe med objektet, mens egenskaper forteller noe om objektet.

For å finne ut hvilken nettleser som brukes, kan en bruke egenskapen `navigator.appName`. Versjon og flere detaljer om nettleseren, finnes i egenskapene `navigator.appVersion` og `navigator.userAgent`. Du vil se bruk av disse utover i kurset, blant annet for å håndtere forskjeller mellom ulike nettlesere slik at koden din kjører riktig uavhengig av nettlesertype.

Du kan også lage dine egne objekter med metoder og egenskaper. Dette kommer vi tilbake til senere i kurset.

2.2.2. Funksjoner i JavaScript

Det er mulig å lage egne funksjoner i JavaScript, og disse kan kalles opp som i alle andre språk. Funksjoner plasseres typisk i hodeseksjonen av et HTML-dokument, inne i en script-tag, slik:

```
<html>
<head><title>Første JavaScript</title>
<script language="javascript">
    function nettleser() {
        var leserenDin = navigator.appName;
        alert ("I dag bruker du nettleseren " + leserenDin);
    }
</script>
</head>
<body>
<h1>Hvilken nettleser bruker du?</h1>
<form>
    <input type="button" value="Klikk meg" onclick="nettleser()">
</form>
</body>
</html>
```

Når du har laget en funksjon, må den kalles opp. Det kan for eksempel gjøres basert på brukerinitierte hendelser. I skjemaet i forrige kodesnutt er det en knapp av type "button", ikke "submit". Når den klikkes, kalles funksjonen `nettleser()`. Siden denne er definert i hodeseksjonen, vet nettleseren at nå er det denne som skal kjøres.



Figur 4: Når knappen klikkes, kjøres funksjonen som skriver ut hvilken nettleser som brukes.

Hensikten med dette eksempelet er bare å vise hvordan en funksjon kalles opp. Du vil få se flere, og bedre eksempler på funksjoner etter hvert: Argumenter, generalisering, returverdier og så videre.

2.2.3. Hendelser

For å sette i gang forrige funksjon, brukte vi hendelsen onclick. Hver gang knappen klikkes, utføres det som eventuelt måtte stå som attributtverdi til hendelsen onclick til knappen. Det fins mange hendelser som nettleseren kan fange opp, de fleste er like selvforklarende som onclick (klikk med museknapp). Vi har hendelser som fanger opp aksjoner fra mus og tastatur, og vi har en rekke andre programvarestyrte hendelser. Typiske navn er: onclick, onabort, onfocus, onerror, og liknende. Les mer om hendelser i egen leksjon om JavaScript.

2.2.4. Syntaks for beslutninger, løkker, matriser (arrays), operatorer++

Det antas at du kan grunnleggende programmeringsprinsipper før du starter med dette kurset, hvis ikke bør du få tak i en introduksjonsbok til programmering eller søke på web etter grunnopplæring. Det er generelt mye enklere å forstå andres kode enn å selv lage kode som virker, og til en viss grad kan du komme langt i dette kurset dersom du er på forstå-stadiet. Du trenger altså ikke være noe programmeringsfantom for å *forstå* Ajax-kode, men for å kunne produsere kode selv må du gripe programmeringsutfordringene, og eventuelt gjøre en ekstra innsats nå i starten for å komme på riktig nivå.

Du bør kjenne til og kunne bruke grunnstenene fra enten JavaScript eller et annet språk før du går videre til leksjon 3. Test kjapt på om følgende stikkord virker kjent: if, for, while, and, or, not, teller++, mineTall[4] (element nr 5 i en matrise/array), function summer(tall1, tall2)... Forhåpentligvis er ikke disse ordene gresk for deg :-)

Her er korte eksempler på syntaksen som brukes i JavaScript, kodesnuttene henger ikke sammen selv om de kommer forløpende:

Variabler: Det er best (blir mest lesbart) å bruke stikkordet `var` foran variabler, men en kan utelate dette.

```
alder = 28; //lov, men best skikk å ha "var" foran nye variabler
var enAnnenAlder = 45;
var navn = "Ola Nordmann";
var etAnnetNavn = 'Kari Nordmann';
var sum = alder + enAnnenAlder; //lagrer 73 i variabelen sum
navn = "Petter Nordmann"; //ikke var her, variabel er deklartert fra før
var rik = false; //boolsk variabel
alder++; //alder er nå 29
enAnnenAlder -= 10; //nå lik 35
```

Beslutning (valg), if then else:

```
if (tall >= 18) {
    document.writeln("du er myndig!");
}
else {
    document.writeln("du er mindre enn 18 år, og derfor ikke myndig");
}
```

For-løkke:

```
var teller;
for (teller = 0; teller < 10; teller++) {
    document.writeln("Tallet er " + teller + "<br />");
}
```

Legg merke til at løkken skriver ut alle tall mellom 0 og 9, men ikke 10. Siden vi bruker en `
` tag, blir det linjeskift mellom hver linje som skrives ut.

