Institutt for informatikk og e-læring, NTNU

Introduksjon til JavaScript

Hentet fra leksjonen «JavaScript og DHTML» av
Svend A. Horgen og Øyvind Hallsteinsen (2011)
Tilpasset av Anette Wrålsen (2012)
Lærestoffet er utviklet for faget LC372D Publisering på internett

I denne leksjonen får du en introduksjon til JavaScript, DOM (Document Object Model) og DHTML.

Hva er Javascript?

JavaScript ble utviklet av Netscape i 1995, og har fått stor utbredelse på web. Alle vanlig brukte nettlesere støtter JavaScript, men det er som regel mulig for brukeren å slå av JavaScript. Dette er viktig å ha i bakhodet når en utvikler websider med JavaScript, siden brukere som har nettlesere hvor JavaScript er avslått eller ikke støttes, ikke vil se noe resultat av JavaScript.

Mange tror at JavaScript er det samme som Java, eller i det minste en nær slektning. Det stemmer ikke. Språkene har nokså lik syntaks, men har ellers ingenting med hverandre å gjøre.

JavaScript er et scriptspråk, som tolkes linje for linje. Det betyr at når et script har en feil i linje 13, kan meget mulig linje 1-12 kjøres før scriptet stopper. JavaScript er også enkelt å integrere i HTML-kode. Det er rett og slett bare å plassere en script-tag et sted i HTML-koden, og legge JavaScript-kode inne i denne script-taggen. JavaScript mangler for øvrig arv og klasser, men er likevel objektbasert til en viss grad.

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 (som er en måte å bruke JavaScript på), fått utviklere verden over til å revurdere gamle fordommer mot språket.

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 nettleserne tilbyr å blokkere kjøring av script eller popup-vinduer, og mange assosierer JavaScript med virus – feilaktig. Det er riktignok en viss fare for at de som tillater JavaScript i nettleseren kan kjøre kode som stjeler informasjonskapsler (cookies) og annen informasjon, eller i verste fall aktivererer skummel programkode, men i denne leksjonen skal vi fokusere på hva JavaScript (kan/bør) brukes til.

En annen årsak til at JavaScript har fått et dårlig rykte har vært den dårlige og trege implementasjonen av JavaScript i nettleserne. Fordi så mange webapplikasjoner nå har omfattende bruk av JavaScript, har dette fått utviklerne av nettlesere til å gjøre noe med dette. For eksempel er én av årsakene til at Google har gitt ut sin egen net-

tleser, Google Chrome, at de ønsker å skape et trykk i nettlesermarkedet for å ha en god JavaScript-implementasjon.

Det er liten tvil om at JavaScript spiller en viktig rolle for mange nettsteder i dag, og vi skal se hva JavaScript er, oppsummere syntaksen kort (slik at du skal kunne lese kode og forstå sånn ca hva som skjer), og vise hvordan ferdige script som andre har laget, kan integreres på ditt nettsted.

Noe av det fine med JavaScript, er at det gradvis kan bygges inn i et nettsted. Her er noen av de tingene du kan bruke JavaScript til:

- ✓ Validere om et skjema er riktig utfylt før det sendes til webtjeneren.
- ✓ Navigere i historikken, for eksempel gå tilbake til forrige side som brukeren var på, ved å trykke på en knapp eller en lenke.
- ✓ Finne ut mer informasjon om hvilken nettleser brukeren har.
- ✓ Vise og skjule informasjon, noe som muliggjør for eksempel å lage menyer som åpner/lukker seg.
- ✓ Utføre beregninger på klientsiden, uten at tjeneren må belastes (små kalkulatorer).
- ✓ Lage mer brukervennlige websider, med for eksempel klikk-og-dra-løsninger og andre GUI-elementer som er kjent fra vanlige applikasjoner.
- ✓ Som bruker benytte et webbasert skrive/regnearkprogram, for eksempel med ”Google Docs & Spreadsheets” (www.google.com).

Syntaks og eksempler på kode

Når du skal bruke JavaScript i websiden din, er det bare å skrive kode i script-taggen:

```
<html>
<head><title>Første JavaScript</title></head>
<body>
<h1>En liten JavaScript-test</h1>
<script language="javascript">
  alert("Hei verden!");
</script>
</body>
</html>
```

Vi skal nå se på JavaScript-syntaks gjennom noen eksempler.

