Høgskolen i Østfold

JavaScript

Et utdrag fra: http://www.ia.hiof.no/~borres/ml

Dette utdraget fra et relativt dynamisk vevsted vil alltid være litt ute av synk. Det er bare ment som litt trøst for de som vil ha noe å lese på senga før de sovner.

Sidene er produsert ved hjelp av WXT (http://www.ia.hiof.no/~borres/wxtdoc/), Python og PrinceXML (http://www.princexml.com/)

Rettighetene er de samme som for vevstedet materialet er hentet fra:

- Bruk det til hva du vil så lenge du refererer til vevstedet
- Det følger ikke med noen som helst garanti for at koden og teksten er riktig i en hver forstand.



Høgskolen i Østfold

Innhold

- JavaScript
- Litt om språket
- Objekter
- Egne Objekter
- Å lese forms
- DOM
- MSXML
- Inputkontroll
- Popup
- ting og tang
- AJAX
- Kodestrategier
- Progresjon
- JSON
- Dataformater
- Caching
- Andre domener
- JStesting
- Picasso
- Bildegalleri
- Slideshow
- Hele verden

Javascript

Hva:

JavaScript

Javascript har sine røtter hos Netscape og er definert som ECMA-script. Se ECMA-script på Wikipedia [1] og ECMA [2]

JavaScript er et objektorientert skriptingspråk. Det er ikke en "lightversjon" av Java. JavaScript brukes i hovedsak for å programmere dynamikk på vevsider. Opp gjennom årene har JavaScript blitt vurdert på forskjellig måte. Noen har dømt språket til en snarlig død, mens andre har vært mer entusiastisk. I skrivende stund ser det ut til at JavaScript har fått en ny vår. Microsoft bruker JavaScript som en integrert del av sine .Net-løsninger og det dukker opp nye utviklingssverktøy/ strategier som i stor grad er basert på JavaScript.

Årsaken til dette er vel først og fremst at alle de sentrale nettleserne støtter tilgang til to viktige programmeringsmekanismer: DOM-programmering og tilgang til import av fragmenter til en vevside. Disse mekanismene er realisert ved to objekter vi kan nå fra JavaScript: Document og XMLHttpRequest. Disse to mekanismene, hver for seg og tilsammen, muliggjør en arbeidsdeling mellom klient og tjener som legger mer ansvar på klienten. Eller for å si det på en annen måte: Vi kan lage mer selvforsynte og dynamiske vevsider. Kombinasjonen av de to mekanismene omtales ofte som AJAX (Asynchronous JavaScript And XML).

Begge mekanismene ovenfor er avhengig av JavaScripts muligheter for å plukke opp brukergenererte begivenheter som musebevegelser, tastetrykk, fokusendringer etc.

Generelt kan vi notere oss at JavaScript har en syntaks som ligner ganske mye på Java. Vi har variable, men ikke typer. Vi har arrayer og de viktigste kontrollstrukturene som if-else, repeat, for, while. Vi kan drive unntaksprogrammering, try-catch-finally.

I de følgende modulene vil vi ta opp de vanligste anvendelsene av JavaScript på vevsider.

Det finnes et svært stort antall vevsider på nettet med løsningsforslag i form av JavaScript-kode for de forunderligste problemer og finurligheter. Mange av kodeeksemplene du finner er skrevet for gamle nettlesere og er unødig kompliserte. Typisk er at de ikke bruker DOM. Vær litt kritisk til en del av denne koden. Ofte kan den både forenkles og gjøres mer robust.

Plassering av JavaScriptet

Vi kan i prinsipp plassere javaScript-kode hvor vi vil på en HTML-side. Koden skal pakkes inn i script-elementer.

```
<script type="text/javascript" type="text/javascript">
  function myfunction()
  {
    ...
  }
</script>
```

Det er ikke særlig lurt å spre skriptkoden, og normalt samler vi koden i head-elementet på en HTML-side. Fordelen er da at scriptet er på plass når resten av siden lastes og eventuelt forsøker å bruke det.

Det er ofte en god ide å legge skriptet på en fil for seg og la nettleseren hente det når siden lastes:

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

Det er to grunner til dette. Den ene grunnen er den åpenbare at vi kan vedlikeholde skriptet ett sted selv om det brukes på mange sider. Den andre grunnen er at vi sparer litt tid ved at skriptfila caches og altså ikke trenger å lastes hver gang sider lastes.

Dersom vi inkluderer skriptet på selve siden er anbefalt å maskere javaskriptkoden slik at den ikke parses som en del at det XHTML-dokumentet er plassert på. Dette kan vi gjøre slik:

```
<script type="text/javascript">
//<![CDATA[
function f()
{
  T='';
  for(ix=1; ix < 4; ix++)
    T+=ix;
  alert(T+' : kjør');
}
//]]>
</script>
```

På denne måten unngår vi at tegn som < og > i skriptet blir tolket som en del av dokumentstrukturen.

Referanser og eksempler:

Testing av Javascript: JStesting

Noen nyttige vevsteder:

- [1] ECMA-script:
  JavaScript
  en.wikipedia.org/wiki/ECMAScript
- [2] ECMA:
  www.ecma-international.org/
- Introduction to JavaScript: www.quirksmode.org/js/
- JavaScript Tutorial: www.webmonkey.com/webmonkey/programming/javascript/tutorials/jstutorial_index.html
- JavaScript Kit: www.javascriptkit.com/jsref/
- JavaScript Quick reference: www.devguru.com/Technologies/ecmascript/quickref/javascript_intro.html
- JavaScript Tutorial: www.howtocreate.co.uk/tutorials/javascript/
- Noen temaer: www.devshed.com/c/b/JavaScript/
- HTML DOM Document Object: www.w3schools.com/htmldom/dom_obj_document.asp
- Debugging Guide: javascript.internet.com/debug-guide.html
- Javascript Random: www.merlyn.demon.co.uk/js-randm.htm
- A (Re)-Introduction to JavaScript: simon.incutio.com/slides/2006/etech/javascript/js-reintroduction-notes.html
- Lyd i JavaScript: www.phon.ucl.ac.uk/home/mark/audio/play.htm
- JsUnit: www.jsunit.net/
- mochikit: mochikit.com/

Litt om JavaScript

Hva:

Noen egenskaper ved Javascript

Dette er ikke noe forsøk på en mer eller mindre komplett lærebok i eller beskrivelse av JavaScript. Hensikten er bare å peke på noen mekanismer som erfaringsmessig er litt vanskelige å få tak og som skaper litt forvirring.

Testing og debugging

Noen enkle mekanismer for å debugge:

- Alle nettlesere som er aktuelle har mulighet for å sette opp et eget vindu som rapporterer feil i JavaScript som kjøres på siden. Jeg pleier å bruke Tools - Error Console i FireFox. Du finner tilsvarende under Tools - Advanced på nye Operaversjoner.
- Det er en grei strategi å bruke alert() funksjonen for å rapportere status på variable under kjøring.

```
s=myVar.toString();
alert(s);
```

- Ellers er det mulig å skrive verdier direkte til vevsiden. Det kan være nyttig i en testfase.

```
document.write('<p>Nå er vi her</p>');
```

- Eller vi kan dumpe verdier i en node på en vevside mens vi tester. F.eks. slik:

```
function dumpArray(A)
{
  res='';
  for (ix=0;ix<A.length;ix++)
    res+=A[ix]+'<br/>';
  document.getElementById('dump').innerHTML=res;
}
```

Mer om systematisk testing av Javascript i modulen: JStesting

Variable

Globale variable i Javascript er rett fram og enkelt. Alt vi definerer utenom funksjoner er tilgjengelig i alle funksjoner. Et enkelt eksempel, myNumber er global:

```
var myNumber;

function setMyNumber(nr)
{
    myNumber=nr;
}

function showMyNumber()
{
    alert('My number is: '+myNumber);
}
```

Ved å bruke **var** foran en variable sikrer vi oss at variabelen er definert for den blokken (scope) vi er i. Det er en god vane å bruke var for å unngå å bruke globale variable med samme navn ved en misforståelse.

```
var myNumber=123;

function setMyNumber(nr)
{
    myNumber=nr;
}

function showMyNumber()
{
    var myNumber=13;
    setMyNumber(24);
    // will allways display 13
    alert('My number is: '+myNumber);
}
```

Tekst

Mye av det vi ønsker å handtere i JavaScript er typisk tekst. Tekst, string, er en klasse i Javascript. Det finnes en del standardfunksjoner for å arbeide med stringer. Siden JavaScript ikke har typer så må vi selv sørge for at det objektet vi anvender metoden på er en string.

```
S='Hallo alle sammen';
```

- ```
  c=S.charAt(0)
  -->c:'H'
  ```
- ```
  L=S.length()
  -->L:17
  ```
- ```
  i=S.indexOf('a')
  -->i:1
  ```
- ```
  i=S.lastIndexOf('a')
  -->i:12
  ```
- ```
  ordliste=S.split(' ')
  -->ordlist[0]:'Hallo'
     ordlist[1]:'alle'
     ordlist[2]:'sammen'
     ordlist.length: 3
  ```
- ```
  T=S.substring(6,9)
  -->T:'alle'
  T=S.substring(10)
  -->T:'sammen'
  ```
- ```
  t=S.toLowerCase()
  -->t: 'hallo alle sammen'
  ```
  ```

- ```
  T=S.toUpperCase()
  -->T: 'HALLO ALLE SAMMEN'
  ```

Vi kan dessuten slå sammen, concatenere, stringer med en enkel +. Dette er fleksibelt men litt farlig siden JavaScript ikke har typer og samtidig er veldig imøtekommende med andre ting en stringer.

```
a='Hallo';
b='alle';
i=3
j=4
T=a+b;
-->T:'Halloalle'
T=a+' '+b+' sammen';
-->T:'Hallo alle sammen'
T=a+' 'b+' '+i;
-->T:'Hallo alle 3'
T=a+' 'b+' '+i+j;
-->T:'Hallo alle 34'
T=a+' 'b+' '+(i+j);
-->T:'Hallo alle 7'
```

String har egenskapene length og prototype

String har metodene anchor, big, blink, bold, charAt, fixed, fontColor, fontSize, indexOf, italics, lastIndexOf, link, small, split, strike, sub, substring, sup, toLowerCase, toUpperCase, replace

replace krever litt spesiell oppmerksomhet.

```
var T='alle forstår alle';
T=T.replace('alle','noen');
-->T:'noen forstår alle'
```

Altså: bare en replace, den første forekomsten av alle byttes. For å få til en global erstatning, alle forekomster, kan vi skrive:

```
var T='alle forstår alle';
T=T.replace(/alle/g,'noen');
-->T:'noen forstår noen'
```

Vi har da brukt regulær uttrykk, vi har skrevet / is tedet for '. g angir at vi vil ha global replace. Mer om regulære uttrykk i JavaScript i [1] .

Tall

Vi har ofte behov for å konvertere fra tekst (string) til tall (integer eller float). JavaScript har to egnede funksjoner: parseInt() og parseFloat()

parseFloat(string)

Funksjonen tar en string som input og produserer et flytetall.

```
parseFloat('2.67') -> 2.67
parseFloat('2.67 meter') -> 2.67
parseFloat('0.2e6') -> 200000
parseFloat('jensen') -> NaN
```
```

Merk at ikke-numeriske verdier på slutten av en god begynnelse ignoreres, og at når funksjonen feiler returnerer den verdien NaN ("Not-a-Number"). Noen nettlesere kan levere 0 istedet for NaN. Vi kan teste om en verdi er NaN med funksjonen: isNaN(verdi), se nedenfor.

Du kan forsøke deg fram med noen konverteringer nedenfor (skriv inn en string og trykk på likhetstegnet)

parseFloat(' ') = ?

parseInt(string[,base])

Funksjonen tar en string som input og produserer et heltall. Vi kan dessuten angi hvilket tallsystem vi vil ha svaret i. 10-tallsystemet er default.

```
parseInt('12') -> 12
parseInt('12.4') -> 12
parseInt('012') -> 10
(verdier som starter på 0 tolkes som oktale)
parseInt('12',8) -> 10
parseInt('0x12') -> 18
(verdier som starter på 0x tolkes som hexadesimale)
parseInt('12',16) -> 18
parseInt('jensen') -> NaN
parseInt('14jens',5) -> 9
```

Merk igjen at ikke-numeriske verdier på slutten av en god begynnelse ignoreres, og at når funksjonen feiler returnerer den verdien NaN. Vi kan teste om en verdi er Nan med funksjonen: isNaN(verdi), se nedenfor. Noen nettlesere kan levere 0 istedet for NaN.

Du kan forsøke deg fram med noen konverteringer nedenfor (skriv inn en string og eventuelt tallsystemet du vil bruke og trykk på likhetstegnet)

parseInt(' ',) = ?

isNaN(verdi)

isNaN står for "Not-a-Number". Denne funksjonen kan brukes for å teste om en verdi er et gyldig tall, flytetall eller heltall. Vi kan teste resultatet av en divisjon, for å se om vi har fått divisjon med null, eller vi kan teste resultatet av en konvertering som vist overfor.

Merk at parameteren til isNaN er en verdi, altså hva som helst, og resultatet er en boolsk verdi (true eller false). Det går dårlig å teste på om resultatet av en konvertering er lik stringen 'NaN'.

Begge de situasjonene vi har nevnt ovenfor, divisjon med 0 og mislykket konvertering, er situasjoner som vi i andre språk vanligvis ville identifisert ved unntaksprogrammering.

Array

Array er en klasse i Javascript. Arrayer i Javascript er fleksible. Det innebærer at vi kan ekspandere en array med å tilordne verdier. Noen eksempler på initialisering av array.

```
A=new Array('Ole','Hans','Peder')

A=new Array();
A=['Ole','Hans','Peder'];

A=new Array();
A[0]='Ole';A[1]='Hans';A[2]='Peder';
```

Merk at:

```
A=new Array();
A[0]='Ole';A[1]='Hans';A[2]='Peder';
A[4]='Marit';
```

Vil gi innholdet: Ole, Hans, Peder, undefined, Marit
Arrayen har altså fått tilstrekkelig mange elementer, men ett av dem er ikke definert.

Array har egenskapen length, og metodene: join, reverse og sort xx

Metoden sort krever en kommentar. Hvis vi skriver følgende:

```
Liste=new Array(1,3,5,2,4,11,33);
Liste.sort();
```

Ender vi med følgende rekkefølge:
1,11,2,3,33,4,5
altså som om heltallene skulle vært stringer. Dersom vi ønsker å sortere numerisk kan vi gjøre slik:

```
function sequenceNumber(a,b)
{
  return a - b
}
Liste=new Array(1,3,5,2,4,11,33);
Liste.sort(sequenceNumber);
```

Dette gir oss:
1,2,3,4,5,11,33

Tidsstyring

Vi kan lett lage tidsbestemte begivenheter i JavaScript, ved å sette en forsinkelse på kall til en funksjon, setTimeout(minfunksjon,tidsforsinkelse). Tidsforsinkelsen i millisekunder.

Du kan teste en teller her 0

HTMLkoden er slik:

```
<p>
<span style="cursor:pointer" onclick="Count()">Du kan teste en teller  her</span>
<span style="font-size:24px" id="counter">0
</span>
</p>
```

JavaScriptkoden er slik:

```
function Count()
{
    elt=document.getElementById('counter');
    x=parseInt(elt.innerHTML);
    elt.innerHTML=(x+1)%10000;
    setTimeout(Count,500);
}
```

Merk at telleren øker tempo hvis vi klikker flere ganger. Du ser kanskje hvorfor ?

Vi har ikke laget noen mekanisme for å stoppe telleren. Vi har selvsagt mulighet for dette, enten ved å sette et egetdefinert flag eller ved å bruke funksjonen: clearTimeout. Denne funksjonen og funksjonene setInterval og clearInterval forutsetter at vi tar vare på en returverdi, en timer ID, fra setTimeout.

Tid og dato

JavaScript har en nyttig klasse som heter Date. Dette kan hjelpe oss å holde styr på ukedager, tidsdifferanser osv. Nedenfor finner du et eksempel på bruk av noen funksjoner i Date:

klikk her

Skriptet er slik:

```
function visFullTime(eltID){
  var today=new Date();
  var T=prettyTime(today)+'  '+norwegianDate(today);
  document.getElementById(eltID).innerHTML=T;
}


function prettyTime(dt){
  var h=dt.getHours();
  var m=dt.getMinutes();
  var s=dt.getSeconds();
  var hs=''+h;
  var ms=''+m;
  var ss=''+s;
  if (h <10) hs='0'+hs;
  if (m <10) ms='0'+ms;
  if (s <10) ss='0'+ss;
  return hs+':'+ms+':'+ss;
}


function norwegianDate(dt){
  var weekday=new Array("Søndag","Mandag","Tirsdag",
                        "Onsdag","Torsdag","Fredag","Lørdag");
  var monthName=new Array("Januar","Februar","Mars","April",
                          "Mai","Juni","Juli","August",
                          "September","Oktober","November","Desember");
  return weekday[dt.getDay()]+' '+dt.getDate()+
         '. '+monthName[dt.getMonth()]+' '+dt.getFullYear();
}
```

Date har metodene: getDate, getDay, getHours, getMinutes, getMonth, getSeconds, getTime, getTimeZoneoffset, getYear, parse, prototype, setDate, setHours, setMinutes, setMonth, setSeconds, setTime, setYear, toGMTString, toLocaleString, UTC

Math

Vi finner det vi forventer i objektet math. De vi oftest trenger er kanskje:

```
  abs(x)
  sin(x),cos(x),tan(x)
  max(x,y),min(x,y)
  ceil(x),floor(x),round(x)
  pow(x,y),exp(x),sqrt(x),log(x)
  random()
```

random-funksjonen returnerer et flytetall mellom 0 og 1. Hvis vi vil ha en funksjon som returnerer et heltall mellom to grenser a <= t <=b kan vi skrive en slik funksjon:

```
function randint(a,b){
  return a+Math.round((b-a)*Math.random());
}
```

klikk her for en randomtest

Testen ser slik ut:

```
function doRandomTest(n,a,b){
  T=''+n+ ' tester av randint('+a+','+b+'): ';
  for (var ix=0;ix < n; ix++)
    T=T+' '+randint(a,b);
  document.getElementById('randomresult').innerHTML=T;
}

function randint(a,b){
  return a+Math.round((b-a)*Math.random());
}
```

DEMO: http://www.ia.hiof.no/~borres/ml/joms/histogram.html

null og undefined

Javascript definerer to konstanter, verdier, **null** og **undefined**.

null tilsvarer det vi kjenner fra andre språk, null i java, None i python, og sier oss at en referanse referer til ingenting. null oppfører seg som false i tester og vi kan skrive:

```
function test1()
{
   var T=null;
   if(T)
    alert(''+T.length);
   else
    alert('T er ikke satt');
}
```

test her

Dersom vi hadde forsøkt å spørre etter T.length når T er null ville vi fått feil.

undefined har en litt annen oppførsel. undefined gir ikke nødvendigvis feil, men rapporterer seg selv som, nettopp: "undefined". Vi kan skrive, der string objekterer altså ikke har noen egenskap value:

```
function test2()
{
   var T='something';
   alert(T.value);
}
```

test her

undefined oppfører seg også som false i tester og vi kan skrive:

```
function test3()
{
   var T='something';
   if(T.value)
    alert(T.value);
   else
    alert('T\'s value is undefined');
}
```

test her

Det betyr at vi kan lage følgende konstruksjon:

```
function arrayTester()
{
  liste=new Array();
  liste[0]='hans';
  liste[1]='jens';
  liste[4]='kari';
  var S=''
  for(var ix=0;ix<liste.length;ix++)
    if(liste[ix])
      S+=liste[ix]+'\n';
  alert(S);
}
```

test her

Det betyr også at vi kan bruke denne mekanismen til å teste om en nettleser har de egenskapene vi er ute etter. F.eks. er en typisk kodesnutt fra en AJAX-request slik:

```
function establishRequest()
{
    var theRequest=null;
    if (window.XMLHttpRequest)
        theRequest = new XMLHttpRequest();
    else if (window.ActiveXObject)
    {
        try { theRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch(e) {
            try {theRequest= new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch(e) {theRequest=null;}
        }
    }
    else
        theRequest=null;
    return theRequest;
}
```

Her bruker vi en kombinasjon av tester på undefined (window.XMLHttpRequest og window.ActiveXObject) og try - catch. Mer om AJAX i modulen AJAX

Referanser og eksempler:

- [1] JavaScript:
  Regexp
  Pattern Matching and regular expressions
  www.webreference.com/js/column5/

Tilgjengelige objekter i Javascript

... under arbeid

Hva:

Tilgjenelige objekter i Javascript

Javascript er et objektorientert språk. Vi har vist bruken av noen slike (string, array, date, math) i modulen Litt om språket. Vi kan videre lage våre egne objekter slik som forklart i modulene: Egne Objekter og JSON.

Når vi kjører Javascript i en nettleser har vi tilgang på en rekke objekter som beskriver de omgivelsene vi opererer i. Vi skal se på noen av disse og gjøre noen eksperimenter. Merk at dette er ikke noen uttømmende beskrivelse. Det er satt fokus på de metodene som kanskje er de mest nyttige. Jeg har unngått de temaene som er vanskelige på grunn av forskjeller i de tre nettleserne (Opera, FireFox, IE).

document

document-objektet er nøkkelen til det meste av det vi gjør når vi skal manipulere en vevside. Det er document-objektet som gir oss tilgang til komponentene på en side. Alle disse komponentene gjøres tilgjengelige for oss som noder av forskjellig type og med forskjellige egenskaper. Modulen DOM viser noen enkle eksempler på programmering mot DOM (Document Objekt Model)

Egenskaper	Metoder
anchors[], forms[], images[], applets[], links[] alinkColor, area, bgColor, cookie, fgColor, lastModified, linkColor, location, referrer, title, vlinkColor	clear, close, open, write, writeln getElementById, getElementsByName, getElementsByTagName

Du kan teste og lese litt kode på siden:

DEMO: http://www.ia.hiof.no/~borres/ml/jsoop/tester1.html

Det er to temaer som peker seg ut som de viktigste i praktisk programmering. Det er DOM-programmering og det er form-programmering.

DOM får vi tak i via documentmetodene: getElementById, getElementsByTagName og hvis vi vil: getElementsByName. Hva vi så kan gjøre med elementer og elementlister er nærmere beskrevet i modulen DOM

Form-programmering kan gjøres på flere måter. Det er laget en egen modul: Å lese forms som beskriver noe av dette. Det er noen momenter som det er verdt å merke seg. Vi kan få tak i forms på flere måter. Du kan se nærmere på dette på:

DEMO: http://www.ia.hiof.no/~borres/ml/jsoop/tester2.html

Window

Egenskaper	Metoder	Begivenheter
defaultStatus, frames, opener, parent, scroll, self, status, top, window	alert, blur, close, confirm, focus, open, prompt, clearTimeout, setTimeout, setInterval, clearInterval	onload, onUnload, onblur, onFocus

De tre metodene, alert, confirm og prompt, viser alle dialogbokser.

klikk her for å kjøre følgende kode:

```
function dboxdemo()
{
   var navn=prompt('hva er navnet ditt ?');
   if (confirm('mener du virkelig dette'))
    alert('hallo ' + navn);
}
```

Vi kan lage nye vinduer, f.eks. som popup. Følgende enkle funksjon brukes på disse sidene for å poppe opp kodesnutter etc.:

```
function simplepopup(url,wname,wstyle)
{
if(wstyle=='*')
    wstyle='scrollbars=yes,resizable=yes,width=600,height=600,status=0';
try{
    newwindow=window.open(url, wname, wstyle);
    if (window.focus) {newwindow.focus()}
}
catch(E){
    alert('If you have blocked pop-ups on this page\n'+
          'Press Ctrl-key when clicking');
  }
}
```

Den kan kalles f.eks. slik:

simplepopup('tester1.html','mypopup','*')

Metodene setTimeout, clearTimeout, setInterval, clearInterval gjør det mulig å controllere tidsbestemte handlinger. Vi kan f.eks. starte og stoppe en teller en teller.

start

stopp

Den involverte koden ser slik ut:

```
var timerID=-1;
function startTimer()
{
   if (timerID!=-1)
        clearTimeout(timerID);
   timerID=setInterval(counter,400);
}
function stoppTimer()
{
   if (timerID!=-1)
     clearTimeout(timerID);
   else
    alert('ingenting startet -> ingenting kan stoppes');
}
function counter()
{
  var val=parseInt(document.getElementById('counter').innerHTML);
  if (isNaN(val))
    val=0;
  document.getElementById('counter').innerHTML=''+(val+1)%10000;
}
```

Du kan jo forsøke å kjøre følgende kode:

```
function changeWindow()
{
  moveBy(50, 50);
  scrollBy(10, 10)
  //resizeTo(800,800);
}
```

klikk her

Location

Vi får tak i location objektet via window.