Funksjon: Du må ikke la variabler og funksjoner ha samme navn, det leder til merkelig oppførsel. Du kan heller ikke ha to funksjoner med samme navn i JavaScript, selv om antall argumenter varierer (slik man kan i mange andre programmeringsspråk).

```
function summerTall(nrEn, nrTo, nrTre) {  
    return nrEn + nrTo + nrTre;  
}
```

En interessant observasjon er at variabler deklart utenfor en funksjon, er tilgjengelig inne i funksjonen uten videre, og beholder sin verdi så lenge JavaScript kjører på siden.

Matriser (Arrays): En matrise er en samling av informasjon i en tabellaktig struktur. Noen kaller det for matriser, andre for arrays, noen for vektorer og atter andre for tabeller. Vi bruker begrepet *matrise* i dette kurset.

```
var dager = new Array(7);  
dager[0] = "mandag";  
dager[1] = "tirsdag";  
dager[2] = "onsdag";  
dager[3] = "torsdag";  
dager[4] = "fredag";  
dager[5] = "lørdag";  
dager[6] = "søndag";  
document.writeln("I dag er det " + dager[6]); //søndag
```

While-løkke som bruker dagene fra forrige matrise kreativt:

```
var teller = 0;  
while (teller <= 6) {  
    document.writeln("Hvilken dag er det i dag? Er det ");  
    document.writeln(dager[teller] + "??? Nei!");  
    document.writeln("Er det " + dager[teller] + "??? Atter nei!");  
}
```

Ansvarlige for Barne-TV på 80-tallet, ville ikke likt denne *uendelige* løkka. Noe bedre ville det vært dersom siste linje inne i while-løkken var for eksempel `teller++`; Oppgave (frivillig): Legg inn en if-setning som gjør at løkka avsluttes med å skrive ut "Er det søndag??? JA!" istedenfor "Er det søndag??? Atter nei!"

Noen nyttige metoder: Objektene har metoder som kan brukes. Her er noen generelle metoder for objekter som inneholder tekststrenger (vi vil gå mer inn på slike metoder utover i kurset, men tar med noen eksempler bare for å illustrere virkemåten)

- `.charAt(n)` – Returnerer tegnet i posisjon n
- `.indexOf("@")` – Returnerer posisjonen til tegnet @ i tekststrengen
- `.substring(a,b)` – Returnerer delstrengen fra posisjon a til posisjon b
- `.toLowerCase()` – konverterer til små bokstaver
- `.toUpperCase()` – konverterer til store bokstaver

```
<script type="text/javascript" language="javascript">
    var navn = "Svend Andreas Horgen";
    //tegn nr    0123456789...

    var nummerNi = navn.charAt(8);
    document.writeln(nummerNi + "<br>"); //skriver ut "d"
    document.writeln(navn.indexOf("A") + "<br>"); //6
    var delstreng = navn.substring(6,9); //henter ut "And"
    document.writeln(delstreng+ "<br>");
    var store = navn.toUpperCase();
    document.writeln(store + "<br>"); //store bokstaver
    document.writeln(navn.toLowerCase() + "<br>"); //små
</script>
```

2.2.5. Eksterne JavaScript-filer

Dersom du får mye kode, kan du samle funksjonene i en egen fil på formen *filnavn.js*. Du inkluderer eksterne filer ved å plassere følgende setning i head-seksjonen til en HTML-side:

```
<script src="filnavn.js" type="text/javascript"></script>
```

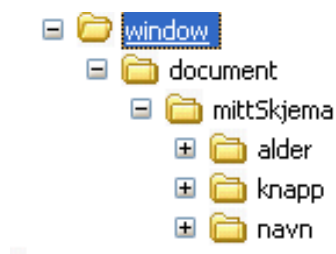
2.3. Document Object Model (DOM)

Vi har nå sett raskt på syntaksen i JavaScript, og at kode kan kalles opp basert på hendelser. JavaScript passer perfekt i en nettbasert verden siden språket kan operere mot det såkalte DOM-treet til en nettside.

2.3.1. Hva er DOM?

Ethvert HTML-dokument har en viss struktur, med overskrifter, avsnitt, tekstlig innhold og så videre. Denne strukturen reflekteres i dokumentets DOM (Document Object Model), som er en objektbasert representasjon av nettsiden. Strukturen i DOM er hierarkisk¹, noe som gjør det enkelt å navigere med for eksempel JavaScript. Alle HTML-elementer blir en del av sidens DOM når siden vises i nettleseren. Sagt med andre ord: Nettleseren har en DOM-beskrivelse av en nettside. Dersom nettleseren har flere nettsider åpne (ulike faner), har hver side en egen DOM med ulikt innhold.