Funksjoner i JavaScript

I JavaScript er det ikke nødvendig å avslutte setninger med semikolon, men det er en god vane fordi det da blir tydelig hvor setningen avsluttes.

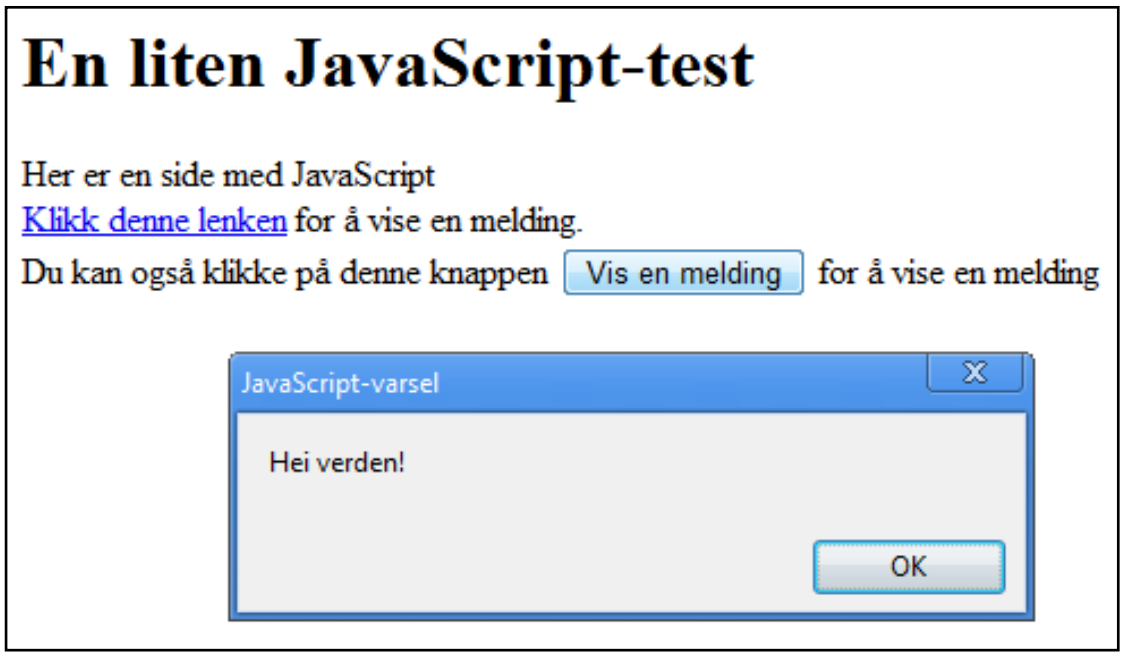
Dersom du skal lage dine egne funksjoner, bør disse defineres i hode-seksjonen i HTML-dokumentet, og så kalles opp, slik:

```
<html>
<head>
```

```
<title>Eksempel på funksjoner</title>
<script language="javascript">
  function visMelding() {
    alert("Hei verden!");
  }
</script>
</head>

<body>
<h1>En liten JavaScript-test</h1>
<p>Her er en side med JavaScript<br />
<a href="javascript:visMelding();">Klikk denne lenken</a>
for å vise en melding. <br />
Du kan også klikke på denne knappen
<input type="button" onclick="visMelding();" value="Vis en meld-
ing" />
for å vise en melding
</p>
</body>
</html>
```

I nettleseren ser eksemplet ut som på figuren nedenfor. Legg merke til hvordan samme JavaScript-funksjon kan kalles fra flere steder i koden, både fra en lenke (ved å angi `javascript:` som prefiks i `href`-verdien), og ved klikk på en knapp.



Det er selvsagt også mulig å sende argumenter til funksjoner som i andre språk, på denne måten:

```
<html>
<head>
<title>Eksempel på funksjoner</title>
<script language="javascript">
function visMelding(melding){
```

```
    alert(melding);
}
</script>
</head>

<body>
<h1>En liten JavaScript-test</h1>
<p>Her er en side med JavaScript<br />
<a href="javascript:visMelding('Hello world!');">Klikk denne
lenken</a>
for å vise en melding. <br />
Du kan også klikke på denne knappen
<input type="button" onclick="visMelding('Hei');" value="Vis en
melding" />
for å vise en melding
</p>
</body>
</html>
```

Hendelser

I koden over så du hvordan en JavaScript-funksjon var knyttet opp mot `onclick`-hendelsen til en knapp. Det er også mulig å knytte andre hendelser opp mot knapper (og andre elementer), og de fleste er like selvforklarende som `onclick` (klikk med museknapp).