Egenskaper	Metoder
hash, host, hostname, href, pathname, port, protocol, search	reload, replace

Prøv denne koden

```
function showLocation()
{
  S='';
  var loc=window.location;
  S+='hash: '+loc.hash+'<br/>';
  S+='host: '+loc.host+'<br/>';
  S+='href: '+loc.href+'<br/>';
  S+='pathname: '+loc.pathname+'<br/>';
  S+='port: '+loc.port+'<br/>';
  S+='protocol: '+loc.protocol+'<br/>';
  S+='search: '+loc.search+'<br/>';
  document.getElementById('location_result').innerHTML=S;
}
```

test

Og du kan reloade: location.reload(true)

History

Historyobjektet er tilgjengelig via window.

Egenskaper	Metoder
length, forward, go	back

Prøv denne koden

```
function showHistory()
{
  var S='Her finnes en historie: '+window.history.length;
  document.getElementById('history_result').innerHTML=S;
}
```

test

-

Vi kan gå f.eks. bakover i historien:

```
    <a href="javascript:history.go(-1)">tilbake</a>
```

men vi har ikke lov (rettigheter) til å vise fram historien. Vi kan ikke inspisere hvor en bruker har vært. Det har vi ikke noe med.

Navigator

Egenskaper	Metoder
plugins[], appCodeName, appName, appVersion, mimeTypes, userAgent	javaEnabled

Prøv denne koden

```
function showNavigator()
{
  var S='';
  S+='userAgent: '+navigator.userAgent+'<br/>';
  S+='appVersion: '+navigator.appVersion+'<br/>';
  S+='appName: '+navigator.appName+'<br/>';
  var plugs=navigator.plugins;
  if(plugs.length==0)
    S+='Finner ingen plugins, kan skyldes denne versjonen av  MS-IE';
  else
  {
    S+='plugins: '+plugs.length+'<br/>';
    for(var ix=0;ix<plugs.length;ix++)
    {
      S+='   '+plugs[ix].name+'<br/>';
      S+='         '+plugs[ix].filename+'<br/>';
      S+='         '+plugs[ix].description+'<br/>';
    }
  }

  document.getElementById('navigator_result').innerHTML=S;
}
```

test

image

Vi har sett litt på images på denne siden:

DEMO: http://www.ia.hiof.no/~borres/ml/jsoop/tester1.html

Egenskaper	Metoder
border, complete, height, hspace, lowsrc, name, src, vspace, width	-

form

Vi har sett litt på forms på disse sidene:

DEMO: http://www.ia.hiof.no/~borres/ml/jsoop/tester1.html

DEMO: http://www.ia.hiof.no/~borres/ml/jsoop/tester2.html

Egenskaper	Metoder	Begivenheter
action, elements, encoding, FileUpload, method, name, target	submit, reset	onSubmit, onReset

text

text i betydningen inputelementer av typen text

Egenskaper	Metoder	Begivenheter
defaultValue, name, type, value	focus, blur, select	onBlur, onChange, onfocus, onSelect

Frame

Egenskaper	Metoder
defaultStatus, frames, opener, parent, scroll, self, status, top, window	alert, blur, close, confirm, focus, open, prompt, clearTimeout, setTimeout

En inspektør

Siden inspectdoc.html muliggjør en ganske generell inspeksjon av noen viktige objekter i nettleseren.

DEMO: http://www.ia.hiof.no/~borres/ml/jsoop/inspectdoc.html

Siden inspectdoc2.html gjør det mulig å inspisere identifiserbare elementer på en side.

DEMO: http://www.ia.hiof.no/~borres/ml/jsoop/inspectdoc2.html

Referanser og eksempler:

- Firebug. En plug-in for FireFox som støtter utvikling av dynamiske vevsider:
  www.getfirebug.com/
  Videopresentasjon av Firebug: yuiblog.com/blog/2007/01/26/video-hewitt-firebug/
- JavaScript Reference: www.javascriptkit.com/jsref/
- Mer om plugins: javascript.internet.com/user-details/plugins.html
- JavaScript DOM (metoder og egenskaper): krook.org/jsdom/

Egne objekter i Javascript

Hva:

Egne objekter i Javascript

JavaScript er et objektorientert språk og vi kan lage våre egne objekter. Det er en del forhold som skiller objektbegrepet i JavaScript fra det vi kjenner i f.eks. Java. JavaScript er interpretert og JavaScript har ikke et sterkt typebegrep. Disse grunnleggende forskjellene manifesterer seg på mange måter, i syntaks og i "objekters livsløp".

Nedenfor skal vi se litt på noen av de sentrale karakteristika ved objekter i JavaScript. Hensikten er primært å gjøre objektorientert programmering tilgjengelig som praktisk verktøy i JavaScript.

Enkelt objekt

Vi begynner med et enkelt eksempel på hvordan vi kan opprette et objekt etter et resonnement som leder oss fram til en form som er funksjonelt sammenlignbar med det vi finner i andre, ikke-interpreterbare, språk som Java.

Anta:

```
function person()
{
    var navn='ole';
    var adresse='halden';
}
```

Hvis vi nå skriver følgende:

```
var naboen = new person();
```

har vi satt opp et nytt object, naboen. Merk at vi altså oppretter objektet ved å kalle en funksjon. Funksjonskallet fungerer både som konstruktør og objektbeskrivelse, og det returnerer et objekt. De to egenskapene, navn og adresse, er lokale og vi har ikke stor glede av naboen. Dersom vi forsøker å få tak i naboen.navn, får vi beskjed om at dette er undefined. en **var** deklarasjon avgrenser variabelene til den lokale blokken, i dette tilfellet funksjonen eller, om vi vil, objektet.

Vi skriver om funksjonen person med tanke på at vi vil ha tilgang til egenskapene.

```
function person()
{
    this.navn='ole';
    this.adresse='halden';
}
```

dersom vi nå oppretter et objekt:

```
var naboen = new person();
```

kan vi skrive naboen.navn og få ole.

Vi kan videre lage følgende:

```
function person(nvn,adr)
{
    this.navn=navn;
    this.adresse=adr;
}
```

dersom vi nå oppretter et objekt:

```
var naboen = new person('Hans','Moss');
```

og naboen.adresse gir oss Moss.

Vi tar eksempelet et steg videre og forsøker å lage en konstruksjon med lokale egenskaper og synlige, public, metoder.

```
function person(nvn,adr)
{
    var navn=nvn;
    var adresse=adr;
    this.getNavn = function(){return navn}
    this.getAdresse = function(){return adresse}
}
```

dersom vi nå oppretter et objekt:

```
var naboen = new person('Jens','Sarp');
```

og naboen.getNavn() gir oss Jens.

Dersom vi lager følgende konstruksjon så har vi en ganske vanlig form, slik vi ofte ordner oss i Java, med lokale egenskaper og åpne metoder.

```
function person(nvn,adr)
{
    var navn=nvn;
    var adresse=adr;
    this.getNavn = function(){return navn}
    this.getAdresse = function(){return adresse}
    this.setNavn = function(nvn){navn=nvn}
    this.setAdresse = function(adr){adresse=adr}
}
```

Alternativt

Vi kan skrive følgende direkte:

```
var minbil = new Object();
minbil.merke = 'toyota';
minbil.farge = 'blå';
```

Det vil si at vi tilordner nye egenskaper til et nytt generelt objekt. Og vi kan fortsette med å plante en funksjon:

```
minbil.visfram = new function()
                  {
                    var vis=function()
                    {
                      return 'Det er en '+this.farge+' '+this.merke;
                    }
                    return vis;
                  };
```

Setningen:

```
minbil.visfram();
```

Vil gi resultatet: "Det er en blå toyota"

Dette viser et par interessante egenskaper ved Javascript. Den mekanismen som gjør at vi kan "plante" en funksjon kalles "closure". Det innebærer at funksjonen visfram kalles, den eksekveres og den returnerer en referanse til den interne funksjonen vis. Det er denne funksjonen, vis, som kjører når vi senere kaller visfram(). Denne mekanismen er nyttig og brukes utstrakt i ikke-trivielle javascriptbiblioteker. Blandt annet er denne mekanismen involvert i det minibiblioteket som er introdusert for enkel AJAX i modulen: Kodestrategier

Eksempelet ovenfor indikerer at vi skal kunne ta for oss et vilket som helst objekt, f.eks. i et bibliotek, og utvide objektet med nye egenskaper og metoder. Vi kan lager to person-objekter og gjør en av dem til bileier:

```
function person(nvn,adr)
{
    this.navn=nvn;
    this.adresse=adr;
    this.visfram=function(){return this.navn+' fra '+this.adresse;}
}

var person1=new person('Kristian','Askim');
var person2=new person5('Ole','Mysen');
person2.bilmerke='Ford';
person2.visfram =
  new function()
     {
          var vis=function()
          {
               return this.navn+' fra '+this.adresse+' kjører '+this.bilmerke;
          }
          return vis;
     };
alert(person1.visfram())
alert(person2.visfram())
```

De to alertene gir hhv: "Kristian fra Askim" og "Ole fra Mysen kjører Ford".

Siden tester3.html viser hvordan vi kan plante og bruke en ny egenskap ved bildeobjekter, altså document.images.

DEMO: http://www.ia.hiof.no/~borres/ml/jseoop/tester3.html

Den sentrale koden i eksempelet er:

```
var imList=null;
function setCopyRight()
{
    return function()
    {
        return this.src+'<br/>CopyRight: '+this.owner;
    }
}
function prepareImages(navn)
{
    imList=document.images;
    for(var ix=0;ix < imList.length;ix++)
    {
        imList[ix].owner=navn;
        imList[ix].copyRight=new setCopyRight();
    }
}
```

Prototype og arv

Alle funksjoner, function-objekter, har en egenskap som heter **prototype**. Denne kan vi bruke til mye rart.

eksempel1

Vi kan lage følgende enkle konstruksjon:

```
function land(navn)
{
 this.land=navn;
}

function sted(hvor)
{
 this.by=hvor;
}

var byliste=new Array();

sted.prototype=new land('norge');
byliste[0]=new sted('moss');
byliste[1]=new sted('halden');

sted.prototype=new land('sverige');
byliste[2]=new sted('uddevalla');

sted.prototype=new land('danmark');
byliste[3]=new sted('aalborg');
```

En utskrift av byliste, byliste[ix].by+' i '+byliste[ix].land, vil gi oss

```
moss i norge
halden i norge
uddevalla i sverige
aalborg i danmark
```

Greitt nok, men gevisnten er ikke stor. Vi ville oppnådd akkurat det samme med å la landsnavnet være en egenskap ved stedet i en flat klassestruktur.

eksempel 2

La oss se på hvordan vi kan subklasse et land til republikk eller kongedømme:

```
function land(nvn)
{
 this.navn=nvn;
 this.show=function(){return this.navn};
}
function republikk(nvn)
{
 this.prototype=new land(nvn);
 this.show=function()
     {
        return this.prototype.show()+ ' som er en republikk'
     };
 }
function kongerike(nvn)
{
 this.prototype=new land(nvn);
 this.show=function()
     {
        return this.prototype.show()+ ' som er et kongedømme'
     };
}

var landliste=new Array();

landliste[0]=new kongerike('norge');
landliste[1]=new kongerike('danmark');

landliste[2]=new republikk('frankrike');
landliste[3]=new republikk('italia');
```

Som gir (landliste[ix].show()):

```
norge som er et kongedømme
danmark som er et kongedømme
frankrike som er en republikk
italia som er en republikk
```

Vi kunne oppnådd det samme ved å skrive:

```
function land(nvn)
 {
   this.navn=nvn;
   this.show=function(){return this.navn};
 }
 function republikk(nvn)
 {
   this.prototype=republikk.prototype;
   this.navn=nvn;
   this.show=function()
      {
         return prototype.navn+' som er en republikk'
      };
 }
 function kongerike(nvn)
 {
   this.prototype=kongerike.prototype;
   this.navn=nvn;
   this.show=function()
      {
         return prototype.navn+' som er et kongedømme'
      };
 }

 var landliste=new Array();

 kongerike.prototype=new land();
 republikk.prototype=new land();

 landliste[0]=new kongerike('norge');
 landliste[1]=new kongerike('danmark');

 landliste[2]=new republikk('frankrike');
 landliste[3]=new republikk('italia');
```

Vi ser at dette åpner for at vi dynamisk kan endre hva som er prototype for en klasse, altså hvilken klasse vi ønsker å subklasse.

eksempel 3

Vi kan ta for oss personer og bileiere. Vi tenker oss altså at bileier skal være en subklasse av person. Vi kan da lage følgende konstruksjon:

```
function person(nvn,adr)
{
    this.navn=nvn;
    this.adresse=adr;
    this.show=function(){return this.navn+' fra '+this.adresse}
}
function bileier(mrk,frg,nvn,adr)
{
    this.merke=mrk;
    this.farge=frg;
    this.prototype=new person(nvn,adr);
    this.show=function()
    {
      return this.prototype.show()+' kjører en ' +this.farge+' '+this.merke
    }
}
```

og vi kan bruke det til å lage en array av personer, der noen er bileiere, slik:

```
var liste=new Array();
liste[1]=new bileier('toyota','blå','ole','moss');
liste[2]=new bileier('ford','gul','hans','mysen');
liste[3]=new bileier('vw','grønn','gudrun','marker');
liste[3].prototype=new person('kristian','sarpsborg');
liste[4]=new bileier('opel','hvit');
liste[4].prototype=new person('marit','askim');
liste[5]=new person('sverre','rakkestad');
```

En utskrift av lista, liste[i].show(), vil gi følgende:

```
ole fra moss kjører en blå toyota
hans fra mysen kjører en gul ford
kristian fra sarpsborg kjører en grønn vw
marit fra askim kjører en hvit opel
sverre fra rakkestad
```

Merk liste[3], som "bytter eier", altså får ny person. I andre språk ville vi vel kalle dette dynamisk subklassing.

Merk også liste[4], som etableres uten eier. Vi ser litt nærmere på dette. Hvis vi skriver:

```
p=new bileier('opel','hvit');
alert(p.show());
p.prototype=new person('marit','askim');
alert(p.show());
```

vil vi i tur og orden få:

```
 undefined fra undefined kjører en hvit opel
 marit fra askim kjører en hvit opel
```

JSON

En drøfting av praktisk bruk av objektorientert javaskripting må ta med JSON (JavaScript Object Notation) som en viktig komponent. I dette materialet er JSON beskrevet i en egen modul, JSON. Der er fokus satt på JSON som en effektiv måte å overføre og tolke data i forbindelse med AJAX-løsninger.

Referanser og eksempler:

JSON: www.json.org/

Å lese data fra forms

Hva:

Hvordan plukke verdier fra forms

Vi skal se litt på stragier for å plukke opp data fra input-felter i forms på en HTML-side. Det kan være flere årsaker til at vi ønsker å inspisere slike data:

- Vi ønsker å kontrollere det brukeren har skrevet før data går til tjeneren på vanlig måte.
- Vi ønsker å kontrollere det brukeren har skrevet før data går til en AJAX-request.
- Vi ønsker å bruke en form for å samle opp data som skal "ferdigbehandles" i Javascript.
- Vi ønsker å bruke Javascript til å samle sammen data fra en HTML-side ved hjelp av DOM-programmering (getElementById().innerHTML) og pakke dem i et skjult (hidden) input felt i en form før de blir sendt.

I alle eksemplene nedenfor brukes et enkelt Pythonskript på serveren for å gi respons. Dette skriptet, http://www.ia.hiof.no/~borres/cgi-bin/demo/std.py, plukker ut alle parameterne og returnerer parameter og verdi pakket inn i en minimalistisk HTML-side. Du kan se scriptet her: std_py.html

Exsempel 1

Den grunnleggende strukturen når vi ønsker å sende data til tjeneren er at vi formulerer en form med input-felter og en submit-knapp. Formen skal ha en action-property og en method-property. F.eks. slik:

Melding:
Send

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/demo/std.py"
      method="get">
      <div>
      Melding: <br/><input type="text" name="message" size="5"/>
      <input type="submit" value="Send"/>
      </div>
</form>
```

Innholdet i form-elementet går ubetinget til tjeneren.

Hva hvis vi ønsker å gjøre noe med disse dataene før de sendes ? Vi kan bryte automatikken mellom et submit-klikk og en action ved å plukke opp museklikket på submit.knappen. F.eks. slik:

Exsempel 2

Melding:
Send

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/demo/std.py"
      method="get">
      <div>
      Melding: <br/><input type="text" name="message" size="5"/>
      <input type="submit" onclick="myFunction()" value="Send"/>
      </div>
</form>
```

myScript er et Javascript-kode vi skriver. f.eks. slik:

```
function myFunction(){
    alert('hello');
}
```

Vi modifiserer HTML-koden slik at vi eksplisitt returnerer en verdi fra funksjonen vår, og vi sender med den aktuelle formen vi er inne i som parameter

Exsempel 3

Melding:
Send

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/demo/std.py"
      method="get">
      <div>
      Melding: <br/><input type="text" name="message" size="5"/>
      <input type="submit"
             onclick="return myFunction1(this.form)"
             value="Send"/>
      </div>
</form>
```

Og vi modifiserer funksjonen slik:

```
function myFunction1(form){
    T=form.message.value;
    if(T.length == 0){
        alert('Du har ikke skrevet noe');
        return false;
    }
    return true;
}
```

Da har vi oppnådd et minimum av inputkontroll i og med at vi ikke sender dataene dersom message-feltet (som er det eneste innholdbærende feltet) er tomt. Formuleringen over er generelt ganske nyttig og vanlig ved input kontroll. Det vi altså gjør er å hindre submit med å returnere false fra vår funksjon.

La oss nå anta at vi ønsker "å sende med" data fra elementer i HTML-siden som ikke er plassert inne i formen. F.eks. slik:

Exsempel 4

- RBK
- Brann

Melding:
Send

```
<ul>
<li id="nr1">RBK</li>
<li id="nr2">Brann</li>
</ul>

<form action="http://www.ia.hiof.no/~borres/cgi-bin/demo/std.py"
      method="get">
     <div>
      Melding: <br/><input type="text" name="message" size="5"/>
      <input type="hidden" name="lagliste"/>
      <input type="submit"
             onclick="return myFunction2(this.form)"
             value="Send"/>
      </div>
</form>
```

Og vi modifiserer funksjonen slik:

```
function myFunction2(form){
    T=form.message.value;
    if(T.length == 0){
        alert('Du har ikke skrevet noe');
        return false;
    }
    var S='';
    for (ix=1;ix!=3;ix++)
        S+=document.getElementById('nr'+ix).innerHTML+',';
    form.lagliste.value=S;
    return true;
}
```

Da har altså hentet data fra andre områder, pakket dem sammen og så og si: "lagt dem på lasset". De dataene vi henter på den måten kan være avhengig av hva vi har skrevet, eller valgt, i andre input-felter og de kan ha blitt plassert ymse steder av det skriptet som kanskje opprinnelig laget den siden vi jobber på.

Det er en kjent problemstilling både i programmering av dialogbokser i vanlige programmer og i handtering av forms på vevsider at bruk av enter-tasten fra brukeren fungerer automatisk som om brukeren hadde klikket på "default"-knappen, "ok" i dialogbokser eller "submit" i forms. Dette oppfattes av mange som standard oppførsel og bør derfor ikke endres. Andre finner det irriterende at et "utilsiktet" enter-trykk skal føre til avslutning av en dialogsituasjon.

Dette lar seg kontrollere, også på forms.

Exsempel 5

I en form kan vi plukke opp tastetrykk og behandle dem som vi måtte ønske.

Skriv i de tre feltene nedenfor og se at de oppfører seg forskjellig når du bruker "enter"-tasten.




```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="post"
      name="formq">
   <div>
<textarea name="message" cols="10" rows="5">
</textarea>
<br/>
<input type="text" name="single1" size="10"/>
<br/>
<input type="text" name="single2" size="10"/>
<br/>
<input type="submit" value="Send"/>
   </div>
</form>
```

Den involverte Javascriptkoden ser slik ut. Merk at funksjonen startup kalles slik <body onload="startup()">

```
function startup(){
  document.forms["formq"].single1.onkeypress=checkKey;
}

function checkKey(evt) {
  var evt = (evt) ? evt : event
  var charCode = (evt.which) ? evt.which : evt.keyCode
  if (charCode == 13) {
    return false;
    // or what ever we want to do
    // form validation is a reasonable task
  }
}
```

Vi kan trikse litt med selve submit-mekanismen. Skriv i de tre feltene nedenfor og se at ingen av dem fører til "submit" når vi trykker "enter"-tasten.

Exsempel 6




```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="post"
      onSubmit="return false"
      name="formx">
   <div>
<textarea name="message" cols="10" rows="5">
</textarea>
<br/>
<input type="text" name="single1" size="10"/>
<br/>
<input type="text" name="single2" size="10"/>
<br/>
<input type="button" value="Send"
       onClick="document.formx.submit()"/>
   </div>
</form>
```

Referanser og eksempler:

QuirksMode drøfter eventmodellen i Javascript: www.quirksmode.org/js/

DOM i Javascript

Hva:

DOM-programmering i Javascript (Document Object Model)

Vi har opp gjennom HTMLs historie kunnet manipulere innholdt på en side i selve nettleseren ved hjelp av scriptspråk, vanligvis Javascript. Dette har vanligvis blitt omtalt som DHTML (dynamisk HTML). Dette har vært en ganske slitsom aktivitet fordi nettleserne har hatt forskjellige oppfatninger av hvordan en vevside er beskrevet, altså vilke deler siden består av og hvordan de kan leses og endres.

Det ser nå ut til at nettleserprodusentene har har tatt inn over seg at velformede dokumenter skal kunne handteres ved hjelp av et DOM-grensesnitt slik som DOM er beskrevet fra W3C. Det betyr et langt steg i retning av at dokumenter betraktes som DOM-trær og at treet kan leses og manipuleres med funksjoner som er beskrevet som del av DOM-anbefalingen. Vi er ikke helt framme ved en felles måte å gjøre ting på, men det er bedre enn det har vært. Et eksempel på et problem ar at Microsoft og Mozilla manipulerer style-attributter på forskjellig måte.

Merk Microsofts generelle XML-maskin, MSXML, som gjør det mulig å anvende DOM-funksjonalitet på andre dokumenter enn det som er lastet opp som "synlig" deler av en side. Blandt annet kan vi via Javascript anvende XSLT-transformasjoner på XML-data. MSXML beskrives i modulen MSXML.

Eksemplene i denne modulen er dersom ikke annet er angitt testet i de seneste versjonene av MSIE, Mozilla og Opera, alle på Windows.

Tell paragrafer

La oss se på et enkelt eksempel: Vi ønsker å telle antall paragrafer, p-elementer, på denne siden og vise resultatet i en alertboks. Prøv dette:

Klikk her for å vise antall paragrafer.

```
<p style="cursor:pointer" onclick="countCode('p');">
 Klikk her for å vise antall paragrafer.
</p>
```

Javascriptkoden er svært enkelt og bruker bare en DOM-metode.

```
function countCode(tagname)
{
   paralist=document.getElementsByTagName(tagname);
   alert("antall "+tagname+": "+paralist.length);
 }
```

Produksjon av innhold

La oss produsere en elementstruktur fra et javascript. Vi vil legge til litt formatert text i paragrafen nedenfor.

Dette er originalteksten.

Klikk her for å vise mer.