Alle HTML-dokumenter vil inneholde window og document, og vi kan spesifisere et bestemt element ved å skrive window.document.etEllerAnnet.



¹ Hierarkisk betyr at et element alltid tilhører (er underordnet eller barn av) et annet element. I HTML (og XML) vil et element som er nøstet inni et annet element være barn av dette elementet. To elementer som begge er nøstet inni det samme elementet, men ikke inni hverandre, sies å være søken, og de har samme forelder.

Figur 5: Visualisering av DOM-hierarkiet for eksemplet nedenfor (ikke fullstendig, har bare tatt med navngitte elementer)

2.3.2. Eksempel på bruk av DOM i JavaScript

Det er interessant at DOM kan manipuleres ved hjelp av JavaScript. Gitt et skjema hvor brukeren skal skrive inn navn og alder (se nedenfor). Før skjemaet sendes til `behandle.php`, vil funksjonen `validerAlder()` utføres. Dersom denne returnerer `true`, blir `onsubmit true`, og `behandle.php` kalles opp. Hvis `validerAlder()` returnerer `false`, blir `onsubmit false`, og dermed blir skjemaet ikke sendt til `behandle.php`.

```
<form name="mittSkjema"
        action="behandle.php"
        method="post"
        onsubmit="return validerAlder();"
>
    Alder: <input type="text" name="alder"><br>
    Navn: <input type="text" name="navn">
    <input type="submit" name="knapp">
</form>
```

Legg merke til at `<form>` spenner over mange linjer. Ikke glem avsluttende `>` på linje 5. Vi kan nå lage en JavaScript-funksjon som validerer at alderen er gyldig. Denne funksjonen kan gjøre noe ved feil. Dersom alderen er mindre enn 0, kommer en alert-boks frem, og alderfeltet får fokus slik at brukeren kan korrigere alderen. Det er viktig at funksjonen returnerer `true` hvis alt går bra, og `false` ellers, slik at `onsubmit` (der funksjonen ble kalt opp) kan få riktig sluttverdi.

```
<html>
<head>
<script type="text/javascript" language="javascript">
function validerAlder() {
    var alder = window.document.mittSkjema.alder.value;
    if (alder < 0 ) {
        alert("alder må være større enn 0");
        window.document.mittSkjema.alder.focus() ;
        return false;
    }
    return true; //skjer bare hvis vi kommer hit
} //slutt på funksjonen her
</script>
</head>
```

Legg merke til sammenhengen mellom setningen i fet skrift og DOM-hierarkiet i Figur 5. Vi kan altså navigere til ett bestemt element (et tekstfelt i skjemaet i dokumentet i vinduet) ved hjelp av kode. Resultatet av denne navigasjonen blir en objektrepresentasjon av DOM-elementet. Dermed kan ulike metoder brukes. Et tekstfelt har metoder som `focus()` og `blur()`, og en kan hente ut nåværende verdi ved hjelp av egenskapen `value`, slik:

```
document.writeln(window.document.mittSkjema.alder.value)
```

2.3.3. HTML eller XHTML?

Vi har tidligere nevnt forskjellen på HTML og XHTML. Hovedforskjellen er at XHTML stiller strenge krav til hvordan koden er strukturert. I vanlig HTML kan du skrive slik:

```
<H1>Tittel her</H1>
Velkommen hit, her kommer <strong>tekst i fet skrift<em> og kursiv inni
der</strong></em>
<p>Nytt avsnitt
<p>Her er et bilde 
```

Dette aksepteres i HTML. I XHTML må for eksempel følgende oppfylles:

- tagger/elementer ha små bokstaver
- alt være innenfor ett eller flere elementer ("Velkommen hit..." har for eksempel ikke noe element rundt seg)
- alle elementer avsluttes, også linjeskift og bilder
- rekkefølgen på avslutning av tagger må være riktig (strong og em avsluttes i feil rekkefølge)

Du ser brudd på alle disse i den første koden rett over. En korrekt XHTML-versjon vil være slik:

```
<h1>Tittel her</h1>
<p>Velkommen hit, her kommer <strong>tekst i fet skrift<em> og kursiv inni
der</em></strong></p>
<p>Nytt avsnitt</p>
<p>Her er et bilde  </p>
```

Når nettleseren skal lage en DOM av siden som mottas, er det en fordel at koden er i form av XHTML slik at DOM-treet kan lages best mulig. Det er vanskelig med JavaScript å erstatte for eksempel teksten "Velkommen hit..." med noe annet i den første HTML-versjonen, mens det blir en enkel jobb i den andre XHTML-versjonen.