Musehendelser:

- ✓ `onclick/ondblclick:` enkelt- eller dobbeltklikk
- ✓ `onmousedown/onmouseup:` museknapp klikkes ned/opp, atomiske operasjoner
- ✓ `onmousemove:` mus beveger seg
- ✓ `onmouseover/onmouseout:` musepeker kommer over et element/bort igjen
- ✓ Tastaturhendelser:
- ✓ `onkeydown:` trykk på tastaturknapp
- ✓ `onkeypress:` så lenge en tastaturknapp holdes nede
- ✓ `onkeyup:` når en nedtrykt knapp slippes
- ✓ Diverse hendelser:
- ✓ `onabort:` noe avbrytes
- ✓ `onfocus/onblur:` fokus til eller bort fra et element, for eksempel tekstfelt
- ✓ `onchange:` noe endres, for eksempel et tegn skrives inn i et tekstfelt
- ✓ `onerror:` når noe har gått galt
- ✓ `ondragdrop:` et element dras og slippes
- ✓ `onload:` siden lastes
- ✓ `onunload:` nettleseren går til en ny side
- ✓ `onreset:` Reset-knapp klikkes
- ✓ `onresize:` brukeren endrer størrelse på et element eller en side
- ✓ `onsubmit:` skjemainformasjon sendes, submit-knappen trykkes

Det er ikke bare knapper i et skjema som kan gjøre nytte av slike hendelser. Alle HTML-elementer kan benytte hendelsene.

Grunnleggende syntaks

Det er ikke nødvendig å deklarere variabler i JavaScript, men det er lov å bruke ordet `var` foran variabler ved opprettelse. Dette er lurt å benytte seg av, fordi koden da blir lettere å lese. Du trenger heller ikke tenke på datatyper, sånn som string, integer, o.l., når du koder i JavaScript.

Neste eksempel illustrerer syntaksen, og trenger neppe nærmere forklaring utover kommentarene.

```
alder = 28; // Lov, men best å ha "var" foran nye variabler
var enAnnenAlder = 45; // Slik som dette
var navn = "Ola Nordmann"; // Doble anførselstegn (") kan brukes
i strenger
var etAnnetNavn = 'Kari Nordmann'; // Det kan enkle (') også
var sum = alder + enAnnenAlder; // Lagrer verdien 73 i variabe-
len sum
navn = "Petter Nordmann"; // Ikke var her, variabel er deklarert
fra før
var rik = false; // boolsk variabel (true eller false)
alder++; // Alder er nå 29
enAnnenAlder -= 10; // enAnnenAlder nå lik 35
navn = 23; // Variabelen navn skifter nå datatype fra string til
integer
```

Med `document.writeln()` kan en vanlig linje skrives ut i nettleseren. Altså ikke i en alert-boks, men som del av den teksten som brukeren ser. Dette kan brukes for å vise en spesiell farge på søndager:

```
<html>
<head><title>Søndager vises med annen farge</title></head>
<body>
<script language="javascript">
  var d = new Date(); //finner informasjon om nåværende dato og
tidspunkt
  var dag = d.getDay();
  //alert(dag); //fjern kommentaren for å teste hvilken dag det
er
  //dag = 0; //fjern kommentarene for å teste at farge vises
riktig
  if (dag == 0) {
    document.writeln("<h1 style='color:green'>I dag er det
søndag</h1>");
  } else {
    document.writeln("<h1 style='color:black'>I dag er det ar-
beidsdag</h1>");
  }
</script>
</body>
</html>
```

Her ser du også eksempel på en if-else-struktur med JavaScript. Du ser også hvordan stilsett (CSS) brukes for å endre fargen til grønn, men her kunne vi også brukt en font-tag med color-attributt satt, eller andre ting. Legg merke til hvordan vi med `document.writeln()` kan skrive ut vanlig tekst og HTML-kode. Nettleseren tolker HTML-koden, og dermed vil "I dag er det søndag" vises som en h1-overskrift dersom det er søndag i dag.