```
<p id="makemore">
   Dette er originalteksten.
</p>
<p style="cursor:pointer" onclick="makeMore('makemore');">
 Klikk her for å vise mer.
</p>
```

Javascriptkoden er slik:

```
function makeMore(hostID)
{
   hostelement=document.getElementById(hostID);
   newelt=document.createElement('div');
   newtext=document.createTextNode('... og her er mer');
   newelt.style.color='red';
   newelt.appendChild(newtext);
   hostelement.appendChild(newelt);
}
```

Bytte innhold i elementer

Dette er en ganske generell problemstilling som kan tjene som illustrasjon av noe av den mest sentrale funskjonaliteten. Vi ønsker altså å bytte innholdet i to elementer som respons på en eller annen brukeraksjon. Du kan test ut eksempelet her.

Hallo
Hei

Klikk her for å bytte linjene

De involverte linjene i HTML-siden er slik:

```
<div id="linje1"><span style="color:red">Hallo</span></div>
<div id="linje2"><span style="color:green;font-size:20;">Hei</span></div>
<div id="temp" style="display:none">her</div>

<p style="cursor:pointer" onclick="switchElements('linje1','linje2');">
Klikk her for å bytte linjene
</p>
```

Javascriptet er slik:

```
function copyFragment(fromNode,toNode)
{
   while(toNode.hasChildNodes())
      toNode.removeChild(toNode.lastChild);

   nod=fromNode.firstChild;
   while(nod!=null)
   {
      nodcopy=nod.cloneNode(true);
      toNode.appendChild(nodcopy);
      nod=nod.nextSibling;
   }
}

function switchElements(elt1ID,elt2ID)
{
            element1=document.getElementById(elt1ID);
            element2=document.getElementById(elt2ID);
            elementtemp=document.getElementById('temp');
            copyFragment(element1,elementtemp);
            copyFragment(element2,element1);
            copyFragment(elementtemp,element2);
}
```

Vi har altså brukt et skjult hjelpeelement, div med id="temp", for å foreta den nødvendige mellomlagringen når vi bytter (Vi kunne kanskje gjort det annerledes med litt mer sofistikert nodeadministrasjon).

Dette er ren DOM-kode ifølge boka. Siden begge de involverte div'ene har en understruktur er det ikke godt nok å kopiere det tekstelementet som er barn av selve div-elementet, vi må kopiere et helt trefragment. Merk spesielt funksjonen getElementById som plukker ut det første elementet i treet med den angitte id-attributten. Attributten id spiller altså en spesiell rolle og er koplet til en egen DOM-funksjon.

Det finnes en kortere versjon av scriptet som baserer seg på den omstridte, ikke DOM, egenskapen innerHTML. Hvis vi bruker denne kan koden vår se slik ut:

```
function switchElements(elt1ID,elt2ID)
{
    element1=document.getElementById(elt1ID);
    element2=document.getElementById(elt2ID);
    temptext=element1.innerHTML;
    element1.innerHTML=element2.innerHTML;
    element2.innerHTML=temptext;
}
```

Unektelig noe enklere, men altså ikke ifølge W3C-standard. innerHTML bryr seg ikke om hva slags trestruktur vi flytter, den fokuserer bare på innholdet i elementet, betraktet som text. De fleste nettlesere har implementert innerHTML. Hvis du søker på nettet finner du mange kommentarer til denne "innerHTML-fiksen".

Bilder

Vi kan bruke den samme teknikken for bilder. Prøv å føre musa inn og ut av dette bildet og se hva som skjer.



Den aktuelle koden er slik:

```
<div id="div1" style="cursor:pointer">
   <img src="Smileys17.gif" alt="image1"
   onmouseover="switchElements('div1','div2');"/>
</div>

<div id="div2" style="display:none">
   <img src="Smileys12.gif" alt="image2"
   onmouseout="switchElements('div1','div2');"/>
</div>
```

Alternativt kan vi gjøre slik hvis vi f.eks. ønsker å markere en popup. Prøv å klikk på bildet.



```
<div id="div11" style="cursor:pointer">
   <img src="Smileys17.gif" alt="image1"
   onmouseover="switchElements('div11','div12');"/>
</div>

<div id="div12" style="display:none">
   <img src="Smileys12.gif" alt="image2"
   onmouseout="switchElements('div11','div12');"
   onclick="simplepopup('http://www.ia.hiof.no/~borres/ml/domscript/p-smileys.html','Pop','*'
</div>
```

Skjulte elementer

Vi ser at en teknikk som er bruk ovenfor er å legge ut deler av vevsiden med **style="display:none"**, for så å flytte innholdet i et slikt skjult element til et element som ikke er skjult. Og/eller omvendt. Dette åpner i prinsipp for at vi kan ommøblere en vevside nesten som vi vil med en enkel "switchElements()" eller "copyElements()" funksjon.

To eksempler

DEMO: http://www.ia.hiof.no/~borres/ml/domscript/demo1.html

DEMO: http://www.ia.hiof.no/~borres/ml/picasso/picasso_hidden.html

Referanser og eksempler:

JavaScript DOM (metoder og egenskaper): krook.org/jsdom/

DOM-scripting

- i MS Internet Explorer.
  msdn.microsoft.com/workshop/author/om/doc_object.asp
- i Mozilla.
  books.mozdev.org/html/mozilla-chp-5-sect-2.html
- i Opera
  www.opera.com/docs/specs/

Scripting mot MSXML

...fordrer MSIE 5+ og MSXML (helst versjon 4) installert

Hva:

DOM-programmering med Microsofts MSXML

Vis DOM-treet

Se på følgende to filer:

Råmaterialet til Olympiade eksempelet	
Som XML: all_results.xml	som DOM-tre i en Alert-boks.
Råmaterialet til Bok eksempelet	
Som XML: bokliste.xml	som DOM-tre i en Alert-boks.

Koden som viser DOM-treet i en Alert-boks er slik:

```
function showURL(adr)
{
  try
  {
    var objDOM = new ActiveXObject("Msxml2.DOMDocument");
    objDOM.async = false;
    if (objDOM.load(adr))
      alert(objDOM.xml);
    else
      alert("Error: " + objDOM.parseError.reason);
  }
  catch(e)
  {
    alert("Error: " + e);
  }
}
```

Merk linja:

```
var objDOM = new ActiveXObject("Msxml2.DOMDocument");
```

Det er her vi involverer MSXML. MSXML kommer i flere varianter. Vi kan navngi versjonen dersom vi ønsker en bestemt versjon, f.eks.:

```
var objDOM = new ActiveXObject("Msxml2.DOMDocument.4.0");
```

Merk at vi kan ha flere versjoner installert samtidig, og de kan kjøre . Hvis du lurer på hvilken versjon du har på din maskin så prøv denne siden: www.bayes.co.uk/xml/index.xml?/xml/utils/msxml_sniffer.htm

XSLT-transformasjon

Vi vet at MSIE kan foreta XSLT-transformasjoner "on-the-fly" på klientsiden når vi laster opp en XML-fil med referanse til en XSL- eller XSLT-fil. I dette eksempelet skal vi se at vi også kan programmere en XSLT-transformasjon.

Vi bruker olympiaderesultatene: all_results.xml , og skriver en transformasjon som plukker ut 100m resultatene:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" version="1.0"
               encoding="ISO-8859-1"
               indent="yes"/>

<xsl:template match="/">
   <xsl:apply-templates select="IOC/OlympicGame"/>
</xsl:template>

<xsl:template match="OlympicGame">
   <h2><xsl:value-of select="@place"/></h2>
  <xsl:apply-templates select="event[@track='100m']"/>
</xsl:template>

<xsl:template match="event">
   <xsl:apply-templates select="athlet">
      <xsl:sort select="result" order="ascending"/>
   </xsl:apply-templates>
</xsl:template>

<xsl:template match="athlet">
   <div>
   <xsl:value-of select="result"/> : <xsl:value-of select="name"/>
   </div>
</xsl:template>

</xsl:stylesheet>
```

HTML-fila som drar det hele, kun100.html, er i sin helhet slik:

```
<html>
   <head>
      <title>Olympic 100m results</title>
   </head>
<body onload="init()">
<h2>Olympiske 100m resultater</h2>

<!-- we will place the result from the transformation here -->
<div id="displayDiv"></div>

<script language="javascript">

// on load
function init()
{
   // load the XML-file
   domDoc1=new ActiveXObject("Msxml2.DOMDocument")
   domDoc1.async=false
   domDoc1.load(
    "http://www.ia.hiof.no/~borres/commondata/olympiade/all_results.xml")

   // load the transformation
   domDocForXsl=new ActiveXObject("Msxml2.FreeThreadedDOMDocument")
   domDocForXsl.async=false
   domDocForXsl.load("kun100.xslt")

   // XSL Transformation
   xslTpl = new ActiveXObject("Msxml2.XSLTemplate")
   xslTpl.stylesheet = domDocForXsl
   var xslProc = xslTpl.createProcessor()
   xslProc.input = domDoc1

   // do the transformation
   xslProc.transform
   htmlCode = xslProc.output
   // it is too tempting to do this a non-DOM way
   // innerHTML is not W3C, but widespread in browsers
   displayDiv.innerHTML = htmlCode
}

</script>
</body>
</html>
```

Vi ser at transfomasjonene kalles når XML-fila lastes opp, så vi oppnår ikke noe annet enn det vi ville gjøre ved å legge en referanse til XSLT-fila i XML-fila.

DEMO: http://www.ia.hiof.no/~borres/ml/dommsxml/kun100.html

Parameterstyrt XSLT

Det ville være svært tungvindt om vi ikke kunne parameterstyre XSLT-transformasjoner. Det ville innbære at vi måtte skrive en egen transformasjon for alle operasjoner vi skulle gjøre. Parameterstyring er en mulighet i alle XSLT-biblioteker, og vi finner slike i Java (f.eks.: Xalan), Python (f.eks.: 4suite) og i C#-biblioteker.Nå skal vi forsøke og programmere dette på kientsiden, ved hjelp av DOM, MSXML og Javascript.

Vi bruker olympiaderesultatene: all_results.xml , og skriver en XSLT-transformasjon som ser slik ut:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<!-- the global parameters that will be nmanipulated -->
<xsl:param name="where"/>
<xsl:param name="what"/>

<xsl:template match="/">
   <xsl:apply-templates select="IOC/OlympicGame[@place=$where]"/>
</xsl:template>

<xsl:template match="OlympicGame">
   <xsl:apply-templates select="event[@track=$what]"/>
</xsl:template>

<xsl:template match="event">
   <xsl:apply-templates select="athlet">
      <xsl:sort select="result" order="ascending"/>
   </xsl:apply-templates>
</xsl:template>

<xsl:template match="athlet">
   <div>
   <xsl:value-of select="result"/> : <xsl:value-of select="name"/>
   </div>
</xsl:template>

</xsl:stylesheet>
```

Merk at det er ingen referanse til XSLT-fila i XML-fila. Vi skal sette opp denne koplingen i Java scriptkoden. I tillegg trenger vi en HTML-fil som skal huse koden vår og vise transformasjonsresultater. Du kan starte denne her og se på resultatet. service2.html

Fila service2.html er i sin helhet slik:

```
<html>
   <head>
      <title>Olympic results</title>
   </head>
<body onload="init()">
<h2>Olympiske sprintresultater</h2>
<hr/>
<table><tr>
<td>
 Olympiade: <select name="OLSelect" size="1" onchange="select()">
     <option value="Barcelona">Barcelona</option>
     <option value="Atlanta">Atlanta</option>
     <option value="Atlanta">Sidney</option></select>
</td><td>
 Øvelse: <select name="EVSelect" size="1" onchange="select()">
     <option value="100m">100m</option>
     <option value="200m">200m</option>
     <option value="400m">400m</option></select>
</td>
</tr></table>

 <hr/>

<!-- we will place the result from the transformation here -->
<div id="displayDiv"></div>

<script language="javascript">
var domDoc1
var domDocForXsl

// on first time load
function init()
{
   // load the XML-file
   domDoc1=new ActiveXObject("Msxml2.DOMDocument")
   domDoc1.async=false
   domDoc1.load(
     "http://www.ia.hiof.no/~borres/commondata/olympiade/all_results.xml")

   // load the transformation
   domDocForXsl=new ActiveXObject("Msxml2.FreeThreadedDOMDocument")
   domDocForXsl.async=false
   domDocForXsl.load("choose.xslt")

   // show default
   select();
}

// user respons on select
function select()
{
   // what user has selected
   var index = OLSelect.selectedIndex;
   var selectedOL = OLSelect.options(index).text;
   index = EVSelect.selectedIndex;
   var selectedEV = EVSelect.options(index).text;

   // XSL Transformation
   xslTpl = new ActiveXObject("Msxml2.XSLTemplate")
   xslTpl.stylesheet = domDocForXsl
   xslProc = xslTpl.createProcessor()
   xslProc.input = domDoc1

   // Pass parameters to the XSL stylesheet
```

```
   xslProc.addParameter("where",selectedOL)
   xslProc.addParameter("what",selectedEV)

   xslProc.transform
   htmlCode = xslProc.output
   // it is too tempting to do this a non-DOM way
   // innerHTML is not W3C, but widespread in browsers
   displayDiv.innerHTML = htmlCode
}
</script>
</body>
</html>
```

Dette åpner for en type systemløsninger som kan plassere mye av prosesseringsarbeidet på klienten. La oss anta at du skal undersøke dine konti og transaksjonshistorie i banken. Du kan laste ned en komplett XML-beskrivelse av ditt kundeforhold og søke i denne bare ved hjelp av nettleseren. Serveren og kommunikasjonskanalene avlastes og behovet for å ta kontakt med serveren oppstår bare dersom du skal gjøre endringer i din portofolio, f.eks. betale regninger.

XML-øyer

I eksemplene ovenfor har vi lastes ned de aktuelle filene, bygget DOM-trær og holdt disse i variable. Her skal vi se på en løsning der vi kan laste opp XML-filer og plassere dem i det som refereres til som XML-islands i dokumentet. Så kan vi plukke dem derfra og bygge trær etter behov. Vi bruker en enkel XML-fil med bøker som rådata: bokliste.xml

Følgende fil viser hvordan vi kan endre layout på et stykke data ved hjelp av to alternative XSLT-tansformasjoner:

```
<html>
   <head>
      <title>XML-Islands</title>
   </head>

<body onload="init()">
 <h1>Bokliste</h1>
 <div>Velg layout:
   <select name="ViewSelect" size="1" onchange="select()">
     <option value="Tabell">Tabell</option >
     <option value="Liste">Liste</option >
  </select >
 </div>

 <div id="displayDiv"></div>

 <script language="javascript">
   // on first time load
   function init()
   {
      raadata.async = false;
      // get the data
      raadata.load("bokliste.xml");
      // show default
      select();
   }

   // user respons on select
   function select()
   {
      //selected index and string
      var index = ViewSelect.selectedIndex;
      var selection = ViewSelect.options(index).text;

      var xslString = selection + "xsl";
      var stylesheet=document.getElementById(xslString)
      var showPiece = raadata.transformNode( stylesheet.documentElement );
      displayDiv.innerHTML = showPiece ;
   }
 </script >

 <XML id="raadata"></XML>
 <XML id="Tabellxsl" src="tabell.xsl"></XML>
 <XML id="Listexsl" src="liste.xsl"></XML>

 </body>
</html>
```

DEMO: http://www.ia.hiof.no/~borres/ml/ml/dommsxml/islands.html

De to transformasjonsfilene er svært enkle. Tabell.xsl:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" version="1.0"
      encoding="ISO-8859-1" indent="yes"/>

<xsl:template match="/">
<table border="1">
   <xsl:for-each select="/literature/book">
   <tr>
   <td><xsl:value-of select="title"/></td>
```

```
   <td><xsl:value-of select="category"/></td>
   <td><xsl:value-of select="author"/></td>
   </tr>
   </xsl:for-each>
</table>
</xsl:template>

</xsl:stylesheet>
```

og Liste.xsl:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" version="1.0"
      encoding="ISO-8859-1" indent="yes"/>

<xsl:template match="/">
<ul>
   <xsl:for-each select="/literature/book">
   <li><xsl:value-of select="title"/></li>
   </xsl:for-each>
</ul>
</xsl:template>

</xsl:stylesheet>
```

Referanser og eksempler:

MSXML hos Microsoft Developers Network:
msdn.microsoft.com/ ,søk etter MSXML.

Filer i eksemplene er sitert i koden

Inputkontroll

Hva:

Kontroll, validering av brukerinput

En fornuftig bruk av JavaScript er kontroll eller validering av det brukeren skriver inn i inputfelter (text, textarea) før det sendes til tjeneren. Typiske kontroller kan være:

1. Kontroll av om et felt er utfylt. Dette er som regel greitt å finne ut av.
2. Kontroll av om et felt har riktig syntaks. Bare siffere, bare bokstaver, riktig antall siffere eller bokstaver. Dette skaper heller ikke store problemer.
3. Riktig format i henhold til en mer presis syntaksbeskrivelse, f.eks. en dato. Dette kan være mer komplisert men bruk av regulære utttrykk kan hjelpe oss langt på vei. Javascript støtter dette.(mer om dette nedenfor)
4. At feltene er fornuftige i forhold til hverande. Dette kan være enkelt, men det kan også bli kanske komplisert og tungvindt å løse i JavaScript.
5. At dataene er lovlige gitt en bestemt situasjon. Dette avhenger helt av situasjonen. "lovlig" i denne sammenheng kan bety så mangt. F.eks. Kan det være greitt å teste om en dato har rett format, men det kan hende vi må konsultere en database eller en annen datakilde for å sjekke om den angitte datoen i f.eks. en bestillingssituasjon er ledig eller effektiv.

Det er trolig en god leveregel å finne en fornuftig balanse mellom det som med rimelighet kan skrives i JavaScript på klienten og det som med fordel kan løses på tjeneren. Vi kan kanskje si at det er rinmelig å foreta syntakskontroller på klienten og logiske kontroller på tjeneren.

Vi må også tenke gjennom hvordan og når vi skal gi tilbakemelding til brukeren om at noe er galt eller mangelfullt. Begivenhetsstyrt DOM-programmering er nøkkelen til dette.

Et enkelt eksempel

Vi begynner med et enkelt eksempel som ikke stiller store krav til selve kontrollen, valideringen. Hensikten er å se litt på organisering og rekkefølge. Vi tar for oss en situasjon der brukeren inviteres til å angi en 4-siffret tallkode og en tekst med 3 bokstaver.

Registrer dine data

Tall:(4 siffer)		
Tekst:(3 bokstaver)		
	Registrer	

Strategien er at vi skriver og kaller tre funksjoner slik:

- controlnumber(). Denne kalles når sifferfeltet mister fokus, begivenheten onblur. Denne setter opp en feilmelding og returnerer false dersom noe er galt i sifferfeltet, ellers true.
- controlalfa(). Denne kalles når bokstavfeltet mister fokus, begivenheten onblur. Denne setter opp en feilmelding og returnerer false dersom noe er galt i bokstavfeltet, ellers true.

- controlboth(). Denne kalles når vi trykker på Registrer-knappen, begivenheten onsubmit, og kaller de to andre funksjonene. Denne returnerer false dersom en av de to andre returnerer false, ellers true.

```
function controlnumber()
{
  inputelt=document.getElementById('number');
  errorelt=document.getElementById('numbererror');
  if ((inputelt.value==null) || (inputelt.value.length!=4))
  {
    errorelt.innerHTML='4 siffere';
    return false;
  }
  var legals = "0123456789";
  var T=inputelt.value;
  for (i = 0; i != T.length; i++)
    if (legals.indexOf(T.charAt(i)) == -1)
    {
      errorelt.innerHTML='bare siffere';
      return false;
    }
  errorelt.innerHTML='';
  return true;
}


function controlalfa()
{
  inputelt=document.getElementById('alfa');
  errorelt=document.getElementById('alfaerror');
  if ((inputelt.value==null) || (inputelt.value.length!=3))
  {
    errorelt.innerHTML='3 bokstaver';
    return false;
  }
  var legals = "abcdefghijklmopqrstuvwxyzæøå";
  var T=inputelt.value;
  T=T.toLowerCase()
  for (i = 0; i != T.length; i++)
    if (legals.indexOf(T.charAt(i)) == -1)
    {
      errorelt.innerHTML='bare bokstaver';
      return false;
    }
  errorelt.innerHTML='';
  return true;
}

function controlboth(){
  if (!controlalfa())
    return false;
  if (!controlnumber())
    return false;

  return true;
}
```

Den aktuelle HTML-koden er som nedenfor. Det kan virke litt overdrevet med table, men hensikten er å ordne de tre komponentene inledningstekst, selve inputfeltet og feilmeldingen.

```
<fieldset>
<legend>Registrer dine data</legend>
<form onsubmit="javascript:return controlboth();"
      action="http://www.ia.hiof.no/~borres/cgi-bin/forms/scriptvalidate1.py"
      method="post">
 <table cellpadding="5px" cellspacing="0px" border="0px">

 <tr>
  <td>Tall:(4 siffer)</td>
  <td><input id="number" name="siffere" onblur="controlnumber()"
      type="text" value="" maxlength="4" size="10"/></td>
  <td id="numbererror" style="color:red"></td>
 </tr>

 <tr>
  <td>Tekst:(3 bokstaver)</td>
  <td><input id="alfa" name="bokstaver" onblur="controlalfa()"
      type="text" value="" maxlength="3" size="10"/></td>
  <td id="alfaerror" style="color:red"></td>
 </tr>

 <tr>
  <td> </td>
  <td><input type="submit" value="Registrer"/></td>
  <td></td>
 </tr>

 </table>
</form>
</fieldset>
```

Merk onsubmit i form-elementet. Dersom controlboth() returnerer false dreper vi submitforsøket.

Et eksempel til

Vi lager et eksempel til der vi bruker regulæruttrykk for å validere input.

Registrer dato

Dato:(åååå:mm:dd)		
	Registrer	

Den aktuelle JavaScript-funksjonen ser slik ut:

```
function controldato()
{
  inputelt=document.getElementById('dato');
  errorelt=document.getElementById('datoerror');
  if ((inputelt.value==null) || (inputelt.value.length!=10))
  {
    errorelt.innerHTML='galt format';
    return false;
  }
  var T=inputelt.value;
  var regex=/(19|20)\d\d[:](0[1-9]|1[012])[:](0[1-9]|[12][0-9]|3[01])/;
  if (regex.test(T))
  {
    errorelt.innerHTML='';
    return true;
  }
  errorelt.innerHTML='galt format';
  return false;
}
```

Vi ser at vi har brukt regulære uttrykk, regex-objektet, til å teste formatet. Vi kunne gjort dette med ren string-programmering (split, lengde kontroll osv), men den løsningen som er valgt er vesentlig smidigere.

Den aktuelle HTML-koden er etter mønster av forrige eksempel slik:

```
  <fieldset>
  <legend>Registrer dato</legend>
  <form onsubmit="javascript:return controldato();"
        action="http://www.ia.hiof.no/~borres/cgi-bin/forms/scriptvalidate2.py"
        method="post">
   <table cellpadding="5px" cellspacing="0px" border="0px">

   <tr>
    <td>Dato:(ååää:mm:dd)</td>
    <td><input id="dato" name="dato" onblur="controldato()"
        type="text" value="" maxlength="10" size="11"/></td>
    <td id="datoerror" style="color:red"></td>
   </tr>

   <tr>
    <td></td>
    <td><input type="submit" value="Registrer"/></td>
    <td></td>
   </tr>

   </table>
  </form>
  </fieldset>
```

Nå har vi laget en ren syntakskontroll av datoformatet.