Se ellers mer om DOM og DOM inspectors blant lenkene på ressursiden til denne leksjonen.

2.3.4. Et lite eksempel på DHTML

DHTML står for Dynamisk HTML. Teknikken har ikke vært brukt så veldig mye, trolig på grunn av mangelen på asynkron kommunikasjon med tjeneren. Nå som den biten er på plass, kan vi si at Ajax egentlig er... "Asynkron DHTML" (!)

Vi kan manipulere stiler med JavaScript-kode:

```
document.getElementById("id").style.property="verdi"
```

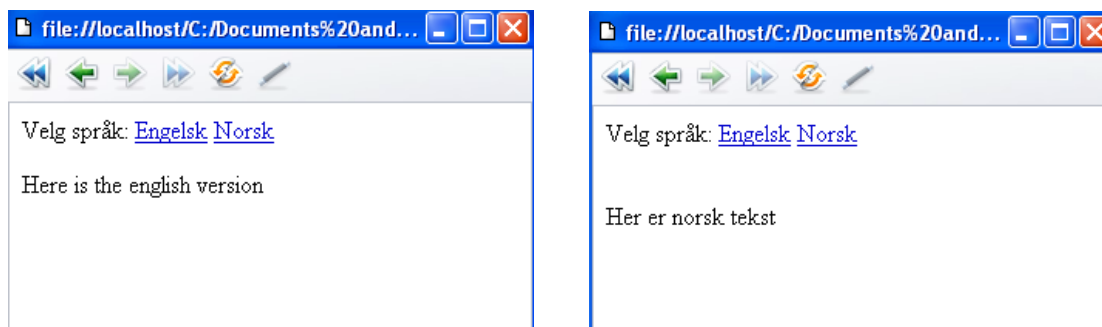
Dermed kan vi for eksempel vise og skjule tekst på følgende måte:

```
<html><head><script type="text/javascript">
function engelsk(){
    document.getElementById("engelsk_tekst").style.visibility = "visible"
    document.getElementById("norsk_tekst").style.visibility = "hidden"
}
function norsk(){
    document.getElementById("engelsk_tekst").style.visibility = "hidden"
    document.getElementById("norsk_tekst").style.visibility = "visible"
}
```

```
}
</script></head><body>
<p>Velg språk: <a href="#" onmouseover="engelsk()">Engelsk</a>
      <a href="#" onmouseover="norsk()">Norsk</a></p>
<div id="engelsk_tekst">Here is the english version</div>
<div id="norsk_tekst">Her er norsk tekst</div>
</body></html>
```

Her ser du samspillet mellom teknologiene. I CSS kan alle elementer ha egenskapen `visibility` satt til `visible` eller `hidden` (synlig eller skjult). Det første som skjer når siden lastes, er at JavaScriptet lastes inn, men det kjøres ikke. Derimot vises teksten "Velg språk..." og begge `div`-taggene (med engelsk og norsk tekst) vises hvis de er satt `visible`. Når musepekeren kommer over for eksempel "Engelsk", vil nettleseren fyre `onmouseover`-hendelsen. Denne er satt til å gjøre et kall til funksjonen `engelsk()`, og siden denne er deklarerert i `head`-seksjonen, utføres den nå.

Funksjonen `engelsk()` bruker metoden `getElementById()` til å plukke ut et helt bestemt element i DOM-treet, nemlig det med `id` satt til "engelsk_tekst". CSS-egenskapen `visibility` settes til `visible`. Deretter hentes et nytt element i DOM-treet frem (det med `id` satt til "norsk_tekst"). Dette skjules. Funksjonen `norsk()` gjør det motsatte (viser norsk tekst og skjuler engelsk). Som resultat ser vi her et helt konkret eksempel på DHTML: Elementer i HTML-koden skjules og vises dynamisk etter at siden er lastet inn, som følge av det brukeren måtte gjøre.



Figur 6: Musepekeren holdes over lenken "Engelsk" til venstre og "Norsk" til høyre. Innholdet endres umiddelbart.

Tenk litt over mulighetene du nå står overfor. Med Ajax kan du gjøre slike ting, men til forskjell fra vårt eksempel, hvor teksten som ble vist/skjult, kan du med Ajax hente teksten fra en database, fil eller liknende på tjeneren. Nettopp *det* utgjør en enorm forskjell. Vi håper du gleder deg til veien videre med Ajax i dette kurset!

2.4. Kilder

- Ajax: A New Approach to Web Applications, Jesse James Garrett, 18.februar 2005. <http://adaptivepath.com/publications/essays/archives/000385.php>
- Håkon Wium Lie, oppfinner av CSS: http://en.wikipedia.org/wiki/Hakon_Wium_Lie
- Se ressursiden tilhørende denne leksjonen for enda mer stoff om Ajax-teknologier.