Innebygde objekter

I JavaScript er det slik at et Date-objekt har mye informasjon om nåværende dato og tidspunkt. Med metoder og egenskaper, kan vi plukke ut informasjon fra dette dato-objektet. Metoden `getDay()` henter ut et tall som representerer dagen i dag. Tallet 0 betyr søndag, mens 6 betyr lørdag. Dermed kan vi i betingelsen i if-testen sjekke om brukeren besøker websiden på en søndag eller en vanlig ukedag. Legg merke til at vi bruker `==` for å teste på likhet, mens `=` brukes for å tilordne verdier til en variabel.

Neste eksempel viser hvordan en løkke kan brukes til å skrive ut alle ukedager. Først lages en matrise (også kalt tabell, og på engelsk array). Innholdet er tekststrenger som utgjør de norske navnene på ukedagene. Deretter finner vi dagens datoinformasjon, og plukker ut tallet som representerer denne dagen ved hjelp av metoden `getDay()`. Deretter skriver vi ut "I dag er det tirsdag" dersom det er tallet 2 som ligger i variabelen som heter `idag`. Legg merke til hvordan vi benytter oss av at matrisen har samme indeks som første mulige dag. Metoden `getDay()` returnerer et tall som er mellom 0 og 6, og matrisen har 7 elementer, med indekser (nøkler) fra 0 til 6. Det er samsvar, og vi kan derfor bruke returen fra `getDay()` til å hente fram riktig dagsnavn på norsk.

```
<html>
<head>
<title>Hvilken dag er det i dag, hvilken ...</title></head>
<body>
<script language="javascript">
var ukedag=new Array("Søndag",
                     "Mandag",
                     "Tirsdag",
                     "Onsdag",
                     "Torsdag",
                     "Fredag",
                     "Lørdag"
                     );
var d = new Date();
var idag = d.getDay();
document.write("<p>I dag er det " + ukedag[idag] + "</p>");
document.write("<p>Alle ukens dager, er: </p>");
document.write("<ul>");
for (teller=0; teller < ukedag.length; teller++){
  document.write("<li>" + ukedag[teller] + "</li>");
}
document.write("</ul>");
</script>
</body>
</html>
```

I for-løkken skrives alle dagene ut. Vi lar for-løkken gå fra 0 til 6, men i stedet for å hardkode tallet 6, henter vi antall elementer i matrisen (som er 7) ved hjelp av egenskapen `length` anvendt på matrisen. Figuren nedenfor viser at dagene skrives ut som en punktmerket liste. Sett deg inn i koden og forstå hvorfor det blir slik.

Det fins også en rekke nyttige metoder for å behandle strenger, for eksempel `toLowerCase()`, `indexOf()` og `substring()`. Vi kommer tilbake til noen av disse i et senere eksempel.

Dynamisk HTML (DHTML)

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 hierarkisk, 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 (flere vinduer og/eller faner), har hver side en egen DOM med ulikt innhold.

Alle vinduer i nettleseren inneholder toppnivåelementet `window` og dets barn `document`. Vi kan nå de øvrige elementene i hierarkiet gjennom disse, enten ved å traversere hierarkiet av objekter eller ved å søke opp det eller de objektene vi ønsker. Som vi skal se etterhvert er det sistnevnte det mest praktiske. Når vi jobber med DOM i JavaScript kalles det gjerne Dynamic HTML (DHTML). Med DHTML kan vi lese, modifisere, slette og legge til objekter i DOM.

Vi kan se på et enkelt eksempel:

```
<html>
  <head>
    <title>Min tittel</title>
  </head>
  <body>
    <!-- Mitt dokument -->
    <h1>Min overskrift</h1>
    <p id="mintekst">Min tekst</p>
  </body>
</html>
```

I henhold til DOM har vi her å gjøre med flere ulike typer noder:

- ✓ **Dokumentnode:** Hele dokumentet er en dokument-node.
- ✓ **Elementnoder:** Alle HTML-elementer er elementnoder.
  I eksemplet vil det si at `<html>`, `<head>`, `<body>`, `<h1>` og `<p>` er elementnoder.
- ✓ **Tekstnoder:** Tekst inne i HTML-elementer er tekstnoder.
  I eksemplet vil det si `"Min tittel"`, `"Min overskrift"` og `"Min tekst".`
- ✓ **Attributtnoder:** Alle attributter er attributtnoder.
  I eksemplet vil det si `"id"`.
- ✓ **Kommentarnoder:** Alle kommentarer er kommentarnoder.
  I eksemplet vil det si `"Mitt dokument".`