```
var regex=/(19|20)\d\d[:](0[1-9]|1[012])[:](0[1-9]|[12][0-9]|3[01])/;
```

Løsningen har noen svakheter. F.eks. vil 2006:02:31 passere syntakskontrollen selv om det er en dato som ikke finnes. På den annen side vil 1889:02:01 feile siden regulæruttrykket vårt bare aksepterer datoer på 1900 og 2000 tallet.

La oss anta at vi bare er intresserte i datoer på 1900 og 2000 tallet.

Det andre problemet, altså å fange opp problemer av typen 2001:02:31, kan vi løse på flere måter. Vi kan utvide regex-uttrykket og fange det opp der eller vi kan skrive string-kode. Vi prøver det siste, og lage følgende funksjoner:

```
function controldatox()
{
  inputelt=document.getElementById('datox');
  errorelt=document.getElementById('datoerrorx');
  if ((inputelt.value==null) || (inputelt.value.length!=10))
  {
    errorelt.innerHTML='galt format';
    return false;
  }
  var T=inputelt.value;
  var regex=/(19|20)\d\d[:]\d\d[:]\d\d/;
  if (regex.test(T))
  {
    if(checkdato(T))
    {
      errorelt.innerHTML='';
      return true;
    }
    else
    {
      errorelt.innerHTML='datoen eksisterer ikke';
      return false;
    }
  }
  errorelt.innerHTML='galt format eller galt årstall';
  return false;
}

function checkdato(s){
  var strArray=s.split(":");
  y=parseInt(strArray[0]);
  m=parseInt(strArray[1]);
  d=parseInt(strArray[2]);
  if ((m==1)||(m==3)||(m==5)||(m==7)||(m==8)||(m==10)||(m== 12))
  {
      return (d <= 31);
  }
  else if ((m==4)||(m==6)||(m==9)||(m==11))
  {
      return (d <= 30);
  }
  else if (m == 2)
  {
      if (d <= 28)
          return true;
      if (d > 29)
          return false;
      // d==29
      if ((y % 4 == 0) && (y % 100 != 0))
          return true;
      return false;
  }
  else
    return false

}
```

Test denne:

Registrer dato

Dato:(åååå:mm:dd)		
	Registrer	

Referanser og eksempler:

Pythonskriptene som gir respons

- scriptvalidate1
- scriptvalidate2

Førstehjelp regulæruttrykk: www.regular-expressions.info/

Popup

... det som alltid irriterer deg (når du ikke har laget det selv)

Hva:

popupvinduer og fragmentekspansjon

Det er mange grunner til å ikke bruke popupvinduer, men det er kanskje noen grunner til å bruke det. "Ubestilte" popupvinduer, vanligvis med reklame, er utrolig irriterende og det er gode grunner til at nettleserne har funksjonalitet for å kontrollere slike vinduer. På den annen side er det trolig noen grunner til at de kan brukes. I dette materialet brukes de til to ting: Eksponering av data som skal kunne kopieres direkte fra vinduet og eksponering av kodebiter eller forklaringer som er ment å fungere som en slags fotnoter for leseren. I alle tilfeller er det slik at popup-knapper er tydelig merket og ingen popupvinduer kommer ubestilt, altså uten at brukeren har valgt å få det opp ved å klikke.

Nedenfor skal vi se på noen teknikker for å lage popupvinduer ved hjelp av JavaScript, for ulike formål

Enkle popupvinduer

Det er en enkel sak å sette opp et vindu som skal fylles fra en kjent URL. Funksjonen som gjør dette er:

```
window.open(url, wname, wstyle)
```

der url er den siden vi vil åpne, wname er navnet på vinduet og wstyle er de egenskapene det nye vinduet skal ha. url er grei nok, men de to andre parametrene krever en kommentar.

Vinduets navn Dette navnet identifiserer vinduet. Det er altså ikke tittelen på vinduet og det har ikke noe med title-elementet å gjøre. Det er det samme som det vi kan bruke når vi oppgir target. I praksis er det slik at dersom vi gjør flere open-kall etter hverandre med samme navn, vil vinduet bli "gjenbrukt".

Vinduets egenskaper Vi kan sette en rekke egenskaper for det nye vinduet. Vi kan angi størrelse, plassering, hvorvidt det skal være scrollbart osv. Den komplette (?) lista er f.eks. slik (det skal ikke være linjeskift når du oppgir en slik string):

```
width=400,height=200,toolbar=yes,location=yes,directories=yes,
status=yes,menubar=yes,scrollbars=yes,copyhistory=yes,resizable=yes
```

Altså en list med kommaseparerte navn=verdi-par. Merk at denne stringen (eller deler av den) må angis uten blanke og uten linjeskift. Min erfaring er at det ikke er alle egenskapene som settes slik jeg ønsker, og noen (eldre?) nettlesere vil ha 1 for yes og 0 for no. Et rimelig vindu for de formålene jeg anga i starten av denne modulen kan være slik:

```
scrollbars=yes,resizable=yes,width=600,height=600,status=no
```

En funksjon som er rimelig robust og som dessuten sørger for at popup-vinduet vårt alltid havner fremst når vi setter det opp er slik:

```
function simplepopup(theURL,wname,wstyle)
{
    if(wstyle=='*')
        wstyle='scrollbars=yes,resizable=yes,width=600,height=600,status=no';
    try{
        newwindow=window.open(theURL, wname, wstyle);
        if (window.focus) {newwindow.focus()}
    }
    catch(E){
        // ??
        alert('If you have blocked pop-ups\n Try Ctrl-Alt when clicking');
    }
}
```

Merk at vi får et "standard" vindu dersom vi oppgir * som egenskaper. Merk dessuten at vi tar et initiativ som respons på en mulig popup-blokkering. Dersom vi forsøker å åpne en side som ikke eksisterer får vi vanlig 404-respons i popupvinduet.

Du kan teste her

simplepopup('http://www.ia.hiof.no/~borres/self/bs1.png','pop','*')

Generert popupvinduer

I stedet for å åpne en url, kan vi produsere innholdet i popupvinduet fra javascript-kode i det vinduet som setter opp popupvinduet. Dette er en interessant variant som gjør det lett å lage små popup-vinduer etter behov. Denne teknikken er basert på at vi bruker document.write- funksjonen i det nye vinduet. Vi kan f.eks. lage følgende:

```
function producepopup(wname,wstyle)
{
     if(wstyle=='*')
          wstyle='scrollbars=yes,resizable=yes,width=600,height=600,status=no';
  try{
    newwindow=window.open('', wname, wstyle);
      newwindow.document.write('<html><head><title>hei</title></head>');
      newwindow.document.write('<body><h1>Dette er en side</h1>');
      newwindow.document.write('<input type="button" value="ta meg vekk" '+
                               'onclick="self.close()"/>');
      newwindow.document.write('</body></html>');
      newwindow.document.close();
    if (window.focus)
    {
      newwindow.focus()
    }
  }
  catch(E){
          // ??
          alert('If you have blocked pop-ups\nTry Ctrl-Alt when clicking');
     }
}
```

Der vi altså begynner med å sette opp et vindu med blank url. Dersom dette går bra fortsetter vi å produsere innhold i det nye vinduet.

Du kan teste her

producepopup('pop','*')

Dette er interessant. Vi kan i en viss forstand kontrollere popupvinduet ut over de egenskapene vi kan sette i window.open(...). Spørsmålet som melder seg er om vi kan opprette en vindu med innhold, og så modifisere dette innholdet. Dette viser seg å være mer komplisert og vi kommer fort bort i situasjoner der nettleserne oppfører seg forskjellig.

AjaxPop

Det vi kan gjøre er å la popupvinduet fylle seg selv ved hjelp av AJAX. Det gjelder å formidle requestadressen til popupvinduet. Dette kan vi gjøre ved å sende adressen, og eventuelle parametere, som parameter til popupvinduets url. Dette fungerer fordi **document.location** inneholder både url'en til dokumentet og alle parameterne. Vi kan da lage oss et generisk ajax-popupvindu med følgende script:

```
// assuming my std library for ajax is present
// see: http://www.ia.hiof.no/~borres/common/jscripts/std.js
// use function: doGetRequest
// b.stenseth
function loadcontent()
{
  try{
    var T=''+document.location;
    // assume format:            url1?url2?par1=v1&par2=v2
    // request address will be:  url2?par1=v1&par2=v2
    var pos=T.indexOf('?');
    if(pos==-1){throw 'no address';}
    T=T.substring(pos+1);
    doGetRequest(T,'content','std');
  }
  catch(E){
    document.getElementById('content').innerHTML=E;
  }
}
```

Denne funksjonen kalles ved **<body onload="loadcontent()">**

DEMO: http://www.ia.hiof.no/~borres/ml/popup/poptest2.html

Ekspansjon av sidefragmenter

I dette materialet ser du at det ofte er brukt en mekanisme for å vise og skjule deler av en side. Motivasjonen for dette er lik motivasjonen for å bruke popup:

-
Jeg ønsker at leseren skal kunne velge hvor mye detaljer han/hun vil se.

Denne mekanismen er selvsagt i sin helhet basert på AJAX og er ganske enkel. Når det dynamiske innholdet er skjult ser det slik ut på vevsiden:

```
<p>
<span class="onoff" onclick="expand('request1.txt',this.parentNode);">+</span>
</p>
```

Når det dynamiske innholdet vises ser det slik ut på vevsiden:

```
<p>
<span class="onoff"
         onclick="unexpand('request1.txt.txt',this.parentNode)"> -
</span>
<p>
```

Den involverte javascriptkoden ser slik ut:

```
-
// dependent of style-class: onoff for visible effect
function expand(address,targetNode){
    var myRequest=establishRequest();
    if(myRequest)
    {
        myRequest.onreadystatechange = function( ) {
                    useExpansion(myRequest,address,targetNode);
        };
        myRequest.open("GET", address, true);
        myRequest.send(null);
    }
}

function useExpansion(theRequest,address,targetNode)
{
    // if the request is complete and successfull
    if (theRequest.readyState == 4) {
        if ((theRequest.status == 200) || (theRequest.status == 304))
        {
            var T=theRequest.responseText;
            var pos1=T.indexOf('<pre');
            var pos2=T.lastIndexOf('</pre');
            if ((pos1 !=-1) && (pos2 > pos1))
                T=T.substring(pos1,pos2);
            T='<span class="onoff" '+
              'onclick="unexpand(\''+address+'\',this.parentNode)">-</span>'
              +T;
            targetNode.innerHTML=T;
        }
        else
          {
            alert("Problem med tilgang til data:\n" + theRequest.statusText);
            targetNode.innerHTML=theRequest.statusText;
          }
    }
}

function unexpand(address,targetNode){
    T='<span class="onoff" '+
    'onclick="expand(\''+address+'\',this.parentNode);">+</span>';
    targetNode.innerHTML=T;
}
```

Disse funksjonene er en del av mitt JS-standarbibliotek og har derfor tilgang på funksjonen: **establishRequest()**. Merk også at stil-klassen **onoff** forutsettes.

Dersom du lurer, hele biblioteket er slik:

-

```
/*
Minialistic library used on my websites
- a simple popup,
     simplepopup(url,wname,wstyle)
- general AJAX with text request/post,
     doRequest(theURL,params,targetId,waiter)
- expand/unexpand text request/get
     expand(address,targetNode)
     unexpand(address,targetNode)
All functions may be used directly, but are also supported by WXT
B.Stenseth 2007.
*/

// ------------  pop ------------------
//popup
  function simplepopup(theURL,wname,wstyle)
  {
      if(wstyle=='*')
          wstyle='scrollbars=yes,resizable=yes,width=600,height=600,status=no';
      try{
          newwindow=window.open(theURL, wname, wstyle);
          if (window.focus) {newwindow.focus()}
      }
      catch(E){
          // ??
          alert('If you have blocked pop-ups\n Try Ctrl-Alt when clicking');
      }
  }
//eofpopup

//------------- std ajax ------------------
function establishRequest()
{
    var theRequest=null;
    if (window.XMLHttpRequest)
        theRequest = new XMLHttpRequest();
    else if (window.ActiveXObject)
    {
        try { theRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch(e) {
            try {theRequest= new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch(e) {theRequest=null;}
        }
    }
    else
        theRequest=null;
    return theRequest;
}

function postRequest(theRequest,params,targetId,theUrl)
{
    theRequest.onreadystatechange =
        function( ) {
                    processRequestChange(theRequest,targetId);
        };
    theRequest.open("POST", theUrl,true);
    theRequest.setRequestHeader("Content-type",
                               "application/x-www-form-urlencoded");
    theRequest.setRequestHeader("Content-length", params.length);
    theRequest.setRequestHeader("Connection", "close");
    theRequest.send(params);
}
```

```
function processRequestChange( aRequest,aTarget )
{
    if (aRequest.readyState == 4) {
        if ((aRequest.status == 200) || (aRequest.status == 304))
            document.getElementById(aTarget).innerHTML=
                aRequest.responseText;
        else
        {
            alert("Problems accessing data:\n" + aRequest.statusText);
            document.getElementById(aTarget).innerHTML=
                aRequest.statusText;
        }

    }
}

//------------------------ general ajax ------------------------
function doRequest(theURL,params,targetId,waiter)
{
    var myRequest=establishRequest();
    if (myRequest)
    {
        if(waiter)
        {
          if(waiter == 'std')
            waiter='http://www.ia.hiof.no/~borres/common/gfx/waiter.gif';
          try{
              document.getElementById(targetId).innerHTML=
                                    '<img src="'+waiter+'" alt=" "/>';
              }
          catch(E){}
        }
        postRequest(myRequest,params,targetId,theURL);
    }
    else
      {
        alert('Browser will not do a request');
    }
}

function doGetRequest(theURL,targetId,waiter)
{
    var myRequest=establishRequest();
    if (myRequest)
    {
        if(waiter)
        {
          if(waiter == 'std')
            waiter='http://www.ia.hiof.no/~borres/common/gfx/waiter.gif';
          try{
              document.getElementById(targetId).innerHTML=
                                    '<img src="'+waiter+'" alt=" "/>';
              }
          catch(E){}
        }
        myRequest.onreadystatechange =
          function( ) {
                      processRequestChange(myRequest,targetId);
          };
        myRequest.open("GET", theURL, true);
        myRequest.send(null);
    }
    else
```

```
      {
        alert('Browser will not do a request');
    }
}


// ------------------------ expansion ----------------------
//expand
// dependent of style-class: onoff for visible effect
function expand(address,targetNode){
    var myRequest=establishRequest();
    if(myRequest)
    {
        myRequest.onreadystatechange = function( ) {
                    useExpansion(myRequest,address,targetNode);
        };
        myRequest.open("GET", address, true);
        myRequest.send(null);
    }
}

function useExpansion(theRequest,address,targetNode)
{
    // if the request is complete and successfull
    if (theRequest.readyState == 4) {
        if ((theRequest.status == 200) || (theRequest.status == 304))
        {
            var T=theRequest.responseText;
            var pos1=T.indexOf('<pre');
            var pos2=T.lastIndexOf('</pre');
            if ((pos1 !=-1) && (pos2 > pos1))
                T=T.substring(pos1,pos2);
            T='<span class="onoff" '+
              'onclick="unexpand(\''+address+'\',this.parentNode)">-</span>'
              +T;
            targetNode.innerHTML=T;
        }
        else
          {
            alert("Problem med tilgang til data:\n" + theRequest.statusText);
            targetNode.innerHTML=theRequest.statusText;
          }
    }
}

function unexpand(address,targetNode){
    T='<span class="onoff" '+
    'onclick="expand(\''+address+'\',this.parentNode);">+</span>';
    targetNode.innerHTML=T;
}
//eofexpand
```

.

Referanser og eksempler:

reference

Ting og tang

Hva:

Noen utfordringer

Nedenfor finner du noen eksempler på Javascriptkode som forsøker å løse noen ganske typiske problemer.

Post-It

I dette eksempelet ønsker vi å lage sider med små gule lapper som "popper" opp eller ekspanderer ved mousedown og forsvinner ved mouseup.

Først et eksempel som ekspanderer et element og som fungerer for både Microsoft Explorer og Mozilla:

DEMO: http://www.ia.hiof.no/~borres/ml/dhtml/p-common.html

Så ser vi på en versjon som popper gule lapper. Eksempelet demonstrerer forskjellen på filosofien i Microsoft Explorer og Mozilla.

Explorer versjonenhttp://www.ia.hiof.no/~borres/ml/dhtml/p-utsagn-ie5.html

Mozilla versonenhttp://www.ia.hiof.no/~borres/ml/dhtml/p-utsagn-nn6.html

En fungerende men ganske sårbar løsning som skal fungere for mange nettlesere. Forsøk f.eks. å introdusere en doctype-linje i starten av siden.

DEMO: http://www.ia.hiof.no/~borres/ml/dhtml/p-all.html

Denne siste løsningen er inspirert av http://www.quirksmode.org/. Det store problemet med å få dette til å virke stabilt på tvers av nettlesere er å finne riktig posisjon for en begivenhet.

Puslespill

Dette eksempelet bruker absolutt posisjonering og litt DOM-skripting:

DEMO: http://www.ia.hiof.no/~borres/ml//dhtml/puzzle/page.html

Dragging

Det er etablert en metode, et sett med rutiner, som gjøre det mulig å "dragge" elementer på en vevside. Se på kildekoden.

DEMO: http://www.ia.hiof.no/~borres/ml/dhtml/drag/drag.html

Vindusillusjon

De samme drag-rutinene brukes sammen med litt kreativ CSS til å lage en illusjon av et vindu som kan åpnes, lukkes, dragges og fylles (vha AJAX).

DEMO: http://www.ia.hiof.no/~borres/ml/modalbox/movepage.html

Bildebølge

Bilder ekspanderer når vi trekker cursor over dem. Eksempel med valgresultat i Østfold-kommuner. Bruk av JSON og JavaScript

DEMO: http://www.ia.hiof.no/~borres/ml/imagewave/index.html

Referanser og eksempler:

Informasjon om scripting:

- Microsoft: msdn.microsoft.com/workshop/author/om/doc_object.asp
- Opera: www.opera.com/docs/specs/

Javascript tutorial: www.webdevelopersnotes.com/tutorials/javascript/

Et bibliotek for popupvinduer: www.bosrup.com/web/overlib/

Ajax

Hva:

Dynamisk oppdatering av vevsider ved hjelp av Javascript



Mye av arbeidet med å lage nettbaserte løsninger går ut på å fordele arbeidet mellom tjener og klient. I dette materialet konsentrer vi oss i all hovedsak om forholdet mellom en nettleser og et CGI-script (skrevet i Python).

Vi har i modulen MSXML sett på hvordan vi kan laste ned mer data enn det som vises fram ved å legge dem enten som "XML-islands" eller vi kan laset opp masse materiale og legge dem slik at de er usynlige. Brukerens interaksjon med nettleseren kan ordne og vise slike skjulte data. Dette ordner vi med javaskripting.

I disse modulene skal vi gå et skritt videre i å flytte logikk til klientene, nettleseren. Vi skal lage HTML-sider som oppdateres dynamisk, uten at hele siden oppfriskes.

Det er to objekter som Javascript kjenner i en html-side som er spesielt aktuelle i sammenhengen:

- document. DOM-objektet er brukt i mange sammenhenger i dette materialet, se f.eks DOM. DOM er ikke en nødvendig del av den mekansimen vi skal se på her, men det er svært nyttig når vi skal foreta ikke-triviell behandling av mottatte data.
- XMLHttpRequest. Det er dette objektet som setter oss i stand til å hente data fra en annen URL. De dataene vi henter kan vi plassere ut på vår HTML-side eller vi kan bruke dem til å kontrollere brukerens inn data eller hva vi måtte finne på.

Basisfunksjoner

```
var myRequest;

function startXMLHttpRequest(url)
{
    if (window.XMLHttpRequest)
        myRequest = new XMLHttpRequest();//IE7
    else if (window.ActiveXObject)
    {
        try { myRequest = new ActiveXObject("Msxml2.XMLHTTP");//IE6
        }
        catch(e) {
            try {myRequest= new ActiveXObject("Microsoft.XMLHTTP");//IE5.5
            }
            catch(e) {myRequest=null;}
        }
    }
    else
        myRequest=null;

    if (myRequest) {
        myRequest.onreadystatechange = processRequest;
        myRequest.open("GET", url, true);
        myRequest.send(null);
    }
    else{
        alert("Nettleseren henger ikke med");
    }
}

function processRequest()
{
    // if the request is complete and successfull
    if (myRequest.readyState == 4) {
        if ((myRequest.status == 200) || (myRequest.status == 304))
            alert(myRequest.responseText);
        else
            alert("Problem med tilgang til data:\n" + myRequest.statusText);
    }
}
```

Koden over definerer et "globalt" XMLHttpRequest-objekt og to metoder. Den første metoden, **startXMLHttpRequest**, adresserer en fil via en URL. Som det framgår er det her, som så mange andre steder slik at vi må spesiell behandling av MSIE, siden det XMLHttpRequest-objektet vi oppretter skal være et ActiveXObject. Når vi har fått opprettet objektet ser vi at vi angir hvilken funksjon som skal være eventhandler: "myRequest.onreadystatechange = processRequestChange;". Deretter åpner vi en GET-forespørsel og vi sender den uten tilleggsdata. De aktuelle metodene og egenskapene ved et XMLHttpRequest-objekt er listet nedenfor.

Den andre funksjonen, **processRequestChange**, er programmert slik at den bare bekymrer seg for tilstanden som tilsier at forespørselene er ferdig behandlet. Dersom status på dette tidspunktet er 200 så har alt gått bra og vi kan plukke opp resultatet som text og viser det fram i en alert-boks.

Du kan teste ved å klikke her.

Fragmentet som sender forespørselen:

```
<p style="cursor:pointer"
    onclick="javascript:startXMLHttpRequest('frej1.txt');return false;">
    Du kan teste ved å klikke her.
</p>
```

En viktig begrensning

JavaScript er underlagt begrensninger i hva vi kan få tak i via koden. Begrensningene administreres av nettleserne. Begrunnelsen er sikkerhet. Når vi lager AJAX-løsninger må de ressursene, filene, vi bruker ligge i samme domene som den HTML-siden som bruker dem. Denne sikkerhetsmekanismen er ganske involvert og det vil føre for langt å forfølge dette her. Kortformen for våre praktiske løsninger er at vi legger alt vi vil bruke på samme server og samme bruker.

I praksis er ikke dette noen alvorlig begrensning. Det normale er vel at vi bygger en løsning med tilgjengelige ressurser innenfor et domene. Det er ikke noe problem å bruke fremmede ressurser, men da gjør vi det via kode på tjeneren, f.eks. et pythonskript, som legges i det lovlige domenet. Webservices kan være eksempler på slike "fremmed" ressurser. Vi kan selvsagt også hente data fra databaser på andre tjenere, typisk en 3-lags løsning.

Modulen Andre domener drøfter nærmere noen strategier for å hente data fra andre domener.

Noen løsninger

Tekst til DOM-node

Vi har altså, hvis øvelsen ovenfor gikk bra, greidd å få tak i en tekst fra en URL. Når vi nå har tak i den burde vi kunne legge den ut på siden i stedet for å vise den fram i en alert boks. Vi bruker altså DOM-programmering for å plassere den teksten vi henter.

DEMO: http://www.ia.hiof.no/~borres/ml/ajax/demo2/page2.html

Velge blandt flere

En vanlig situasjon er selvsagt at vi ønsker å tilfredsstille et valg som brukeren treffer. brukeren vil velge blandte flere alternativer. Eksempelet viser hvordan vi kan kople inn en select-boks.

DEMO: http://www.ia.hiof.no/~borres/ml/ajax/demo4/page4.html

Header informasjon

Vi kan sende en forespørsel som bare returnerer header-informasjon fra en fil. På denne måten kan vi finne ut når en ressurs sist ble oppdatert, mime-typen, servertypen etc. Vi bruker **HEAD** i stedet for GET eller POST. Vi henter ut **getAllResponseHeaders()** i stedet for responseText.

DEMO: http://www.ia.hiof.no/~borres/ml/ajax/demo3/page3.html

Tolke resultatet som XML

Vi kan tolke det svaret vi får på en forespørsel som en DOM-struktur. Det vil si at vi kan hente en XML-fil, analysere denne som et DOM-tre og bruke de delene vi vil til å møblere HTML-siden, som vi betrakter som et annet DOM-tre. Vi plukker da opp **responseXML** i stedet for responseText.

DEMO: http://www.ia.hiof.no/~borres/ml/ajax/demo5/page5.html

Bruk av pythonskript på tjeneren

I stedet for å laste ned en fil direkte, kan vi la kode på tjeneren returnere de dataene vi er ute etter, i den form vi vil ha dem. Vi må da tenke gjennom arbeidsdelingen mellom klient og tjener. Eksempelet demonstrerer hvordan vi kan hente 1 av Shakespears sonetter via et Pythonskript på tjeneren.

DEMO: http://www.ia.hiof.no/~borres/ml/ajax/demo6/page6.html

Skriptet som betjener siden: demo6/shakereturn_py.html

Bruk av pythonskript på tjeneren, POST

Samme som ovenfor, men vi bruker **POST** istedet for GET.

DEMO: http://www.ia.hiof.no/~borres/ml/ajax/demo7/page7.html

Et enkelt "vevsted"

Et enkelt vevsted med 3 sider, eller egentlig en side der hovedinnholdet byttes ut.

DEMO: http://www.ia.hiof.no/~borres/ml/ajax/demosite/index.html

GET eller POST

Vi kan hente data med GET eller med POST. POST har den egenskapen at den kan transportere større datamengder. Anta at vi har etablert et XMLHttpRequest-objekt og at vi har gjort klar parameterlista, params.

POST:

```
...
myRequest.open("POST", url,true);
myRequest.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
myRequest.setRequestHeader("Content-length", params.length);
myRequest.setRequestHeader("Connection", "close");
myRequest.send(params);
```

GET:

```
...
myRequest.open("GET", url+'?'+params, true);
myRequest.send(null);
```

Det samme Pythonskriptet på betjener begge metodene.

Metoder og egenskaper

Et XMLHttpRequest-objekt har følgende metoder og egenskaper:

Metode	
abort()	Kansellerer pågående forespørsel
getAllResponseHeaders()	Returnerer et komplett sett av alle "http headers" som text
getResponseHeader("headername")	Returnerer verdien av en "http header"
open("method","URL",async,"user","password")	Spesifiserer metoden URL-adressen og tre valgfrie andre parametre i en forespørsel **method** kan ha verdiene "GET", "POST", eller "PUT" ("GET" når vi vil hente data og "POST" når vi sender data. Når vi sender mer enn 512 byte må "POST" brukes). **URL** parameteren kan være absolutt eller relativ. **async** parameteren spesifiserer om forespørselen skal behandles asynkront eller ikke. Dersom den er true betyr det at skriptet kan fortsette uten å vente på svar.
send(content)	Sender forespørsel. content legger data til en forespørsel som er åpnet med POST
setRequestHeader("label","value")	Legger til et par(navn og verdi) til en "http header" som skal sendest
Egenskaper	
onreadystatechange	Inneholder en funksjon, en event handler, som skal benyttes når tilstanden til XMLHttpRequest-objektet endres.
readyState	Returnerer tilstenden til objektet: • 0 = ikke initialisert • 1 = lastes • 2 = lastet • 3 = interktivt • 4 = komplett
responseText	Returnerer responsen på en forespørsel som en tekst
responseXML	Returnerer responsen på en forespørsel som XML. Responsen er i form av et XML dokument som kan parses ved hjelp av DOM-metoder.
status	Returnerer status som et tall (f.eks.: 404 for "Not Found" eller 200 for "OK")
statusText	Returnerer status som en tekst(f.eks.: "Not Found" eller "OK")

Referanser og eksempler:

Kort om HTTP, f.eks.: www.answers.com/topic/http

AJAX-ressurser: www.ajaxprojects.com/

Ajax (Asynchronous JavaScript and XML): en.wikipedia.org/wiki/AJAX

Prototype, et rimelig JavaScript bibliotek med AJAX-funksjonalitet: prototypejs.org/

Ajax for viderekommende:
Johnson, White and Charland: Enterprice AJAX,
Prentice Hall,2008
ISBN-13:978-0-13-224206-6

Ruby on Rails: www.rubyonrails.com/

Kodestrategier

Hva:

AJAX-koding på klienten

Når vi skal lage ikke-trivielle AJAX-løsninger må vi planlegge programmeringsarbeidet. Det kan fort bli mye Javascriptkode og det kan fort bli mye gjentagelser.

Det er i hvertfall to valgsituasjoner vi må ta stilling til:

- Vi må ta stilling til om vi skal kode alt fra bunnen av eller om vi skal bruke et bibliotek. Dersom vi har et overkommelig problem og vi bestemmer oss for å kode fra bunnen av, så bør vi likevel planlegge koden.
- Vi må ta stilling til hva slags dataformat vi skal bruke på de dataene vi henter og sender med i en request. Skal vi bruke tekst (myRequest.responseText) eller DOM (myRequest.responseXML) på klientsiden, og hvordan skal vi pakke data i text-formatet. Mer om dette i modulen Dataformater

La oss ta utgangspunkt i det enkle tilfellet at vi bruker POST og at vi tolker svaret på en henvendelse som text, dvs. text som er formatert som XHTML fra tjeneren. Videre at denne teksten i sin helhet skal plasseres som innhold i en bestemt tag, merket med id, på siden. Dette er trolig den situasjonen som er mest vanlig i relativt enkle anvendelser. Koden for å handtere denne situasjonen er i sin enkle form slik:

```
var myRequest;

function startXMLHttpRequest(theUrl)
{
    if (window.XMLHttpRequest)
        myRequest = new XMLHttpRequest();
    else if (window.ActiveXObject)
    {
        try { myRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch(e) {
            try {myRequest= new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch(e) {myRequest=null;}
        }
    }
    else
        myRequest=null;

    if (myRequest) {
      myRequest.onreadystatechange = processRequestChange;
      params="<something we read form the page>";
      theRequest.open('POST',theUrl,true);
      theRequest.setRequestHeader("Content-type",
                                  "application/x-www-form-urlencoded");
      theRequest.setRequestHeader("Content-length", params.length);
      theRequest.setRequestHeader("Connection", "close");
      theRequest.send(params);
    }
    else{
        alert("Nettleseren henger ikke med");
    }
}

function processRequestChange()
{
    // if the request is complete and successfull
    if (myRequest.readyState == 4) {
        if ((myRequest.status == 200) || (myRequest.status == 304))
            document.getElementById('resultat').innerHTML=
                myRequest.responseText;
        else
            alert("Problem med tilgang til data:\n" + myRequest.statusText);
    }
}
```

Koderasjonalisering

Vi merker oss at vi trenger en global variabel, myRequest, for å huske hvilken henvendelse som er "utestående". Vi ser dessuten at dersom vi har en løsning med mange henvendelser, så må vi for sikkerhets skyld huske mange henvendelser eller vi må sørge for at vi bare har en utestående om gangen.

Vi ser dessuten at dersom vi har mange mulige henvendelser så blir mye av koden ren gjentagelse: Vi skal opprette en forespørsel, vi skal sende den og vi skal ta i mot og plassere svaret. Det eneste som skiller en situasjon fra en annen er at vi har forskjellige adresser å plassere resultatet i, at vi har forskjellige parametere og kansje at vi har forskjellige URL'er.

Vi kan løse opp i denne situasjonen med å bruke de mulighetene Javascript gir oss for å kople en anonym funksjon til et request-objekt som onreadystatechange. Koden nedenfor gir oss et rammeverk, eller om du vil et lite bibliotek, som gjør at vi kan rasjonalisere koden kraftig.

```
function establishRequest()
{
    var theRequest=null;
    if (window.XMLHttpRequest)
        theRequest = new XMLHttpRequest();
    else if (window.ActiveXObject)
    {
        try { theRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch(e) {
            try {theRequest= new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch(e) {theRequest=null;}
        }
    }
    else
        theRequest=null;
    return theRequest;
}

function postRequest(theRequest,params,targetId,theUrl)
{
    theRequest.onreadystatechange =
        function( ) {
                    processRequestChange(theRequest,targetId);
        };
    theRequest.open("POST", theUrl,true);
    theRequest.setRequestHeader("Content-type",
                                "application/x-www-form-urlencoded");
    theRequest.setRequestHeader("Content-length", params.length);
    theRequest.setRequestHeader("Connection", "close");
    theRequest.send(params);
}

function processRequestChange( aRequest,aTarget )
{
    if (aRequest.readyState == 4) {
        if ((aRequest.status == 200) || (aRequest.status == 304))
            document.getElementById(aTarget).innerHTML=
                aRequest.responseText;
        else
            alert("Problem med tilgang til data:\n" + aRequest.statusText);

    }
}
```

Ved hjelp av disse funksjonene kan et par AJAX-henvendelser fra siden vår kode se slik ut:

```
function startMenuRequest()
{
    var myRequest=establishRequest();
    if (myRequest) {
        postRequest(myRequest,
                    'theJob=country-list','leftmenu','../myscript.py');
    }
    else{
        alert("Nettleseren henger ikke med");
    }
}

function startResultHeaderRequest(ligaId)
{
    var myRequest=establishRequest();
    if(myRequest){
        postRequest(myRequest,
                'theJob=result-header&ligaId='+ligaId,
                'resultheader',
                '../myscript.py');
    }
    else{
        alert("Nettleseren henger ikke med");
    }
}
```

Dersom vi har behov for å sjekke data i en form eller hva det måtte være, kan vi ggøre dette i vår funkjson før vi setter opp en forespørsel. F.eks. slik:

```
function startMenuRequest(form)
{
    // pick up and control data from the form

    if (<something wrong or missing>)
    {
      <perform the proper alert or errormarking>
      return;
    }
    // proceed with the requst
    var myRequest=establishRequest();
    if (myRequest) {
        postRequest(myRequest,'theJob=country-list','leftmenu','../myscript.py');
    }
    else{
        alert("Nettleseren henger ikke med");
    }
}
```

Biblioteker

Selv om det vi har foreslått ovenfor kan være et nyttig knippe funkjsoner for å effektivisere koding av enkle eksempler er det ikke et bibliotek som er generelt nok til å handtere og feilsikre mer komplekse situasjoner. Ikke uventet har den økte bruken av AJAX ført til at det dukker opp en rekke biblioteker av uilkt omfang som skal hjelpe oss å skrive rask og sikker AJAX-kode på klientsiden.

Det samlede, hjemmelagde, biblioteket jeg bruker på disse sidene og andre lignende sider jeg vedlikeholder ser slik ut:

-

```
/*
Minialistic library used on my websites
- a simple popup,
     simplepopup(url,wname,wstyle)
- general AJAX with text request/post,
     doRequest(theURL,params,targetId,waiter)
- expand/unexpand text request/get
     expand(address,targetNode)
     unexpand(address,targetNode)
All functions may be used directly, but are also supported by WXT
B.Stenseth 2007.
*/

// ------------  pop ------------------
//popup
  function simplepopup(theURL,wname,wstyle)
  {
      if(wstyle=='*')
          wstyle='scrollbars=yes,resizable=yes,width=600,height=600,status=no';
      try{
          newwindow=window.open(theURL, wname, wstyle);
          if (window.focus) {newwindow.focus()}
      }
      catch(E){
          // ??
          alert('If you have blocked pop-ups\n Try Ctrl-Alt when clicking');
      }
  }
//eofpopup

//------------- std ajax ------------------
function establishRequest()
{
    var theRequest=null;
    if (window.XMLHttpRequest)
        theRequest = new XMLHttpRequest();
    else if (window.ActiveXObject)
    {
        try { theRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch(e) {
            try {theRequest= new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch(e) {theRequest=null;}
        }
    }
    else
        theRequest=null;
    return theRequest;
}

function postRequest(theRequest,params,targetId,theUrl)
{
    theRequest.onreadystatechange =
        function( ) {
                    processRequestChange(theRequest,targetId);
        };
    theRequest.open("POST", theUrl,true);
    theRequest.setRequestHeader("Content-type",
                               "application/x-www-form-urlencoded");
    theRequest.setRequestHeader("Content-length", params.length);
    theRequest.setRequestHeader("Connection", "close");
    theRequest.send(params);
}
```

```
function processRequestChange( aRequest,aTarget )
{
    if (aRequest.readyState == 4) {
        if ((aRequest.status == 200) || (aRequest.status == 304))
            document.getElementById(aTarget).innerHTML=
                aRequest.responseText;
        else
        {
            alert("Problems accessing data:\n" + aRequest.statusText);
            document.getElementById(aTarget).innerHTML=
                aRequest.statusText;
        }

    }
}

//------------------------ general ajax ------------------------
function doRequest(theURL,params,targetId,waiter)
{
    var myRequest=establishRequest();
    if (myRequest)
    {
        if(waiter)
        {
          if(waiter == 'std')
            waiter='http://www.ia.hiof.no/~borres/common/gfx/waiter.gif';
          try{
              document.getElementById(targetId).innerHTML=
                                    '<img src="'+waiter+'" alt=" "/>';
              }
          catch(E){}
        }
        postRequest(myRequest,params,targetId,theURL);
    }
    else
      {
        alert('Browser will not do a request');
    }
}

function doGetRequest(theURL,targetId,waiter)
{
    var myRequest=establishRequest();
    if (myRequest)
    {
        if(waiter)
        {
          if(waiter == 'std')
            waiter='http://www.ia.hiof.no/~borres/common/gfx/waiter.gif';
          try{
              document.getElementById(targetId).innerHTML=
                                    '<img src="'+waiter+'" alt=" "/>';
              }
          catch(E){}
        }
        myRequest.onreadystatechange =
          function( ) {
                      processRequestChange(myRequest,targetId);
          };
        myRequest.open("GET", theURL, true);
        myRequest.send(null);
    }
    else
```

```
      {
        alert('Browser will not do a request');
    }
}


// ------------------------ expansion ----------------------
//expand
// dependent of style-class: onoff for visible effect
function expand(address,targetNode){
    var myRequest=establishRequest();
    if(myRequest)
    {
        myRequest.onreadystatechange = function( ) {
                    useExpansion(myRequest,address,targetNode);
        };
        myRequest.open("GET", address, true);
        myRequest.send(null);
    }
}

function useExpansion(theRequest,address,targetNode)
{
    // if the request is complete and successfull
    if (theRequest.readyState == 4) {
        if ((theRequest.status == 200) || (theRequest.status == 304))
        {
            var T=theRequest.responseText;
            var pos1=T.indexOf('<pre');
            var pos2=T.lastIndexOf('</pre');
            if ((pos1 !=-1) && (pos2 > pos1))
                T=T.substring(pos1,pos2);
            T='<span class="onoff" '+
              'onclick="unexpand(\''+address+'\',this.parentNode)">-</span>'
              +T;
            targetNode.innerHTML=T;
        }
        else
          {
            alert("Problem med tilgang til data:\n" + theRequest.statusText);
            targetNode.innerHTML=theRequest.statusText;
          }
    }
}

function unexpand(address,targetNode){
    T='<span class="onoff" '+
    'onclick="expand(\''+address+'\',this.parentNode);">+</span>';
    targetNode.innerHTML=T;
}
//eofexpand
```

De flestes har vel en terskel for å ta i bruk store biblioteker. Komleksiteten i problemet du ønsker å løse må stå i forhold til kompleksiteten i verktøyet, du må ha en gevinst for sette deg inn i verktøyet.

Dette materialet inneholder foreløpig ikke materiale om AJAX-biblioteker, men det er all grunn til å følge med på hva som skjer. Du finner noen referanser nedenfor til biblioteker som kanskje er verdt en studie.

Referanser og eksempler:

- AJAX-ressurser: www.ajaxprojects.com/
- Prototype, et rimelig JavaScript bibliotek med AJAX-funksjonalitet: prototypejs.org/
- Mochikit, et JavaScript bibliotek med AJAX-funksjonalitet: mochikit.com/

Progress

... mens vi venter

Hva:

indikasjon av at vi venter på en AJAX-forespørsel

Når vi laster opp en helt ny side vil nettleseren med forskjellige mekanismer indikere en progresjon og i hvert fall i en viss utstrekning holde oss som brukere informert om hvordan vår forespørsel om en ny vevside går.

Når vi bruker AJAX får vi ingen slike indikasjoner. Vevsiden endrer seg ikke før forespørselen er fullført og nettleseren gir ingen indikasjoner på at noe er på gang. Det betyr at dersom vi gjør forespørsler som antas å ta litt tid, databaseoppslag, beregninger bruk av webservices eller hva det måtte være, så må vi selv ta ansvaret for å gi en form for "vente"-informasjon eller "ta det med ro jeg arbeider med saken"-indikasjon.

Vi kan løse dette på mange måter ved hjelp av Javacsript. Nedenfor er en enkel løsning som bruker animert GIF som indikator:

start
stop

I dette tilfellet starter og stopper vi progress indikatoren eksplisitt ved å klikke for start og stopp. Det er enkelt å knytte disse triggerne (startspin og stopspin) til de funksjonene i en AJAX-løsning som henholdsvis sender en forespørsel og mottar resultatet.

Det involverte Javascriptet ser slik ut:

```
function stopspin(){
  eltShow=document.getElementById('showspin');
  eltWhat=document.getElementById('spin0FF');
  eltShow.innerHTML=eltWhat.innerHTML;
}

function startspin(){
  eltShow=document.getElementById('showspin');
  eltWhat=document.getElementById('spinON');
  eltShow.innerHTML=eltWhat.innerHTML;
}
```

Den delen av HTML-koden på denne siden som er invovert i denne mekanismen ser slik ut:

```
<!-- starting the spintest -->
<div style="cursor:pointer" onclick="startspin()">start</div>

<!-- stop the spin test -->
<div style="cursor:pointer" onclick="stopspin()">stop</div>

<!-- spin position -->
<p id="showspin">
<image src="wait28_OFF.gif" alt ="0"/>
</p>

<!-- storing spin images the first is a placeholder with same size as the spinner -->
<div id="spin0FF" style="display:none"><image src="wait28_OFF.gif" alt ="0"/></div>
<div id="spinON" style="display:none"><image src="wait28.gif" alt ="1"/></div>
```

Et AJAX-eksempel

Du kan prøve et AJAX-eksempel som bruker mitt standardbibliotek med noen spinnervarianter nedenfor. Biblioteket ser slik ut:

-

```
/*
Minialistic library used on my websites
- a simple popup,
     simplepopup(url,wname,wstyle)
- general AJAX with text request/post,
     doRequest(theURL,params,targetId,waiter)
- expand/unexpand text request/get
     expand(address,targetNode)
     unexpand(address,targetNode)
All functions may be used directly, but are also supported by WXT
B.Stenseth 2007.
*/

// ------------  pop ------------------
//popup
  function simplepopup(theURL,wname,wstyle)
  {
      if(wstyle=='*')
          wstyle='scrollbars=yes,resizable=yes,width=600,height=600,status=no';
      try{
          newwindow=window.open(theURL, wname, wstyle);
          if (window.focus) {newwindow.focus()}
      }
      catch(E){
          // ??
          alert('If you have blocked pop-ups\n Try Ctrl-Alt when clicking');
      }
  }
//eofpopup

//------------- std ajax ------------------
function establishRequest()
{
    var theRequest=null;
    if (window.XMLHttpRequest)
        theRequest = new XMLHttpRequest();
    else if (window.ActiveXObject)
    {
        try { theRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch(e) {
            try {theRequest= new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch(e) {theRequest=null;}
        }
    }
    else
        theRequest=null;
    return theRequest;
}

function postRequest(theRequest,params,targetId,theUrl)
{
    theRequest.onreadystatechange =
        function( ) {
                    processRequestChange(theRequest,targetId);
        };
    theRequest.open("POST", theUrl,true);
    theRequest.setRequestHeader("Content-type",
                               "application/x-www-form-urlencoded");
    theRequest.setRequestHeader("Content-length", params.length);
    theRequest.setRequestHeader("Connection", "close");
    theRequest.send(params);
}
```

```
function processRequestChange( aRequest,aTarget )
{
    if (aRequest.readyState == 4) {
        if ((aRequest.status == 200) || (aRequest.status == 304))
            document.getElementById(aTarget).innerHTML=
                aRequest.responseText;
        else
        {
            alert("Problems accessing data:\n" + aRequest.statusText);
            document.getElementById(aTarget).innerHTML=
                aRequest.statusText;
        }

    }
}

//------------------------ general ajax ------------------------
function doRequest(theURL,params,targetId,waiter)
{
    var myRequest=establishRequest();
    if (myRequest)
    {
        if(waiter)
        {
          if(waiter == 'std')
            waiter='http://www.ia.hiof.no/~borres/common/gfx/waiter.gif';
          try{
              document.getElementById(targetId).innerHTML=
                                    '<img src="'+waiter+'" alt=" "/>';
              }
          catch(E){}
        }
        postRequest(myRequest,params,targetId,theURL);
    }
    else
      {
        alert('Browser will not do a request');
    }
}

function doGetRequest(theURL,targetId,waiter)
{
    var myRequest=establishRequest();
    if (myRequest)
    {
        if(waiter)
        {
          if(waiter == 'std')
            waiter='http://www.ia.hiof.no/~borres/common/gfx/waiter.gif';
          try{
              document.getElementById(targetId).innerHTML=
                                    '<img src="'+waiter+'" alt=" "/>';
              }
          catch(E){}
        }
        myRequest.onreadystatechange =
          function( ) {
                      processRequestChange(myRequest,targetId);
          };
        myRequest.open("GET", theURL, true);
        myRequest.send(null);
    }
    else
```

```
      {
        alert('Browser will not do a request');
    }
}


// ------------------------ expansion ----------------------
//expand
// dependent of style-class: onoff for visible effect
function expand(address,targetNode){
    var myRequest=establishRequest();
    if(myRequest)
    {
        myRequest.onreadystatechange = function( ) {
                    useExpansion(myRequest,address,targetNode);
        };
        myRequest.open("GET", address, true);
        myRequest.send(null);
    }
}

function useExpansion(theRequest,address,targetNode)
{
    // if the request is complete and successfull
    if (theRequest.readyState == 4) {
        if ((theRequest.status == 200) || (theRequest.status == 304))
        {
            var T=theRequest.responseText;
            var pos1=T.indexOf('<pre');
            var pos2=T.lastIndexOf('</pre');
            if ((pos1 !=-1) && (pos2 > pos1))
                T=T.substring(pos1,pos2);
            T='<span class="onoff" '+
              'onclick="unexpand(\''+address+'\',this.parentNode)">-</span>'
              +T;
            targetNode.innerHTML=T;
        }
        else
          {
            alert("Problem med tilgang til data:\n" + theRequest.statusText);
            targetNode.innerHTML=theRequest.statusText;
          }
    }
}

function unexpand(address,targetNode){
    T='<span class="onoff" '+
    'onclick="expand(\''+address+'\',this.parentNode);">+</span>';
    targetNode.innerHTML=T;
}
//eofexpand
```

Standard spinner implementert i biblioteket std.js

doRequest('http://www.ia.hiof.no/~borres/cgi-bin/ajaxtest/waiter.py','','result1','std')

Klikk her for å hente litt tekst

?

Hjemmlagd spinner: inspirert av MS-Windows load

doRequest('http://www.ia.hiof.no/~borres/cgi-bin/ajaxtest/waiter.py','','result2','spinbar.gif')

Klikk her for å hente litt tekst

?

Ingen spinner

doRequest('http://www.ia.hiof.no/~borres/cgi-bin/ajaxtest/waiter.py','','result3')

Klikk her for å hente litt tekst

?

Referanser og eksempler:

All koden er sitert på siden.

- Noen spinnere, animert GIF: www.andrewdavidson.com/articles/spinning-wait-icons/
- Lag dine egne fra maler: www.ajaxload.info/
- GIF Animator, for detaljert håndarbeid: www.jhepple.com/gif_animator.htm

JSON

JavaScript Object Notation

Hva:

JSON , Javascriptobjekter

JSON er, slik vi skal betrakte det, en måte å pakke data på slik at de blir enkle å bearbeide på klientsiden. Vi har hele tiden AJAX som referanseramme. Det betyr at vi skal bruke JSON som format når vi returnerer svar på en forespørsel, en XMLHttpRequest.

Grunnlaget for JSON er at vi kan tolke tekstformat som Javascriptobjekter i et javaskript. Vi kan f.eks. gjøre slik:

```
var T='{"navn":"Ole","adresse":"Halden"}'
var person=eval("("+T+")");
alert(person.navn);
```

Nøkkelen er eval-funksjonen som tolker teksten som evalueres som Javascriptkode. Siden Javascrupt er et interpretert språk er dette en vanlig og lett implementerbar mekanisme.

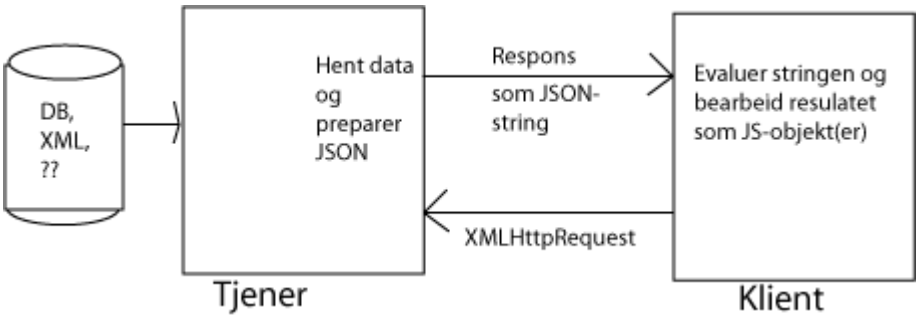
Vi kan også gjøre slik:

```
var T='{"personer":[{"navn":"Ole","adresse":"Halden"},{"navn":"Jens","adresse":"Moss"}]}'
var p=eval("("+T+")");
var person= p.personer;
alert(person[1].navn);
```

Full forklaring av syntaksen finnes på [1] . Vi kan nøye oss med å konstatere at vi kan pakke attributetr i objekter ved å angi "attributtnavn":"attributtverdi". Vi kan lage en kommaseparert liste av slike par. Videre ser vi at vi kan lage arrays med [].

Det interessante med denne angrepsvinkelen er altså at vi får svært enkel koding på klientsiden. Vi får tilgang til objekter og kan plukke attributter etter ønske.

Hvis vi ser dette i en AJAX situasjon, har vi følgende:



Hvorvidt bruk av JSON er en effektiv eler enkel metode avhenger av hvordan dataene våre er representerte på tjeneren, og det avhenger av hva vi må gjøre med dem på klienten. Det er klart at dersom vi skal overføre en hel fil som er ferdiglagd som et HTML-fragment og vi skal plassere dette fragmentet samlet på klienten er det mest effektive å bruke ren tekst slik:

```
document.getElementById("target").innerHTML= myRequest.responseText
```

Dersom teksten som kommer tilbake består av mange "deler" som skal plasseres på forskjellige steder på siden stiller saken seg litt annerledes. Både kommaseparert tekst, XML og JSON kan være alternativer.

På tjenersiden vil mye avhenge av dataformatet og programmeringsverktøyet vi har til rådighet. Som regel kan vi vel regne med at programmeringsverktøyet på tjeneren er bedre og raskere enn Javascript. Dette skulle i så fall tale for at så mye som mulig gjøres klart på tjeneren.

På den annen side er det ofte slik at bruken av de data vi sender tilbake vil endre seg i forskjellige situasjoner avhengig av bruksønsker og designbeslutninger. Det er kanskje en fordel å lage en generell tjenerfunksjon og så lage spesielle klientløsninger etter ønske. Dette taler for et fleksibelt format som peker i retning av JSON, eller XML. De to, JSON og XML, kan langt på vei sidestilles prinsippielt, men JSON gjør trolig lettere å programmere på klienten.

Nedenfor skal vi gå gjennom noen eksempler på pakking av data på tjenersiden og bruk på klientsiden. Vi skal bruke vindataene som gjennomgangseksempel. Disse er beskrevet i modulen Noen datasett. Vi skal bruke både XML-versjonen og databaseversjonen. Vi skal i alle tilfellene lage en klientside som gir oss mulighet for å velge vin av en bestemt type (rød, hvit, rose, musserende) og fra et bestemt land. Vi skal i alle løsningen bruke JSON, selv om et par av eksemplene åpenbart kunne løses med lettere med tekst og alle kunne løses med XML.

Eksempler

I alle eksemplene bruker vi et JSON format som ser slik ut, uten linjeskift:

```
{"diagnose":"OK",
 "heading":"Rødvin fra Island",
"viner":[
{"navn":"Masi","terning":"4","beskrivelse":"OK"},
{"navn":"Perequita","terning":"5","beskrivelse":"God"},
...
]}
```

Altså en diagnose som sier om dataoppslaget har gått bra eller ikke, en passelig overskrift pluss en array med vinobjekter.

I alle eksemplene bruker vi samme javascript på klienten:

-

```
ï»¿//------------- std ajax ------------------
function establishRequest()
{
    var theRequest=null;
    if (window.XMLHttpRequest)
        theRequest = new XMLHttpRequest();
    else if (window.ActiveXObject)
    {
        try { theRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch(e) {
            try {theRequest= new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch(e) {theRequest=null;}
        }
    }
    else
        theRequest=null;
    return theRequest;
}

function postRequest(theRequest,params,theUrl,targetId)
{
    theRequest.onreadystatechange =
        function( ) {
                    processRequestChange(theRequest,targetId);
        };
    theRequest.open("POST", theUrl,true);
    theRequest.setRequestHeader("Content-type",
                                "application/x-www-form-urlencoded");
    theRequest.setRequestHeader("Content-length", params.length);
    theRequest.setRequestHeader("Connection", "close");
    theRequest.send(params);
}

function doRequest(theURL,form,targetId)
{
    var myRequest=establishRequest();
    if (myRequest)
    {
        var params="vintype="+form.vintype.value+"&land="+form.land.value;
        var waiter='http://www.ia.hiof.no/~borres/common/gfx/waiter.gif';
        try{document.getElementById(targetId).innerHTML=
                        '<img src="'+waiter+'" alt=" "/>';}
        catch(E){}
        postRequest(myRequest,params,theURL,targetId);
    }
    else
      {
        alert('Browser will not do a request');
    }
}

function processRequestChange( aRequest,targetId)
{
    if (aRequest.readyState == 4) {
        if ((aRequest.status == 200) || (aRequest.status == 304))
            processJSON(aRequest.responseText,targetId);
        else
            alert("Problems accessing data:\n" + aRequest.statusText);

    }
}
//psjson
```

```
function processJSON(S,targetId)
{
    var p = null;
    try{
        p = eval("(" + S + ")");
    }
    catch(ex)
    {
        document.getElementById('result').innerHTML=
            '<p>Feil i dataformatet: '+ex.message+'</p>';
        return;
    }

    var dia=p.diagnose;
    if(dia.indexOf('OK')==0)
    {
        var vinlist=p.viner;
        // manual count due to diff between browsers on empty fields
        var count=0;
        var S='';
        for(var ix=0;ix<vinlist.length;ix++)
        {
            try
            {
              S+='<p>';
              S+='<span style="font-size:20px;color:blue;margin-right:20px">'+
                   vinlist[ix].terning+'</span>';
              S+=vinlist[ix].navn+'<br/>';
              S+=vinlist[ix].beskrivelse+'<br/>';
              S+='</p>';
              count++;
            }
            catch(e)
            {
            }
        }
        var T='<h1>'+count+' '+p.overskrift+'</h1>';
        document.getElementById(targetId).innerHTML=T+S;
    }
    else
    {
        document.getElementById(targetId).innerHTML='<pre>'+dia+'</pre>';
    }
}
//endpsjson
```

Den interessante koden biten er funksjonen processJSON:

```
function processJSON(S,targetId)
{
    var p = null;
    try{
        p = eval("(" + S + ")");
    }
    catch(ex)
    {
        document.getElementById('result').innerHTML=
            '<p>Feil i dataformatet: '+ex.message+'</p>';
        return;
    }

    var dia=p.diagnose;
    if(dia.indexOf('OK')==0)
    {
        var vinlist=p.viner;
        // manual count due to diff between browsers on empty fields
        var count=0;
        var S='';
        for(var ix=0;ix<vinlist.length;ix++)
        {
            try
            {
              S+='<p>';
              S+='<span style="font-size:20px;color:blue;margin-right:20px">'+
                   vinlist[ix].terning+'</span>';
              S+=vinlist[ix].navn+'<br/>';
              S+=vinlist[ix].beskrivelse+'<br/>';
              S+='</p>';
              count++;
            }
            catch(e)
            {
            }
        }
        var T='<h1>'+count+' '+p.overskrift+'</h1>';
        document.getElementById(targetId).innerHTML=T+S;
    }
    else
    {
        document.getElementById(targetId).innerHTML='<pre>'+dia+'</pre>';
    }
}
```

En typisk tekst som overføres og parses ser slik ut: typisk string

Eksempel 1

Vi bruker XML-data på tjeneren, skriver DOM-code som identifiserer de vinene vi er interesserte i og produserer JSON. På klientsiden lager vi HTML av JSON-formatet.

DEMO: http://www.ia.hiof.no/~borres/ml/json/page1.html

Koden på tjeneren ser slik ut:

```
#! /usr/bin/python2
# -*- coding: cp1252 -*-
import xml.dom.minidom,urllib,cgi

#-------------------------------------------------------------
# Receive: land and vintype
# Return JSON format for selected wines
#{"diagnose":"OK",
# "overskrift":"Hvite viner fra Italia",
# "viner":[
#         {"navn":"A","terning":"4","beskrivelse":"grei"},
#         {"navn":"B","terning":"4","beskrivelse":"ok"}
#         ]
#}
# Datasource:
# http://www.ia.hiof.no/~borres/commondata/vin/viner.xml
# B. Stenseth  2007
#-------------------------------------------------------------

#-----------------------------------------------------
# string for one wine
wine="""{"navn":"%s","terning":"%s","beskrivelse":"%s"}"""

#-----------------------
# downloading an xml-file as string
def loadUrl(urladdress):
    try:
        f=urllib.urlopen(urladdress)
        s=f.read()
        f.close()
        return s
    except:
        return None

#-----------------------
# collect all text in a node
def getText(nodelist):
    rc = ''
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            t=node.data.encode('ISO-8859-1')
            rc += t
    return rc

def doOneWine(vin,theCountry,theType):
    land=getText(vin.getElementsByTagName('country')[0].childNodes)
    if land != theCountry:
        return ''
    vintype=getText(vin.getElementsByTagName('type')[0].childNodes)
    if vintype != theType:
        return ''
    navn=getText(vin.getElementsByTagName('name')[0].childNodes)
    navn=navn.replace('"','\\"')
    terning=getText(vin.getElementsByTagName('dice')[0].childNodes)
    beskrivelse=getText(vin.getElementsByTagName('description')[0].childNodes)
    return wine%(navn,str(terning),beskrivelse)


# buld tree
def doit(theCountry,theType):
    try:
        vintxt=loadUrl("http://www.ia.hiof.no/~borres/commondata/vin/viner.xml")
        if vintxt==None:
            return None
```

```
        vindom = xml.dom.minidom.parseString(vintxt)
        vinlist=vindom.getElementsByTagName('wine')
        result=''
        for vin in vinlist:
            t=doOneWine(vin,theCountry,theType)
            if len(t)>0:
                result=result+t+','
        return result
    except:
        return None

#----------------------
form=cgi.FieldStorage()
print 'Content-type: text/html\n'
land='Ukjent'
vintype='Ukjent'
dia='"diagnose":"BAD"'

if form.has_key('land'):
    land=form['land'].value
if form.has_key('vintype'):
    vintype=form['vintype'].value

if vintype=='red':
    header='"overskrift":"Røde viner fra %s"'%land
elif vintype=='white':
    header='"overskrift":"Hvite viner fra %s"'%land
elif vintype=='rose':
    header='"overskrift":"Roseviner fra %s"'%land
elif vintype=='sparkling':
    header='"overskrift":"Musserende viner fra %s"'%land
else:
    header='"overskrift":"Ukjent vintype"'

viner=doit(land,vintype)
if viner !=None:
    dia='"diagnose":"OK"'

retstr='{'+dia+','+header+',"viner":['+viner[:-1]+']}'
print retstr
```

Eksempel 2

Vi bruker XML-data på tjeneren, skriver en XSLT-transformasjon som lager JSON direkte. På klientsiden lager vi HTML av JSON-formatet.

DEMO: http://www.ia.hiof.no/~borres/ml/json/page2.html

Koden på tjeneren ser slik ut:

```
#! /usr/bin/python2
# -*- coding: cp1252 -*-
import xml.dom.minidom,urllib,cgi
from Ft.Xml.Xslt import Processor
from Ft.Xml import InputSource
from Ft.Xml.InputSource  import DefaultFactory

#-------------------------------------------------------------
# Receive: land and vintype
# Return JSON format for selected wines
#{"diagnose":"OK",
# "overskrift":"Hvite viner fra Italia",
# "viner":[
#         {"navn":"A","terning":"4","beskrivelse":"grei"},
#         {"navn":"B","terning":"4","beskrivelse":"ok"}
#         ]
#}
# Datasource:
# http://www.ia.hiof.no/~borres/commondata/vin/viner.xml
# and XSLT-transformation
# http://www.ia.hiof.no/~borres/cgi-bin/json/vin/trans2.xsl
# Transformation: trans2.xsl
# B. Stenseth  2007
#-------------------------------------------------------------

#-----------------------------------------------------
# string for one wine
wine="""{"navn":"%s","terning":"%s","beskrivelse":"%s"}"""

#-----------------------
# helper to load a text file
def getTextFile(filename):
    try:
        file=open(filename,'r')
        intext=file.read()
        file.close()
        return intext
    except:
        return None

#-----------------------
# downloading an xml-file as string
def loadUrl(urladdress):
    try:
        f=urllib.urlopen(urladdress)
        s=f.read()
        f.close()
        return s
    except:
        return None

#-----------------------
def doAll(land,vintype):
    xsltstr=loadUrl('http://www.ia.hiof.no/~borres/cgi-bin/json/vin/trans2.xsl')
    xmlstr=loadUrl('http://www.ia.hiof.no/~borres/commondata/vin/viner.xml')
    xmlstr=xmlstr.strip()
    # hack to keep "
    parts=xmlstr.split('\n',1)
    xmlstr=parts[0]+'\n'+parts[1].replace('"','\\"')


    processor=Processor.Processor()
    transform= DefaultFactory.fromString(xsltstr,
                  'http://www.ia.hiof.no/~borres/cgi-bin/json/vin/trans2.xsl')
```

```
    processor.appendStylesheet(transform)


    source= InputSource.DefaultFactory.fromString(xmlstr,
                   'http://www.ia.hiof.no/~borres/commondata/vin/viner.xml')
    params={'theCountry':land,'theType':vintype}
    result= processor.run(source,topLevelParams=params)
    return result


#----------------------
form=cgi.FieldStorage()
print 'Content-type: text/html\n'
land='Portugal'
vintype='red'
dia='"diagnose":"BAD"'

if form.has_key('land'):
    land=form['land'].value
if form.has_key('vintype'):
    vintype=form['vintype'].value

if vintype=='red':
    header='"overskrift":"Røde viner fra %s"'%land
elif vintype=='white':
    header='"overskrift":"Hvite viner fra %s"'%land
elif vintype=='rose':
    header='"overskrift":"Roseviner fra %s"'%land
elif vintype=='sparkling':
    header='"overskrift":"Musserende viner fra %s"'%land
else:
    header='"overskrift":"Ukjent vintype"'

viner=doAll(land,vintype)
if len(viner) > 0:
    dia='"diagnose":"OK"'

viner=viner.decode('utf-8')
viner=viner.encode('iso-8859-1')

retstr='{'+dia+','+header+','+viner+'}'
print retstr
```

Transformasjonen er slik:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text"/>
  <xsl:param name="theCountry" select="'Frankrike'"/>
  <xsl:param name="theType" select="'red'"/>

  <xsl:template match="/">
    "viner":[
    <xsl:apply-templates select="//wine[country=$theCountry and type=$theType]">
      <xsl:sort order="descending" select="dice"/>
    </xsl:apply-templates>
    ]
  </xsl:template>

  <xsl:template match="//wine">
    {"terning":"<xsl:value-of select="dice"/>",
     "navn":"<xsl:value-of select="name"/>",
    "beskrivelse":"<xsl:value-of select="description"/>"},
```

```
  </xsl:template>

</xsl:stylesheet>
```

Eksempel 3

Vi bruker en database på tjeneren og lager JSON basert på de recordene vi plukker opp med en SQL-setning. På klientsiden lager vi HTML av JSON-formatet.

DEMO: http://www.ia.hiof.no/~borres/ml/json/page3.html

Koden på tjeneren ser slik ut:

```
#! /usr/bin/python2
import MySQLdb,cgi,urllib

#-------------------------------------------------------------
# Receive: land and vintype
# Return JSON format for selected wines
#{"diagnose":"OK",
# "overskrift":"Hvite viner fra Italia",
# "viner":[
#         {"navn":"A","terning":"4","beskrivelse":"grei"},
#         {"navn":"B","terning":"4","beskrivelse":"ok"}
#         ]
#}
# Datasource: mySQL base as described in
# http://www.ia.hiof.no/~borres/ml/datasett/p-datasett.html
# B. Stenseth  2007
#-------------------------------------------------------------

#-----------------------------------------------
# connect and execute a sql-request
#-----------------------------------------------
def connectAndExecute(sql):
    try:
        myBase=MySQLdb.connect(host='frigg.hiof.no',
                               user='student',
                               passwd='student',
                               db='vin')
        myTab=myBase.cursor()
        myTab.execute(sql)
        myBase.close()
        return myTab.fetchall()
    except MySQLdb.Error, e:
        print "Error %d: %s" % (e.args[0], e.args[1])
        return None

#-----------------------------------------------
# do the job
#-----------------------------------------------
def doAll(land,vintype):
    resultTxt=''
    sql="""select name,dice,description from wines
           where country='%s' and type='%s'
           order by dice desc;"""
    result=connectAndExecute(sql%(land,vintype))
    if result==None:
        return None
    wine="""{"navn":"%s","terning":"%s","beskrivelse":"%s"}"""
    for row in result:
        navn=str(row[0])
        navn=navn.replace('"','\\"')
        terning=str(row[1])
        beskrivelse=str(row[2])
        resultTxt=resultTxt+wine%(navn,terning,beskrivelse)+','
    return resultTxt

#----------------------
form=cgi.FieldStorage()
print 'Content-type: text/html\n'
land='Ukjent'
vintype='Ukjent'
dia='"diagnose":"BAD"'

if form.has_key('land'):
    land=form['land'].value
```

```
if form.has_key('vintype'):
    vintype=form['vintype'].value

if vintype=='red':
    header='"overskrift":"Rode viner fra %s"'%land
elif vintype=='white':
    header='"overskrift":"Hvite viner fra %s"'%land
elif vintype=='rose':
    header='"overskrift":"Roseviner fra %s"'%land
elif vintype=='sparkling':
    header='"overskrift":"Musserende viner fra %s"'%land
else:
    header='"overskrift":"Ukjent vintype"'

viner=doAll(land,vintype)
if viner != None:
    dia='"diagnose":"OK"'

retstr='{'+dia+','+header+',"viner":['+viner[:-1]+']}'
print retstr
```

Referanser og eksempler:

- [1] JSON:
  (JavaScript Object Notation)
  www.json.org/

Dataformater

Hva:

Hvordan pakke data som går fram og tilbake

Vi må ta stilling til hva slags dataformat vi skal bruke på de dataene vi henter og sender med i en request. Hvis vi ser isolert på hvordan XMLHttpRequest er implementert vet vi at vi kan hente data på 2 måter, enten som ren tekst (myRequest.responseText) eller som et DOM-tre (myRequest.responseXML). Vi vet videre at dersom vi plasserer velformet tekst på HTML-siden så parses og tolkes dette av nettleseren slik at det blir en del av dokumentets DOM-struktur.

Når klienten sender en forspørsel vet vi at parametere som sendes med må sendes i formatet: navn=verdi, eventuelt flere slike par, skilt med ?. Dette gjelder enten vi bruker GET eller POST.

Gitt disse rammene, har vi fortsatt stor handlefrihet i hvordan vi organiserer data. Organiseringen er selvsagt tett koplet til hvordan vi ønsker å fordele arbeidet mellom klient og tjener.

Vi prøver oss med en oversikt over strategier, fordeler og ulemper.

Format fra en forspørsel	Fordeler	Ulemper
XML-format	• Vi kan sende en forspørsel som besvarer mange spørsmål. DOM-strukturen gir oss et godt verktøy for å plukke ut det vi skal plassere på forskjellige steder på siden. • Vi trenger liten behandling på tjenersiden dersom vi har et XML-format som datakilde. • Forenkler den situasjonen at vi skal legge til nye komponenter i et element uten å fjerne de gamle.	• DOM-programmering er mer komplisert enn ren tekstflytting. Javascript er ikke det beste språket å programmere DOM i, blandt annet må vi ta hensyn til noen nettleserproblemer.
HTML-formatert text	• Det er svært enkelt å plassere teksten som et HTML-fragment på siden. • Vi kan bruke kraftige verktøy som XSLT og velprøvde DOM-biblioteker på tjenersiden for å formatere dataene. Dette forutsetter selvsagt at vi har kontroll og tilgjengelige verktøy på tjenersiden	• Vi får en klump med data som det er vanskelig å gjøre noe med på klientsiden, annet enn å plassere den i et element.
Kommaseparert text	• Vi står fritt til å sende tekst fra tjeneren som inneholder alternativer og data som er	• Vi må til med ganske mye parsing i Javascript.

	beskrivende og skal plasseres avhengig av innhold. • Vi kan lage DOM-struktur av kommaseparert tekst og dermed distribuere dataene som fragmenter på siden.	
JSON (mer om dette i modulen JSON)	• Vi trenger ikke bekymre oss for data som sådan (tekst eller DOM). Vi har data tilgjengelig som programobjekter og felter i programobjekter.	• Vi må sette oss inn i objektstrukturen på de involverte språkene, blandt annet Javascript på klientsiden. • Vi er avhengige av parsing- og pakkerutiner som må fungere etter oppskriften på klienten (Javascript) og tjeneren (f.eks. Python).

Referanser og eksempler:

JSON: www.json.org/

Caching

Hva:

Hvis nettelseren cacher en URL

Nettlesere lagrer ofte en vevside lokalt på din maskin slik at den er raskt tilgjengelig hvis du vil ha den opp igjen. Denne strategien (caching) er bra med tanke på rask tilgang til en side, men den er ikke så bra dersom siden er endret siden sist. De fleste nettlesere har muligheter for å stille inn hvordan denne strategien skal gjennomføres. F.eks. kan du tømme cachen når du ønsker, eller du kan gi beskjed om at den skal tømmes ved jevne mellomrom eller når nettleseren avslutter eller når et dokument i cachen har nådd en viss alder.

Når vi lager AJAX-løsninger er caching av og til et problem. Internett explorer har automatisk caching skrudd på. Konsekvensen er at dersom du gjør en XMLHttprequest med den samme adressen flere ganger, får du den cachede versjonen. Du får det samme som du fikk sist. Det er ikke alltid vi ønsker dette. Det kan hende at dataene vi spør etter er endret av andre programmer, eller det kan hende vi har en eller annen autentiseringsstrategi som krever flere oppslag eller at vi overlater til serverskriptet å lage et tilfeldig eller tidsavhengig datasett som retur.

Vi kan endre IE's cahcing strategi og vi kan tømme cachen for å teste at løsningen vår fungerer, men dersom løsningen skal fungere for brukere som ikke er interesserte i å sette seg inn våre cachingproblemer, så må vi finne på noe annet.

Random parameter

En enkel løsning er å sende med en ekstra "tilfeldig" parameter i forespørselen.

```
http://www.ia.hiof.no/~borres/cgi-bin/dagenstext.py?language=latin
```

er forskejllig fra

```
http://www.ia.hiof.no/~borres/cgi-bin/dagenstext.py?language=latin&tull=x
```

selv om "tull=x" er meningsløs både for vevsiden som sender forespørselen og for skriptet som skal betjene den.

En funksjon som generalisere en slik løsning, med å lage en ny, unik, dummyparameter kan være f.eks. slik :

```
function uncache(url){
    var d = new Date();
    var time = d.getTime();
    return url + '&time='+time;
}
```

"time" trenger ikke ha noen mening for kommuniksajonen mellom nettleser og tjener, men har den egenskapen at den blir forskjellig for hver gang vi lager den. Et alternativ kan være slik:

```
function uncache(url){
    return url + '&tull='+Math.random()
}
```

Siden random-funksjonen i Javascript tar utgangspunkt (seed) i tidsfunksjonen, vil dette for vårt formål gjøre samme jobben.

DEMO: http://www.ia.hiof.no/~borres/ml/ajaxmore/caching/nocache1.html

Styring av caching

Hvis den kilden vi etterspør er et skript, kan vi sende tilbake en header som kontrollerer hvordan nettleseren skal forholde seg til caching. Hvis vi har et Pythonskript kan vi skrive tilbake:

```
print "Cache-Control: no-cache, must-revalidate\nContent-type: text/html\n\n"
```

alt som en del av headeren, som det framgår.

DEMO: http://www.ia.hiof.no/~borres/ml/ajaxmore/caching/nocache2.html

Merk: at det er noen problemer med å unngå caching i MSIE når vi bruker GET. Et alternativ er å bruke POST. POST cacher ikke. Det er også mulig å bruke andre Cache-Control- verdier, som Max-Age. Se HTTP-dokumentasjon for å studere alternativene.

DEMO: http://www.ia.hiof.no/~borres/ml/ajaxmore/caching/nocache3.html

Referanser og eksempler:

Microsoft og caching: support.microsoft.com/kb/222064

W3C om HTTP www.w3.org/Protocols/

AJAX fra andre domener

Hva:

Hvordan hente data fra andre domener og plassere dem på en webside.

Vi skal bruke vevstedet GeoNames [1] som eksempel. geonames tilby en rekke tjenseter som har med geografisk informasjon å gjøre. Vi kan laste ned filer og databaser og vi kan bruke webservices. Begrepet webservice brukes i romslig betydning, og innebærer ikke SOAP- eller WSDL-formater. Det vi kan gjøre er rett og slett å bruke en url med parametere for å hente ferdigpakkede data. Pakkene er enten JSON eller XML.

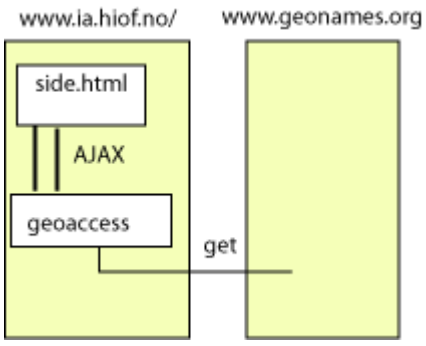
Du kan eksperimentere med de interaktive tjenestene på geonames, eller du kan forsøke å kopiere følgende URL'er i nettleserens addressfelt:

```
ws.geonames.org/postalCodeSearch?postalcode=1850&country=NO
ws.geonames.org/postalCodeSearchJSON?postalcode=1880&country=NO
ws.geonames.org/findNearbyPlaceName?lat=59.5166667&lng=11.2666667&radius=20
ws.geonames.org/findNearbyPlaceNameJSON?lat=59.5166667&lng=11.2666667&radius=20
```

Dette er bare noen av de tjenestene som er tigjengelige. Parameterne i eksemplene ovenfor er vel rimelig selvforklarende. I en rekke av de andre tjenestene må vi sette oss litt inn i hvordan spørsmål kan parametriseres og hvordan svarene skal tolkes.

Hvis jeg nå ønsker å bruke noen av tjenestene på mine vevsider, er det selvsagt fristende å bruke AJAX. De fleste tjenestene er slik at de gir svaret enten i JSON- eller XML-format. Problemet er at tjenesten ligger i et annet domene enn min vevside, og jeg kan ikke uten videre sende en XMLHttpRequest. Vi skal se på to måter å løse dette på.