Dette med tekstnoder er spesielt viktig å få med seg -- elementnoder som `<title>` og `<h1>` inneholder ikke tekst selv, men er foreldre til tekstnoder som inneholder tekst. Som vi skal se må vi i DOM bevege oss inn til barnenoden av elementnodene for å hente ut teksten, eller vi kan bruke egenskapen `innerHTML`.

DOM-hierarkiet

DOM er oppbygd hierarkisk. Det betyr at et element alltid tilhører (er underordnet eller barn av) et annet element. I HTML vil et element som er nøstet inni et annet element være barn (*child*) av dette elementet. To elementer som begge er nøstet inni det samme elementet, men ikke inni hverandre, sies å være søsken (*siblings*), og de har samme forelder (*parent*).

I eksemplet nedenfor ser vi DOM for eksempeldokumentet vårt:

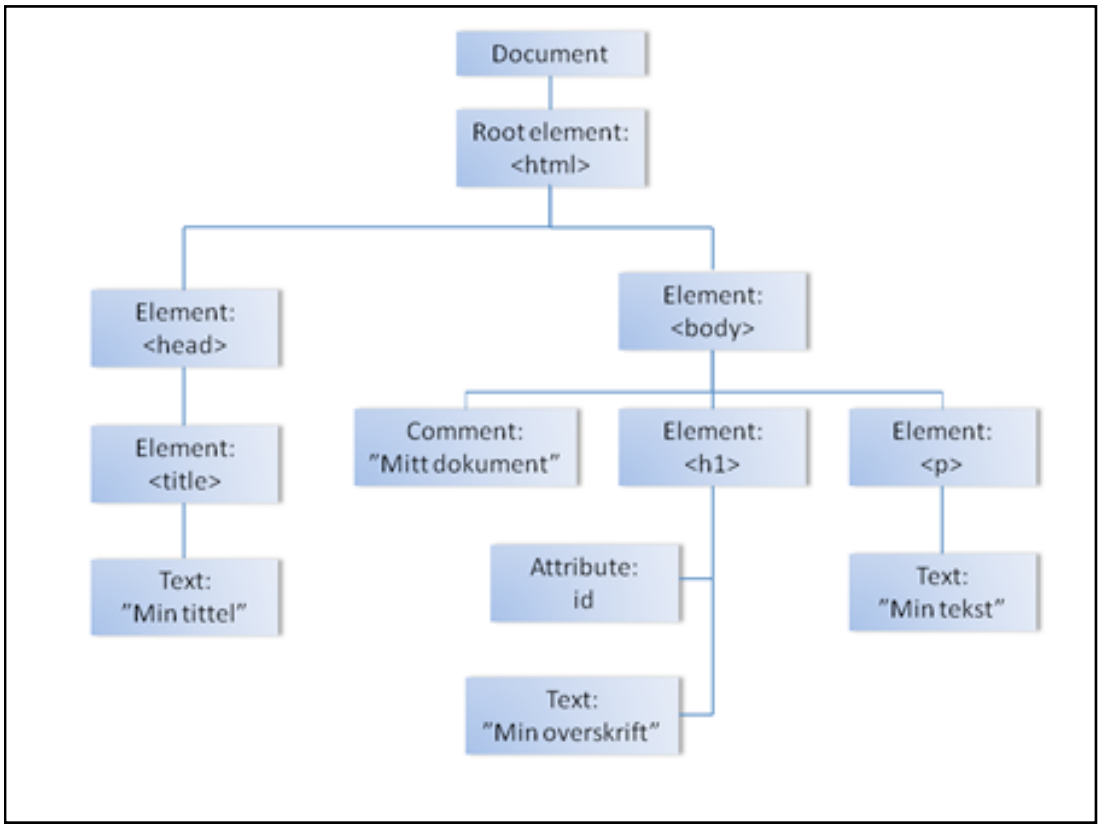
Merk at all tekst er egne noder, teksten er altså ikke noe som tilhører for eksempel `<h1>`- eller `<p>`-elementene.