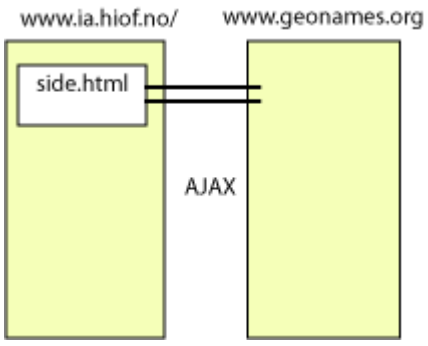
Indirekte



Vi skriver et program på tjeneren, geoaccess, som henter data fra geoNames. Vi kommuniserer fra vevsiden til dette programmet med AJAX.

Dette innbærer at AJAX-kommunikasjonen foregår innen samme domene.

Direkte



Vi bruker en litt tvilsom strategi som gjør det mulig å hente materialet direkt via AJAX.

Dette innbærer at AJAX-kommunikasjonen foregår på tvers av domener.

Som eksempel bruker vi for enkelhets skyld tjenesten findNearbyPlaceNameJSON. Vi vil altså finne vilke steder som ligger innenfor en bestemt radius fra de koordinatene som er registrert for et beestemt postnummer. Vi holder oss til norske postnummere.

Indirekte

Vi begynner med et skript som skal gå på tjeneren. Skriptet er laget slik at det videreformidler en forespørsel, tar mot svaret og sender det direkte tilbake til vevsiden. Skriptet er laget så generelt som mulig og er ikke begrenset til den oppgaven vi har satt oss.

```
#! /usr/bin/python2

import urllib
import cgi
import cgitb; cgitb.enable()

# Attempt to convey an AJAX-request to
# geonames as general and smooth as possible

form=cgi.FieldStorage()
print 'Content-type: text/html\n'

#expect to find: function and any set of parameters
function='postalCodeSearch'
if form.has_key('function'):
    function=form['function'].value
lookUp='http://ws.geonames.org/'+function

ks=form.keys()

params=''
for k in ks:
    if k=='function':
        continue
    params+=k+'='+form[k].value+'&'

if len(ks) > 1:
    params=params[:-1]

lookUp+='?'+params
try:
    f=urllib.urlopen(lookUp)
    S=f.read()
    f.close()
    S=S.decode('utf')
    S=S.encode('iso-8859-1')
    print S
except:
    print 'cannot load'
```

Fra vevsiden bruker vi skriptet slik:

```
var theURL=
 'http://www.ia.hiof.no/~borres/cgi-bin/ajax/geonames/geoaccess.py';
function establishRequest()
{
    var theRequest=null;
    if (window.XMLHttpRequest)
        theRequest = new XMLHttpRequest();
    else if (window.ActiveXObject)
    {
        try { theRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch(e) {
            try {theRequest= new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch(e) {theRequest=null;}
        }
    }
    else
        theRequest=null;
    return theRequest;
}

function findNearbyPostalCodesJSON(postnummer,radius)
{
    var myRequest=establishRequest();
    if (myRequest)
    {
        var params='function=findNearbyPostalCodesJSON&country=NO&postalcode='
                    +postnummer+'&radius='+radius;
        myRequest.onreadystatechange =
        function( ) {
          processNearbyPlaceNameJSON(myRequest);
        };
        myRequest.open("POST", theURL,true);
        myRequest.setRequestHeader("Content-type",
                                   "application/x-www-form-urlencoded");
        myRequest.setRequestHeader("Content-length", params.length);
        myRequest.setRequestHeader("Connection", "close");
        myRequest.send(params);
    }
    else
    {
        alert('Browser will not do a request');
    }
}

function processNearbyPlaceNameJSON(aRequest)
{
    if (aRequest.readyState == 4) {
        if ((aRequest.status == 200) || (aRequest.status == 304))
        {
            try{
                p=eval('('+aRequest.responseText+')');
                var pcodes= p.postalCodes;
                T=''
                for(ix=0;ix<pcodes.length;ix++)
                {
                    var navn=pcodes[ix].placeName+'<br/>';
                    if(T.indexOf(navn)==-1)
                        T+=navn;
                }
                document.getElementById('result').innerHTML=T;
            }
            catch(e)
```

```
            {
                document.getElementById('result').innerHTML=
                                    'Finner ikke noe';
            }
        }
        else
            alert("Problems accessing data:\n" + aRequest.statusText);
    }
}
```

DEMO: http://www.ia.hiof.no/~borres/ml/crossdomain/indirect.html

Direkte

Nøkkelen er følgende script av Jason(!) Levitt [2] . Sikkerhetsadvarselen er lagt til i den versjonen som anbefales brukes av geonames.

```
// JSONscriptRequest -- a simple class for accessing Yahoo! Web Services
// using dynamically generated script tags and JSON
//
// Author: Jason Levitt
// Date: December 7th, 2005
//
// A SECURITY WARNING FROM DOUGLAS CROCKFORD:
// "The dynamic <script> tag hack suffers from a problem. It allows a page
// to access data from any server in the web, which is really useful.
// Unfortunately, the data is returned in the form of a script. That script
// can deliver the data, but it runs with the same authority as scripts on
// the base page, so it is able to steal cookies or misuse the authorization
// of the user with the server. A rogue script can do destructive things to
// the relationship between the user and the base server."
//
// So, be extremely cautious in your use of this script.
//

// Constructor -- pass a REST request URL to the constructor
//
function JSONscriptRequest(fullUrl) {
    // REST request path
    this.fullUrl = fullUrl;
    // Keep IE from caching requests
    this.noCacheIE = '&noCacheIE=' + (new Date()).getTime();
    // Get the DOM location to put the script tag
    this.headLoc = document.getElementsByTagName("head").item(0);
    // Generate a unique script tag id
    this.scriptId = 'YJscriptId' + JSONscriptRequest.scriptCounter++;
}

// Static script ID counter
JSONscriptRequest.scriptCounter = 1;

// buildScriptTag method
//
JSONscriptRequest.prototype.buildScriptTag = function () {

    // Create the script tag
    this.scriptObj = document.createElement("script");

    // Add script object attributes
    this.scriptObj.setAttribute("type", "text/javascript");
    this.scriptObj.setAttribute("src", this.fullUrl + this.noCacheIE);
    this.scriptObj.setAttribute("id", this.scriptId);
}

// removeScriptTag method
//
JSONscriptRequest.prototype.removeScriptTag = function () {
    // Destroy the script tag
    this.headLoc.removeChild(this.scriptObj);
}

// addScriptTag method
//
JSONscriptRequest.prototype.addScriptTag = function () {
    // Create the script tag
    this.headLoc.appendChild(this.scriptObj);
}
```

Vi bruker dette biblioteket fra vår egen javascriptkode slik:

```
// this function will be called by our JSON callback
// the parameter jData will contain an array with postalcode objects
function processNearbyPlaceNameJSON(jData)
{
  if (jData == null) {
    document.getElementById('result').innerHTML='ikke noe svar';
    return;
  }
  try{
      var pcodes = jData.postalCodes;
      if (pcodes.length > 0)
      {
            T=''
            for(ix=0;ix<pcodes.length;ix++)
            {
                var navn=pcodes[ix].placeName+'<br/>';
                if(T.indexOf(navn)==-1)
                    T+=navn;
            }
            document.getElementById('result').innerHTML=T;
      }
      else
          document.getElementById('result').innerHTML='fant ikke noe';

    }
    catch(E)
    {
          document.getElementById('result').innerHTML='feil spørsmål';
    }
}

function findNearbyPostalCodesJSON(pcode,radius)
{

  request = 'http://ws.geonames.org/findNearbyPostalCodesJSON?postalcode='
            + pcode  + '&country=NO&radius='+radius
            + '&callback=processNearbyPlaceNameJSON';
  // Create a new script object
  aObj = new JSONscriptRequest(request);
  // Build the script tag and execute it
  aObj.buildScriptTag();
  aObj.addScriptTag();
}
```

DEMO: http://www.ia.hiof.no/~borres/ml/crossdomain/direct.html

Forklaring

Dersom vi skal oppsummere kort hvordan denne koden virker så må det bli omtrent slik:

1. Vi etablerer et scriptelement som dersom vi skulle skrive det i klartekst blir slik(jeg har lagt inn linjeskift av praktiske årsaker):
   ```
   <script
     type="text/javascript"
     src="http://ws.geonames.org/findNearbyPostalCodesJSON?
          postalcode=1850&
          country=NO&
          radius=2&
          callback=processNearbyPlaceNameJSON&
          noCacheIE=1193327617484"
     id="YJscriptId1">
   ```

```
</script>
```

Dette bygges opp ved hjelp av funksjonene: JSONscriptRequest(request) og buildScriptTag(). Merk at navnet på den funksjonen som skal ta mot resultatet inngår i parameter lista: callback=processNearbyPlaceNameJSON.

2. I det øyeblikk vi setter inn dette script-elementet, addScriptTag(), som barn til head-elementet blir det evaluert. Denne evalueringen innebærer at nettleseren forsøker å hente skriptet. Nøkkelen til å forstå hva som skjer er en analyse av hva som kommer tilbake. Med parametrene ovenfor vil vi få følgende tilbake (jeg har lagt inn linjeskift av praktiske årsaker):

```
processNearbyPlaceNameJSON(
{"postalCodes":[
        {"adminName2":"EIDSBERG","adminCode2":"0125",
         "adminCode1":"01","postalCode":"1850",
         "countryCode":"NO","lng":11.3333333,"placeName":"Mysen","lat":59.55,
         "adminName1":"Oestfold"},{"adminName2":"EIDSBERG","adminCode2":"0125",
         "adminCode1":"01","postalCode":"1851",
         "countryCode":"NO","lng":11.3333333,"placeName":"Mysen","lat":59.55,
         "adminName1":"Oestfold"},
         {"adminName2":"EIDSBERG","adminCode2":"0125",
          "adminCode1":"01","postalCode":"1859",
          "countryCode":"NO","lng":11.3333333,"placeName":"Slitu","lat":59.55,
          "adminName1":"Oestfold"}
           ]});
```

Det vil si at vi får tilbake et funksjonskall med et JSON-objekt som parameter. GeoNames-serveren produserer dette formatet, og kan gjøre det fordi vi har sendt over navnet på vår mottagerfunksjon som parameter.

3. Funksjonskallet effektueres siden det oppfattes om en JavaSciptlinje, som skal utføres på linje med alle JavaScript-setninger som parses i scriptelementet. Og vår funksjon, processNearbyPlaceNameJSON, utføres. Merk at parameteren som er et JSON-objekt som også evalueres.

Vi ser at det ikke er noe XMLHttpRequest involvert. Alt er basert på at vi "sniker" oss utenfor kontrollmekanismene ved å manipulere dom-treet ved å sette in et scriptelemment som evalueres.

Eksperiment (lokalt)

Vi forsøker å anvende denne teknikken mot en serverfunksjon vi skriver selv. Vi gjør dette på samme domene som web-siden, altså ikke fremmed domene.

Vi skriver følgende script på tjeneren:

```
#! /usr/bin/python2
import cgi
print 'Content-type: text/plain\n'
form=cgi.FieldStorage()
callback='error'
if form.has_key('callback'):
    callback=form['callback'].value
print callback+'({"melding":"Halleluja"});'
```

Vi bruker biblioteket fra javascript koden over, og skriver funksjonene:

```
function takeIt(melding)
{
        alert('Im back, '+melding.melding);
}
function doRequest(theURL)
{
  request = theURL+'?callback=takeIt';
  // Create a new script object
  aObj = new JSONscriptRequest(request);
  // Build the script tag and execute it
  aObj.buildScriptTag();
  aObj.addScriptTag();
}
```

DEMO: http://www.ia.hiof.no/~borres/cgi-bin/ajax/callback/index2.html

En enda enklere løsning

```
#! /usr/bin/python2
import cgi
print 'Content-type: text/plain\n'
form=cgi.FieldStorage()
callback='error'
if form.has_key('callback'):
    callback=form['callback'].value
print callback+'("Halleluja");'
function doRequest(theURL)
{
  request = theURL+'?callback=alert';
  // Create a new script object
  aObj = new JSONscriptRequest(request);
  // Build the script tag and execute it
  aObj.buildScriptTag();
  aObj.addScriptTag();
}
```

DEMO: http://www.ia.hiof.no/~borres/cgi-bin/ajax/callback/index3.html

Tja

Hvilken av teknikkene skal vi bruke ? Det er fristende å bruke den direkte metoden. Den krever lite kode og er åpenbart rask.

Det er imidlertid noen negative momenter/bergrensninger som det er verdt å merke seg:

- teknikken omgår de sikkerhetselementene som er innebygd i domenegrensene for AJAX.
- Kan ikke motta responsekoder, fungerer bare når forespørselene returnerer kode: 200.
- Fungerer bare med get, og bare asynkront.
- vi ser bort fra de mulighetene vi har for datakontroll og formatering som vi lett kan realisere i et tjenerskript.

Referanser og eksempler:

- [1] Geonames:
  www.geonames.org/
- [2] Jason Levitt:
  JSON and the Dynamic Script Tag
  www.xml.com/pub/a/2005/12/21/json-dynamic-script-tag.html
- Cross Domain Ajax: a Quick Summary: snook.ca/archives/javascript/cross_domain_aj/
- Modulen AJAX
- Modulen JSON

Testing av Javascript

med JsUnit

Hva:

Hvordan teste Javascript

Når vi jobber med javascript er situasjonen litt anneledes enn når vi jobber med andre språk. Javascript er interpretert og vi har ingen gode utviklingsmiljøer. Vi skriver som regel kode i en editor som beste fall har fargekoding for syntaksen og vi tester ved å kjøre en HTML-side i en nettleser. Ofte er de feilene vi sliter med en kombinasjon av selve Javascriptkoden og det dokumentet (HTML-siden) koden arbeider mot. Det er to hjelpemidler som er lett til hånds:

- De fleste nettleserne har en funksjon som gjør at vi lett finner syntaksfeil, unntak vi har glemt å fange opp, nullpekere osv. Min favoritt er FireFox sin Error Console, menyen Tools - Error Console.
- Alert. Vi kan spore interpreteringen av koden ved å kalle alert i kritiske stadier, typisk: alert('Verdien til T. '+T);. Det er ganske masete å flytte slike alert-kall rundt i koden, men det kan være ganske effektivt.

Det finnes Unit-løsninger også for Javascript, slik som for Java, Python, .Net-språkene osv.. Vi skal se litt på en av disse i denne modulen.

JsUnit

Vi skal altså holde oss i "Unit"-tradisjonen og se på JsUnit for testing av Javascriptnkode. Merk at det er flere implementasjoner av xxUnit for Jacvascripttesting. Jeg holder meg til JsUnit [1] . Jeg har ikke grunnlag for å mene at denne er bedre enn andre. Det er bare å lete og prøve seg fram.

JsUnit må lastes ned og pakkes ut på din lokale maskin, eller den kan installeres på tjeneren. Vi holder oss til den første settingen her. Selve testingen foregår ved at vi laser opp en side i nettleseren og ber "siden" teste en side med Javascript som vi har skrevet. Testeren, testRunner.html, ser slik ut:

testrunner.gif

Utfordringen vår blir å lage de testsidene som vi kan laste opp og teste. En slik testside må bestå av:

- Inkludering av testkode
- Inkludering av vår egen kode som vi vil teste.
- Tester

Dokumentasjonssidene til JsUnit er vel ikke verdens beste, men med litt inspeksjon og prøving og feiling er det ganske kurant å sette opp enkle tester. Det er viktig å merke seg det dokumentasjonen sier om noen krav enkelte nettlesere stiller.

Et typisk eksempel på en testside kan se slik ut:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JsUnit test Sample0</title>
<!-- JsUnit library -->
<script language="JavaScript" type="text/javascript"
        src="file:///C:\jsunit\app\jsUnitCore.js"></script>
<!-- my code -->
<script language="JavaScript" type="text/javascript"
        src="myCode.js"></script>
<!-- testing -->
<script language="JavaScript" type="text/javascript">
    function test1() {assert(true);}
    function test2() {assert(true);}
</script>
  </head>
  <body>
    <h1>Test nothing</h1>
    <p>with JsUnit</p>
  </body>
</html>
```

Eksempel 1

Vi lager et eksempel med et svært enkelt oppsett. Det som er viktig å merke seg er at funksjoner med navn som begynner på "test" blir automatisk kjørt av testRunner.

Utgangspunktet er følgende "kravspesifikasjon":

> Vi ønsker å skrive en funksjon, convert, som konverterer et naturlig tall til en sekvens av tallord.

Vi begynner med å lage en testfunksjon:

```
function test1(){
    assertEqual(convert(123),'en to tre');
    assertEqual(convert(2.3),'ikke et naturlig tall');
    assertEqual(convert('jensen'),'ikke et naturlig tall');
}
```

Første versjon av koden vår, convert, lager vi slik:

```
function convert(T){
    return 'nonsens';
}
```

Vi kan da lage følgende testside:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JsUnit test Sample0</title>
<!-- JsUnit library -->
<script language="JavaScript" type="text/javascript"
    src="file:///C:\jsunit\app\jsUnitCore.js"></script>
<!-- my code with function convert -->
<script language="JavaScript" type="text/javascript"
    src="convertCode.js"></script>
<!-- testing -->
<script language="JavaScript" type="text/javascript">
    function test1(){
        assertEquals("2.3",convert(2.3),"ikke et naturlig tall");
        assertEquals("jensen",convert("jensen"),"ikke et naturlig tall");
        assertEquals("123",convert(123),"en to tre");
    }
</script>
  </head>
  <body>
    <h1>Test number conversion</h1>
    <p>with JsUnit</p>
  </body>
</html>
```

Denne feiler selvsagt, og testRunner gir følgende diagnose:

```
 Tests with problems (1 total) - JsUnit
 ...
 1. sample1.html:test1 failed
 Expected nonsens (string) but was en to tre (string)
 ...
```

Vi arbeider litt mer med funksjonen vår, convert. Vi tester og feiler og kommer etter hvert fram til følgende innhold i fila convertCode.js:

```
var numbers="0123456789";
var wds = new Array("null","en","to","tre","fire","fem","seks","sju","åtte","ni");
function convert(T){
    var S=''+T;
    var res='';
    for (ix=0;ix<S.length;ix++)
    {
        var pos =numbers.indexOf(S.substring(ix,ix+1));
        if(pos==-1)
            return "ikke et naturlig tall";
        res+=wds[pos]+' ';
    }
    return res.substring(0,res.length-1);
}
```

Når vi tester denne får vi ingen feil.

Eksempel 2

I eksempelet ovenfor laged vi en testside som inkluderte Javascriptkoden vi ønsket å teste. I dette eksempelet utvider vi problemet slik at vi involverer samspillet mellom Javascriptkoden og HTML-siden.

Utgangspunktet er følgende "kravspesifikasjon":

> Vi ønsker å lage en side der brukeren kan skrive inn naturlige heltall og få disse addert til en synlig sum.

Første utkast: http://www.ia.hiof.no/~borres/ml/jstest/sample20.html

Kildekoden er slik:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
    <title>page to test</title>
    <!-- testcode. remove when tested -->
        <script language="JavaScript" type="text/javascript"
            src="file:///C:\jsunit\app\jsUnitCore.js"></script>
        <script language="JavaScript" type="text/javascript"
            src="testadder0.js"></script>
    <!-- end of testcode. remove when tested -->
    <script language="JavaScript" type="text/javascript"
        src="myAdder0.js"></script>
</head>
<body>
    <h1>Min sumside</h2>
    <p>Summen er: <span id="sum">0</span></p>
    <form action="#">
    <div>
    Nytt tall:
     <input type="text" id="tall"
            name="tall" value="0"
            maxlength="3" size="4"/>
    <input type="button" value="Add"
           onclick="add(this.form.tall.value);return false;"/>
    </div>
    <div id="error" style="color:red">
    </div>
    </form>
</body>
</html>
```

Der vår kode, myAdder.js, er slik:

```
function add(tallS){}
```

Jobben blir å skrive fila: testadder.js og funksjonen add().

Vi begynner med første utkast til test, testadder0.js. Vi lager den slik:

```
function exposeTestFunctionNames()
{
    return ['test1'];
}

function test1()
{
    document.getElementById('sum').innerHTML='1';
    add("1");
    assertEquals("2",document.getElementById('sum').innerHTML);
}
```

Siden vi ikke har fylt funksjonen add() med noe fornuftig feiler testen:

```
Tests with problems (1 total) - JsUnit
...
1. sample20.html:test1 failed

Expected 2 (string) but was 1 (string)
...
```

Vi lager en mer sofistikert test før vi begynner med å utvikle funksjonen add().

```
function exposeTestFunctionNames()
{
    return ['test1','test2','test3','test4'];
}


function test1()
{
    document.getElementById('sum').innerHTML='1';
    add("1");
    assertEquals("2",document.getElementById('sum').innerHTML);
}

function test2()
{
    document.getElementById('sum').innerHTML='1';
    add("jensen");
    assertEquals("1",document.getElementById('sum').innerHTML);
}

function test3()
{
    document.getElementById('sum').innerHTML='1';
    add("2.3");
    assertEquals("1",document.getElementById('sum').innerHTML);
}
function test4()
{
    document.getElementById('sum').innerHTML='1';
    add("2TT");
    assertEquals("1",document.getElementById('sum').innerHTML);
}
```

Så setter vi fokus på add()-funksjonen. Vi prøver oss med følgende:

```
function add(tallS)
{
    var sumS=document.getElementById('sum').innerHTML;
    var sum=parseInt(sumS);
    var tall=parseInt(tallS);
    document.getElementById('error').innerHTML='';
    if (isNaN(tall))
    {
        document.getElementById('error').innerHTML='kun heltall';
        return;
    }
    document.getElementById('sum').innerHTML=''+(sum+tall);
}
```

test3 og test4 feiler siden parseInt() i Javascript tolker siffere "så lang den kan". Vi må altså plukke opp de situasjonene der inputteksten begynner på et siffer, men inneholder noe annet etterpå. Vi skriver om funksjonen add():

```
function add(tallS)
{
    var sumS=document.getElementById('sum').innerHTML;
    var sum=parseInt(sumS);
    var tall=parseInt(tallS);
    document.getElementById('error').innerHTML='';
    if (isNaN(tall))
    {
        document.getElementById('error').innerHTML='kun heltall';
        return;
    }
    for(ix=0;ix<tallS.length;ix++)
        if('0123456789'.indexOf(tallS.substring(ix,ix+1))==-1)
        {
            document.getElementById('error').innerHTML='kun heltall';
            return;
        }
    document.getElementById('sum').innerHTML=''+(sum+tall);
}
```

Ferdig HTML-side: http://www.ia.hiof.no/~borres/ml/jstest/sample21.html

Referanser og eksempler:

- [1] JSUnit:
  Testing av JavaScript
  www.jsunit.net/

Picasso

3 måter å gjøre det samme på

Hva:

Hvordan velge språk på en side



Vi skal ta for oss en enkel side med et bilde og en kort historie om Picasso. Vi ønsker å framstille historien, teksten, på forskjellige språk. Brukeren skal enkelt kunne velge språk. Problemet blir løst på tre forskjellige måter:

1. Programmering av stilsett. Alle tekstene lastes opp. Brukerens valg fører til at vi endrer aktivt stilsett (CSS) for å vise den ønskede teksten.
2. Skjulte felter (div'er). Alt lastes opp. Brukerens valg fører til at vi henter fram den ønskede skjulte tekstblokken.
3. AJAX. Vi henter den ønskede teksten fra tjeneren etter behov.

Oversettelsene er gjort fra fransk i Alta Vistas Babel Fish og er vel heller tvilsomme som språklige perler. Det er laget oversettelser fra fransk til engelsk, spansk, italiensk og tysk.

Programmerte stilsett

Strategien er at vi lager et stilsett for hver av de aktuelle språkene. Når brukeren velger forsøker vi å aktivisere det aktuelle stilsettet. Stilsettene har inegn annen funksjon enn å skjule alle andre tekster enn den vi vil vise fram

Stilsettet som viser fram engelsk tekst er slik:

```
.rightimage{float:right; margin: 0 0 10px 10px;padding:2px;}
.english{display:visible;}
.german,.spanish,.italian{display:none}
```

Det som viser fram tysk er slik:

```
.rightimage{float:right; margin: 0 0 10px 10px;padding:2px;}
.german{display:visible;}
.english,.spanish,.italian{display:none}
```

Tilsvarende for fransk, italiensk og spansk.

De 5 stilsettene lenkes til i headeren på siden slik:

```
<link rel="stylesheet" href="styleFrench.css">
<link rel="stylesheet" href="styleEnglish.css">
<link rel="stylesheet" href="styleGerman.css">
<link rel="stylesheet" href="styleSpansish.css">
<link rel="stylesheet" href="styleItalian.css">
```

Utfordringen blir da å skrive en Javascript-funksjon som aktiviserer det riktige stilsettet:

```
 function chooseStyle(theChoice){
  if(document.styleSheets)
    for (i=0;i<document.styleSheets.length;i++)
       if (i==theChoice)
        document.styleSheets[i].disabled=false;
       else
        document.styleSheets[i].disabled=true;
}
```

Hvordan vi kaller denne funksjonen avhenger av hvordan vi lager layout på siden. På eksempelt du kan teste er det enkle stringer som kan klikkes.

DEMO: http://www.ia.hiof.no/~borres/ml/picasso/picasso_with_style.html

Skjulte felt

Strategien her er også å laste opp all teksten med en gang og skjule det vi ikke ønsker å se. Men her flytter vi den ønskede skjulte teksten til et element der den blir synlig. Prinsippielt blir det slik:

```
...
  <div>
    <img class="rightimage" src="Le-fauteuil-rouge.gif" alt="Le-fauteuil-rouge"/>
    <div id="show">
    </div>
  </div>

  <div style="display:none">
      <div id="french">
        Picasso poursuit ...
      </div>
      <div id="english">
        Picasso continues ...
      </div>
      ...
  </div>
```

JavaScriptet som henter en skjult tekst til "show"-elementet ser slik ut:

```
  function showLanguage(languageID){
    T=document.getElementById(languageID).innerHTML;
    document.getElementById('show').innerHTML=T;
```

DEMO: http://www.ia.hiof.no/~borres/ml/picasso/picasso_hidden.html

AJAX

Her er strategien å hente den teksten vi ønsker når det er behov for den. Vi henter de filene vi trenger direkte.

Javascriptkoden blir slik:

```
var myRequest;

function fetchLanguage(language)
{
    if (window.XMLHttpRequest)
        myRequest = new XMLHttpRequest();
    else if (window.ActiveXObject)
    {
        try { myRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch(e) {
            try {myRequest= new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch(e) {myRequest=null;}
        }
    }
    else
        myRequest=null;

    if (myRequest) {
        myRequest.onreadystatechange = processRequestChange;
        myRequest.open("GET", language+".txt", true);
        myRequest.send(null);
    }
    else{
        alert("Nettleseren henger ikke med");
    }
}

function processRequestChange()
{
    // if the request is complete and successfull
    if (myRequest.readyState == 4) {
        if ((myRequest.status == 200) || (myRequest.status == 304)) {
            elt=document.getElementById('show');
            elt.innerHTML=myRequest.responseText;
            }
        else
            alert("Problem med tilgang til data:\n" + myRequest.statusText);
    }
}
```

DEMO: http://www.ia.hiof.no/~borres/ml/picasso/picasso_ajax.html

Referanser og eksempler:

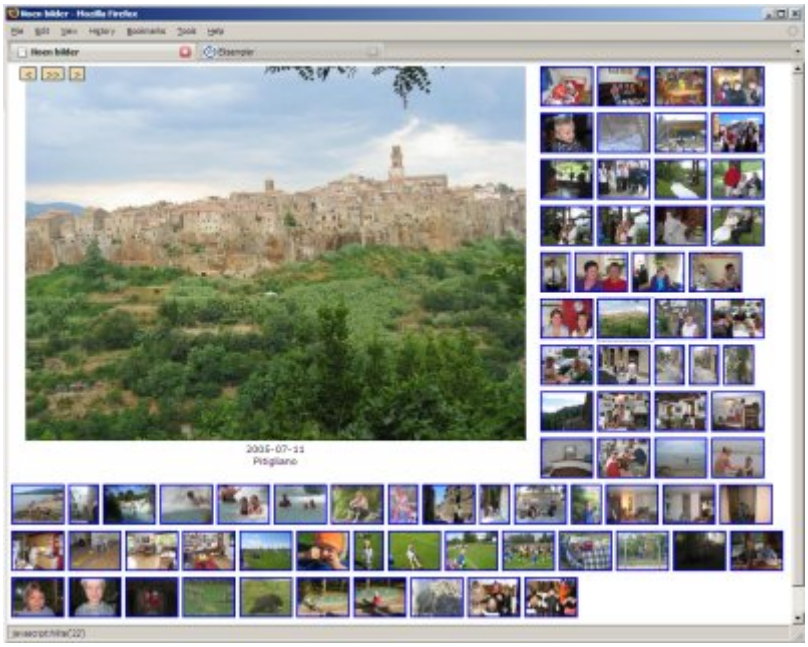
AltaVista BabelFish: world.altavista.com/

Bildegalleri

En enkel ordning av bilder for framvisning

Hva:

Et enkelt bildegalleri



Utgangspunktet er at vi har en katalog med bilder av varierende størrelse. Vi antar for enkelhets skyld at det er JPG-bilder. Vi skal lage en halvautomatikk som lar oss vise et utdrag av bildene i et enkelt galleri, med thumbnails.

Vi ønsker følgende funksjonalitet:

- Vi skal se en full versjon av den thumbnailen vi klikker på
- Vi skal kunne vandre framover og bakover
- Vi skal kunne ha et automatisk "slideshow"

Vi skal lage løsningen på følgende måte:

1. Vi lager en XML-fil som inneholder en kopling mellom bildenavn, dato når bildet er tatt(hvis det er et fotografi) og en kort beskrivelse, **data.xml**
2. Vi lager en master for en framvisningsside, **master.html**
3. Vi lager en Pythonmodul, **makegalleri.py**, som leser data.xml, master.html og produserer **index.html** som er den siden som skal vise fram galleriet.
4. Vi skriver en javascript-fil, **galleri.js**, som inkluderes i index.html og tar seg av dynamikken på framvisningssiden,
5. Vi skriver css-fil, **galleri.css**, som justerer layouten for oss.

Datafil

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bilder>
    <front>
    <![CDATA[
       <h1>Noen bilder</h1>
       <p>Her finner du  litt diverse bilder
       </p>
    ]]>
    </front>
    <bilde>
        <imagename>bilde1</imagename>
        <dato>2007-03-30</dato>
        <text>Pussig fyr</text>
    </bilde>
        ...
</bilder>
```

Master for indexfil

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

<head>
    <link href="galleri.css" rel="STYLESHEET" />
    <title>Noen bilder</title>
    <script type="text/javascript" src="galleri.js"> </script>
</head>

<body onload="init()">

  <!-- navigator -->
  <div  class="printhide" style="position:absolute;left:17px;top:7px">
      <span class="mybutton" onclick="previous()">&lt;</span>
      <span class="mybutton" id="autobutton" onclick="toggle()">&gt;&gt;</span>
      <span class="mybutton"  onclick="next()">&gt;</span>
  </div>

<div style="float:left">
  <div <!-- STYLE FROM PYCODE --> id="hilitedimg">

  </div>
</div>

<div id="content" class="printhide">
            <!-- CONTENT FROM PYCODE -->
</div>

<div style="clear:left;text-align:center">
    <hr/>
    <p>
        En enkel demonstrasjon.
        Skaler vinduet så ser du at
        de små bildene flyter.
        Avspillingskontrollen burde kanskje flyttes ?
    </p>
    <p>
        <a href="javascript:history.back()">Tilbake</a>
    </p>
</div>

</body>
</html>
```

Vi har satt inn to kommentarer som skal erstattes med materiale som produseres i Pythonmodulen.

Pythonmodul

Merk at modulen bruker Python Imaging Library (PIL) til å skalere ned bildet til en akseptabel størrelse og til å lage thumbnails.

```
import xml.dom.minidom
import Image
import os
"""
See: http://www.ia.hiof.no/~borres/ml/galleri/p-galleri.html
"""



thumbFragment="""
<span>
<a href="javascript:hilite('%s')" title="Vis dette">
<img alt="?" src="images/%s" />
</a>
<div style="display:none" id="thumb%s">
  <div><img src="images/%s.jpg" alt="-"/></div>
  <div ><!-- datoen -->
     %s
  </div>
  <div><!-- teksten -->
    %s
  </div>
</div>
</span>
"""

#------------------------
# Read / write text files
def getTextFile(filename):
    try:
        file=open(filename,'r')
        res=file.read()
        file.close()
        return res
    except:
        print 'Trouble reading: '+filename
        return None

def storeTextFile(filename,txt):
    try:
        file=open(filename,'w')
        file.write(txt)
        file.close()
    except:
        print 'Trouble writing to: '+filename

#-------------------------------------------
# extract text from a node
def getText(nodelist):
    rc = ''
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE or \
           node.nodeType == node.CDATA_SECTION_NODE:
            t=node.data.encode('ISO-8859-1')
            rc += t
    return rc

#-------------------------------------------
# produce a thumbnail
def makeThumb(infile,outfile):
    global maxWidth
    global maxHeight
    im = Image.open(infile)
    w=im.size[0]
```

```
    if w > maxWidth:
        maxWidth=w
    h=im.size[1]
    if h > maxHeight:
        maxHeight=h
    tsize=T_SIZE,T_SIZE
    if w > h:
        tsize= int(round(1.0*w/h*T_SIZE)),int(round(1.0*w/h*T_SIZE))
    im.thumbnail(tsize,Image.ANTIALIAS)
    im.save(outfile,"JPEG")

#-----------------------------------------------
# resize an image, down to max height and width
def resizeImage(imfile,maxW,maxH):
    im = Image.open(imfile)
    w=im.size[0]
    h=im.size[1]
    if w <= maxW and h <= maxH:
        return
    if w > maxW:
        h=int(round((1.0*h/w)*maxW))
        w=maxW
    if h > maxH:
        w=int(round((1.0*w/h)*maxH))
        h=maxH
    isize=w,h
    im=im.resize(isize,Image.ANTIALIAS)
    im.save(imfile,"JPEG")


#-----------------------------------------------
# build DOM tree from XML-file
def buildDom(filename):
    T=getTextFile(filename)
    if T== None:
        return None
    try:
        return xml.dom.minidom.parseString(T)
    except:
        print 'Error building dom '
        return None

#-----------------------------------------------
# Find all image elements and populate
# a spanFragment list
def buildPage(dom,catalog):
    global masterPage
    res=''
    # pick up frontpage
    front=''
    try:
        front=getText(dom.getElementsByTagName('front')[0].childNodes)
    except:
        front='<p>Bilder</p>'
    res="""
    <div style="display:none" id="thumb0">
    %s
    </div>
    """%front

    # pick up imagedescriptions
    bildeliste=dom.getElementsByTagName('bilde')
    ix=0
    for bilde in bildeliste:
```

```
        ix+=1
        # pick up imagename (106_0629), dato(2006-04-14)
        # and txt(dette er en tekst)
        try:
            imx=bilde.getElementsByTagName('imagename')[0]
            imagename=getText(imx.childNodes)
            if imagename.endswith('.jpg'):
                imagename=imagename.replace('.jpg','')
            dato=getText(bilde.getElementsByTagName('dato')[0].childNodes)
            txt=getText(bilde.getElementsByTagName('text')[0].childNodes)
            imagefile=catalog+'images\\'+imagename+'.jpg'
            thumbfile=catalog+'images\\t_'+imagename+'.jpg'
            resizeImage(imagefile,I_WIDTH,I_HEIGTH)
            makeThumb(imagefile,thumbfile)
            res+='\n'+thumbFragment%\
            (str(ix),'t_'+imagename+'.jpg',str(ix),imagename,dato,txt)
        except:
            print 'something went wrong,bilde nr: '+str(ix)
    style='style="text-align:center;width:%spx;height:%spx"'%\
          (str(maxWidth+5),str(maxHeight+40))
    masterPage=masterPage.replace('<!-- STYLE FROM PYCODE -->',style)
    masterPage=masterPage.replace('<!-- CONTENT FROM PYCODE -->',res)
    return masterPage

#-------------
catalog='f:\\html\\ml\\galleri\\'
#-------------------------------------------
# max size for images to show
I_WIDTH=700
I_HEIGTH=500
# will be set as max image dimensions found
maxWidth=20
maxHeight=20
# max heigth for thumbnail
T_SIZE=50

masterPage=getTextFile(catalog+'master.html')
if masterPage==None:
    exit(0)
dom=buildDom(catalog+'data.xml')
if dom != None:
    res=buildPage(dom,catalog)
    try:
        outfile=open(catalog+'index.html','w')
        outfile.write(res)
        outfile.close()
    except:
        print 'Error writing file ',filename
```

Javascript

```
var currentNr=0;
var maxNr=1;
var stopped=true;

function init()
{
  var nr=0;
  while (document.getElementById('thumb'+nr))
   nr+=1
  maxNr=nr-1;
  hilite('0');
}

function hilite(nr)
{
  var content=document.getElementById('thumb'+nr).innerHTML;
  document.getElementById('hilitedimg').innerHTML=
    '<div>'+content+'</div>';
  currentNr=parseInt(nr);
}

function previous()
{
    if(currentNr==0)
      hilite(maxNr);
    else
      hilite(currentNr-1);
}

function next()
{
  if(currentNr==maxNr)
    hilite(0);
  else
    hilite(currentNr+1);
}

function toggle()
{
  if(stopped)
  {
    stopped=false;
    setTimeout("show()",3000);
    document.getElementById('autobutton').innerHTML=
      ' || ';
  }
  else
  {
    stopped=true;
    document.getElementById('autobutton').innerHTML=
      ' &gt;&gt; ';
  }
}

function show()
{
  if (!stopped)
  {
    next();
    setTimeout("show()",3000);
  }
}
```

Stilsett

```
@media screen
    {
        body{
                color: black;
                font-family:"Verdana","Geneva","Arial","Times",sans-serif;
                font-size:13px;
                margin:5px;
                padding:0px;
        }
        .text{font-size:14px}
        .dato{color:gray}
        .mybutton{cursor:pointer;color:black;
                  background-color:#EED9B0;
                  border-style:solid;
                  border-width:thin;
                  padding:5px;
                  font-weight:bold;
                  margin-right:2px}
        h2{background-color:#EED9B0;}
    }

    @media print
    {
        .printhide{display:none;font-size:16px;text-align:center}
    }
```

DEMO: http://www.ia.hiof.no/~borres/ml/galleri/index.html

Referanser og eksempler:

Python Imaging Library, PIL: www.pythonware.com/products/pil/

Et komplett eksempel: sample.zip

Slideshow

Hva:

En enkel bildeframviser

Dette eksempelet bygger på noe av det samme resonnementet som i modulen Bildegalleri, men kan leses uavhengig av denne modulen. Vi antar at vi har en katalog med bilder, og vi ønsker å lage en vevside som viser disse bildene i tilfeldig rekkefølge.

Vi skal lage:

- En vevside, **auto.html**
- Et Javascript, **auto.js**
- Et Pythonscript som går på tjeneren, **auto.py**
- Et stilsett, **auto.css**

Vevsiden

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

<head>
    <title>Bilde sekvens</title>
    <link href="auto.css" rel="STYLESHEET" />
    <script type="text/javascript" src="auto.js"> </script>
</head>

<body onload="init()">
<h2 class="printhide">Slideshow</h2>
<div id="hilitedimg" style="height:560px;text-align:center">
    <p>
        <img src="wait.gif" alt="Starter straks"/>
    </p>
</div>

<div class="printhide">
  <p id="toggle" onclick="toggleShow()" class="mybutton">
      Start
  </p>
  <hr/>
  <p>
        <a href="javascript:history.back()">Tilbake</a>
    </p>
</div>

</body>
</html>
```

Javascriptet

Løsningen er basert på at vi laster inn en randomisert liste av bilder. Dette gjøres i JSON-format når siden lastes, funksjonen **init**. Hvordan JSON-formatet brukes går fram av funksjonen: **processJSON** nedenfor.

Denne løsningen innebærer at den tilfeldige rekkefølgen gjentar seg. Det er lett å modifisere Javascriptkoden slik at vi får en ny tilfeldig liste når den første er "oppbrukt".

Funksjonene **toggleShow** og **Show** tar seg av selve bildevisningen. De resterende funksjonene realiserer AJAX.

```
var imList=null;
var currentNr=0;
var stopped=true;
var delay=4000;
var scriptURL='<scriptadressen>';
var imageCatalog='<absolutt adresse på tjeneren>';
var imageURL='<absolutt url ti bildekatalogen>';

function establishRequest()
{
    var theRequest=null;
    if (window.XMLHttpRequest)
        theRequest = new XMLHttpRequest();
    else if (window.ActiveXObject)
    {
        try { theRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch(e) {
            try {
              theRequest= new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch(e) {
              theRequest=null;
            }
        }
    }
    else
        theRequest=null;
    return theRequest;
}

function postRequest(theRequest,params,theURL)
{
    theRequest.onreadystatechange =
        function( ) {
                    processJSON(theRequest);
        };
    theRequest.open("POST", theURL,true);
    theRequest.setRequestHeader("Content-type",
          "application/x-www-form-urlencoded");
    theRequest.setRequestHeader("Content-length", params.length);
    theRequest.setRequestHeader("Connection", "close");
    theRequest.send(params);
}

function processJSON( aRequest)
{
    if (aRequest.readyState == 4) {
        if ((aRequest.status == 200) || (aRequest.status == 304))
        {
            try{
              p = eval("(" +aRequest.responseText+")");
              imList=p.imList;
              toggleShow();
            }
            catch(ex){
              document.getElementById('hilitedimg').innerHTML=
               'Finner ingen bilder';
            }
        }
        else
            alert("Problems accessing data:\n" + aRequest.statusText);

    }
```

```
}

function doRequest(theURL,params)
{
    var myRequest=establishRequest();
    if (myRequest)
    {
        postRequest(myRequest,params,theURL);
    }
    else
      {
        alert('Nettleseren nekter å hente data');
    }
}

  function toggleShow()
  {
       if(stopped)
        {
          stopped=false;
          show()
          document.getElementById('toggle').innerHTML='Stopp';
        }
        else
        {
          stopped=true;
          document.getElementById('toggle').innerHTML='Start';
        }
  }

  function show()
  {
    if (!stopped)
    {
      currentNr+=1;
      if (currentNr >= imList.length)
          currentNr=0
      imsrc=imageURL+imList[currentNr].adr;
      imalt=imList[currentNr].adr;
      imtag='<img src="'+imsrc+'" alt='+imalt+'"/>'
      document.getElementById('hilitedimg').innerHTML=imtag;
      setTimeout("show()",delay);
    }
  }

function init()
{
    params='catalog='+imageCatalog;
    doRequest(scriptURL,params)
}
```

Pythonscriptet

```
#! /usr/bin/python2
"""
Scans a directory and returns a randomized list of images
use see: http://www.ia.hiof.no/~borres/ml/slideshow/p-slideshow.html
"""
import os,random
import cgi
import cgitb; cgitb.enable()
import operator


def buildImageList(catalog):
    imageList=[]
    for root, dirs, files in os.walk(catalog, topdown=False):
        for name in files:
            # only jpg-files
            if not name.endswith('.jpg'):
                continue
            # drop thumbs
            if name.startswith('t_'):
                continue
            imageList.append(name)
    if len(imageList)==0:
        return '{"imList":[{"adr":"Finner ingen bilder i katalogen"}]}'
    random.shuffle(imageList)
    T=''
    for im in imageList:
        T+='{"adr":"%s"},'%im
    return '{"imList":[%s]}'%T[:-1]


form=cgi.FieldStorage()
print 'Content-type: text/plain\n'

#expect to find: catalog

if form.has_key('catalog'):
    catalog=form['catalog'].value
    print buildImageList(catalog)
else:
    print '{"imList":[{"adr":"Finner ingen bilder"}]}'
```

Stilsettet

Stilsettet inneholder mulighet for å skrive ut bildene i ren form, uten menyer og overskrifter.

```
@media screen
{
    body{
            color: black;
            font-family:"Verdana","Geneva","Arial","Times",sans-serif;
            font-size:13px;
            margin:5px;
            text-align:center;
            padding:0px;
    }
    .text{font-size:14px}
    .dato{color:gray}
```

```
    .mybutton{cursor:pointer;color:black;
              background-color:#EED9B0;
              border-style:solid;
              border-width:thin;
              margin-right:2px}
    h2{background-color:#EED9B0;}
}
@media print
{
    .printhide{display:none;font-size:16px;text-align:center}
}
```

DEMO: http://www.ia.hiof.no/~borres/ml/slideshow/auto.html

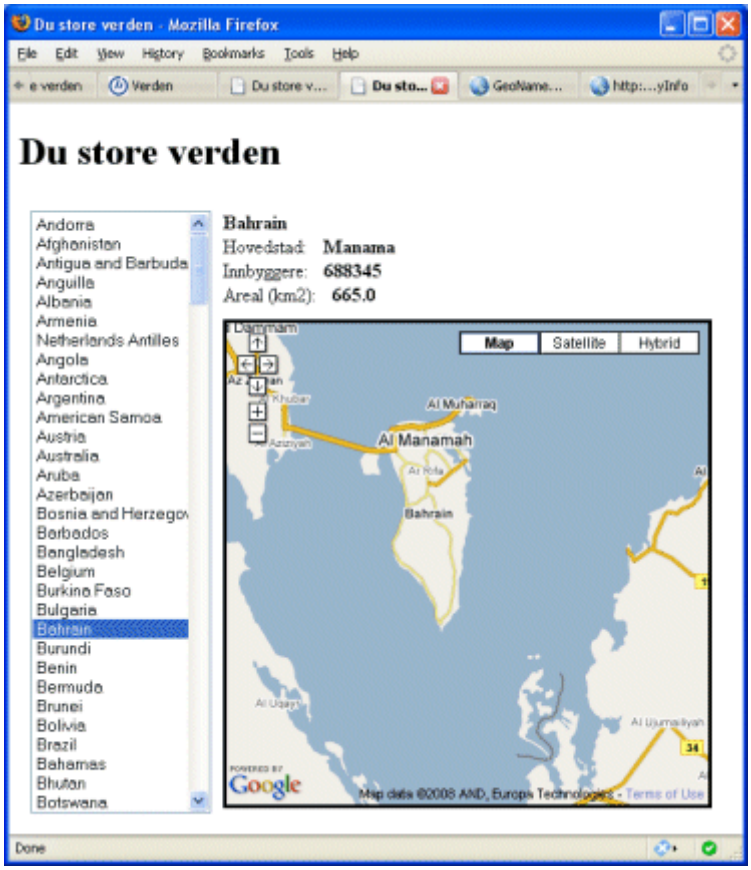
Referanser og eksempler:

- Modulen AJAX
- Modulen JSON

Land i verden

Hva:

Kart (Google Maps) over alle land i verden



I denne modulen skal vi lage en enkel vevside som lar oss finne fram noe informasjon om land i verden. Utgangspunktet er data fra geonames [1] . De XML-dataene vi skal bruke er referert på modulen Noen datasett, og har adresse: http://www.ia.hiof.no/~borres/commondata/geonames/land.xml. XML-fila ser slik ut: land.xml

Vi bruker GoogleMaps [2] . Merk at for å bruke Google Maps må du skaffe deg en ID som gjelder for et domene.

Dette lar seg løse på mange måter. Vi kan lage en helt enkel vevside som oppdateres ettersom vi gjør valg, eller vi kan lage en side som er AJAX-basert, dvs. den henter bare de dataene vi trenger. Vi skal imidlertid gjøre det ennå enklere: Vi lager en HTML-side som er selvforsynt. Det vi si at vi skal kunne velge nye land uten å hente data fra tjeneren.

DEMO: http://www.ia.hiof.no/~borres/cgi-bin/gmaps/verden.py

Det er to komponenter som skal lages:

- Et Pythonscript som henter nødvendige data og apkker dem pent sammen på en HTML-side, som så leveres til klienten
- Et javascript som tar seg av dialogen med brukeren, og som altså er selvforsynt med de data som er pakket på vevsiden

python scriptet

Scriptet er ganske enkelt. Det henter XML-fila, bygger et DOM-tre og lager en listeboks. Denne plasseres inn på et vevsideskjelett som så skrives tilbake. Listeboksen bygges opp slik at den inneholder alle de data vi trenger for å tegne nytt kart. Noen linjer i boksen:

```
<option value="Nassau,301790,13940.0,-78.978897,26.929165,-72.737503,20.915276,Bahamas">
Bahamas
</option>
<option value="Thimphu,2232291,47000.0,88.751938,28.325275,92.115265,26.703049,Bhutan">
Bhutan
</option>
```

Dataene som ligger som value er i tur og orden: capital,population,areaInSqKm,bBoxWest,bBoxNorth,bBoxEast,bBoxSouth,countryName. Dette tilsvarer elementer til hvert country-element i vår kildefil, land.xml. Box-dataene er hjørner i det rektangelet som omgir landet.

Pythonscriptete er i sin helhet slik:

Javascriptet

Javascriptet leser de dataene som er pakket ned i listeboksen, plasserer dem i dertil angitte elementer, og bruker spesielt box-datene til å sette opp data for å tegne kart.

Referanser og eksempler:

- [1] Geonames:
  geografiske data
  www.geonames.org/
- [2] Google Maps:
  Map API
  code.google.com/apis/maps/index.html

Koden er sitert i teksten