Egenskaper og metoder

DOM er objektorientert, det vil si at alle nodene i DOM er et objekt med egenskaper og metoder. En egenskap er en verdi som sier noe om objektet. Eksempelvis har `document`-objektet en egenskap `document.URL` som sier hvilken URL dokumentet ligger på, for eksempel `http://localhost/minfil.html`. Metoder er funksjoner vi kan bruke på objektet. Eksempelvis har `document`-objektet en metode `document.writeln(line)` som brukes for å skrive en linje med tekst i dokumentet. Eksempel:

```
document.writeln("Min URL er: " + document.URL);
```

En interessant egenskap hos elementer som har tekstelementer som barn, er innerHTML. Denne egenskapen kan vi bruke for å hente ut teksten uten å måtte traversere hierarkiet helt ned til tekstobjektet. Vi kan ta utgangspunkt i vårt eksempel, hvor vi har følgende linje:




```
<p id="mintekst">Min tekst</p>
```

For å hente ut teksten kan vi gjøre dette enten ved å traversere hele hierarkiet (NB: linjen kan i din leser være delt i to, men all teksten hører sammen):

```
document.writeln("Dette er teksten: " +
document.getElementById("mintekst").childNodes[0].nodeValue);
```

Eller vi kan bruke innerHTML (skal stå på samme linje, men blir delt pga. bredden):

```
document.writeln("Dette er teksten: " +
document.getElementById("mintekst").innerHTML);
```

Effektiv DOM-behandling krever ryddig kode

For å kunne jobbe effektivt med DOM er det viktig å lage websider med ryddig kode.

Vi kan se på et eksempel -- koden nedenfor vil fungere og vises slik den skal i alle nettlesere:

```
<h1>Tittel her</h1>
```

```
Her er <strong>tekst i fet skrift <em>og kursiv inni der</
strong></em>
<p>Nytt avsnitt
<p>Her er et bilde <img src=”bilde-4.png”>
```

Mens dette altså aksepteres av nettleserne vil koden kunne skape problemer når du skal jobbe med den fra JavaScript.

Følgende huskeliste bør følges:

- ✓ **Små bokstaver:** Elementer (tagger) skal ha små bokstaver
- ✓ **Ingen løs tekst:** Alt skal være innenfor ett eller flere elementer (”Her er…” har for eksempel ikke noe element rundt seg
- ✓ **Avslutt elementer:** Alle elementer må avsluttes, også linjeskift og bilder
- ✓ **Riktig rekkefølge:** Rekkefølgen på avslutning av tagger må være riktig (strong og em avsluttes i feil rekkefølge)

Du ser brudd på alle disse punktene i den koden ovenfor. Skal vi følge regellisten vil koden se slik ut:

```
<h1>Tittel her</h1>
<p>Her er <strong>tekst i fet skrift <em>og kursiv inni der</
em></strong></p>
<p>Nytt avsnitt</p>
<p>Her er et bilde <img src=”bilde-4.png” /></p>
```

Når nettleseren skal lage en DOM av siden som mottas, er det en stor fordel at koden er i i tråd med disse reglene, slik at DOM-treet kan lages best mulig. Det er vanskelig med JavaScript å erstatte for eksempel teksten ”Her er…” med noe annet i den første versjonen, mens det blir en enkel jobb i den andre versjonen.

Enklere og mer robust identifikasjon av elementer i DOM-treet

I eksemplet med validering av alder og epostadresse navigerte vi hele veien gjennom DOM frem til elementene med innholdet i feltene alder og epost. Dette er tungvindt og unødvendig, når vi kan plukke ut elementene direkte ved hjelp av attributtet `name` eller attributtet `id`.

Vi kan se på et utdrag av skjemaet:

```
<form name="mittSkjema"
...
    <td><input type="text" name="alder" /></td>
...
    <td><input type="text" name="epost" /></td>
...
</form>
```

Koden for å hente ut innholdet av feltene så slik ut:

```
var alder = window.document.mittSkjema.alder.value;
var epost = window.document.mittSkjema.epost.value;
```

Det er mulig å utelate window, fordi JavaScript implisitt vil anta at vi starter der:

```
var alder = document.mittSkjema.alder.value;
var epost = document.mittSkjema.epost.value;
```

Fordi vi hadde gitt de to elementene et `name`, kunne vi brukt dette for å hente frem verdiene ved hjelp av metoden `getElementsByName()`. Attributtet `name` kan brukes på én eller flere elementer i samme dokument. Det betyr at vi kan jobbe med flere elementer samtidig dersom vi bruker `getElementsByName()`, selv om det ikke er aktuelt i vårt tilfelle. En fordel med å bruke `getElementsByName()` fremfor å navigere i DOM-treet er at vi ikke er avhengige av å kjenne strukturen i treet for å identifisere riktig element. Vi kunne gjort slik:

```
var alder = document.getElementsByName("alder").value;
var epost = document.getElementsByName("epost").value;
```

Mens flere elementer kan ha samme name, kan vi bruke attributtet `id` for å identifisere ett enkelt element unikt. Per definisjon skal et dokument aldri ha mer enn ett element med samme `id`. Vi kunne definert elementene slik:

```
<form name="mittSkjema"
...
    <td><input type="text" name="alder" id="alder1" /></td>
...
    <td><input type="text" name="epost" id="epost1" /></td>
...
</form>
```

Bruk av `id`-attributtet åpner for å ha flere skjemaelementer som inneholder det samme, vi kan for eksempel tenke oss et skjema hvor vi registrerer flere personer. Da vil vi gi alle input-feltene for alder samme `name ("alder")`, men ulik `id ("alder1", "alder2", ...)`. Vi kan så identifisere et enkelt element ved hjelp av metoden `getElementById()`:

```
var alder = document.getElementById("alder1").value;
var epost = document.getElementById("epost1").value;
```

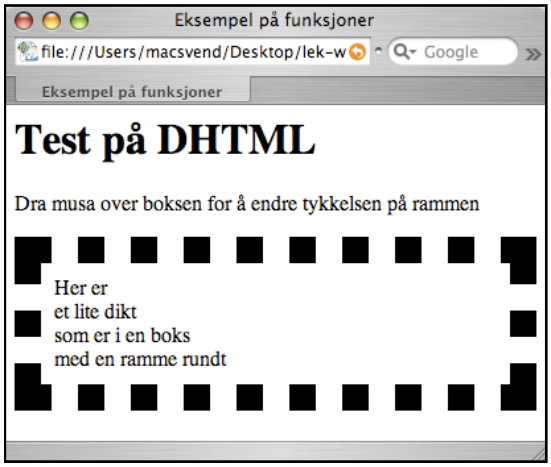
Som vi ser får vi til en enklere og mer robust identifisering av elementer i DOM-treet dersom vi legger litt arbeid i konstruksjonen av dokumentet.

Et lite eksempel på DHTML og getElementById

Vi skal ta for oss et eksempel som illustrerer dynamikken i bruk av DHTML, hvor vi også tar i bruk metoden `getElementById`.

Som vist i neste eksempel, har vi laget et div-element med `id="boksenMin"`. Denne boksen kan vi henvise til med JavaScript-kode ved å bruke `getElement-ById()` metoden til document-objektet.

I det vi åpner websiden, ser vi en tynn ramme rundt teksten. Men når vi beveger musa over teksten, vises derimot en tykk, prikket linje som ramme:



Som vist i koden til denne websiden, plukker vi ut et bestemt element i DOM-treet til nettleseren, og manipulerer dette på direkten:

```
<html>
<head><title>Eksempel på funksjoner</title>
<script language="javascript">
  function endreRamme(){
    var boks = document.getElementById("boksenMin");
    boks.style.border = "20px dotted";
  }
</script>
</head>

<body>
<h1>Test på DHTML</h1>
<p>Dra musa over boksen for å endre tykkelsen på rammen</p>
<div id="boksenMin"
  style="border-width:1px; border-style:solid; padding:10px"
  onmouseover="endreRamme();">
  Her er <br />
  et lite dikt <br />
  som er i en boks <br />
  med en ramme rundt
</div>
</body>
</html>
```

Mer dynamikk med DHTML

Vi skal se på enda et enkelt eksempel hvor vi bruker JavaScript til å modifisere CSS-egenskapene til to elementer når brukeren beveger musa på websiden:

```
<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 deklarert 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.

Neste uke: Mer om JavaScript