LAB3: Hendelser (events)

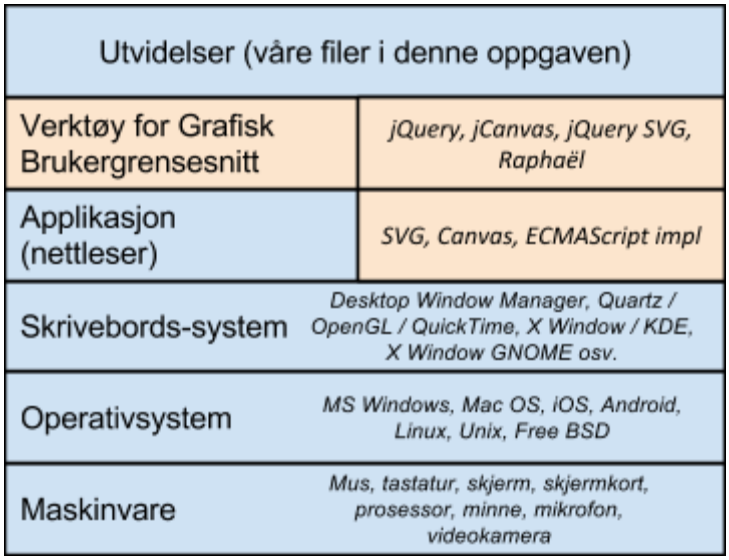
PUBLISERT: 27.09.2013
FRIST: Tirsdag 08.10.2013 kl. 23:59 i Class Fronter
TILHØRENDE MATERIAL: mal for skriving av koden er publisert i Class Fronter

Innledning

I denne oppgaven skal dere se på hvordan man kan behandle hendelser som skapes av brukeren mens den bruker nettleserens grafiske brukergrensesnitt. De vanlige interaksjonsmetodene per i dag er mus (og eventuelt styrespaken e.l.), tastatur (diverse knapper med symboler på) og berøring av skjermen (eng. touch screen; samme prinsipp som knapper med noen flere muligheter for å dra og manipulere grafiske elementer med fingrene eller annet hjelpemiddel). Lyd og gester (eventuelt mimikk) vil være en del av fremtidens metoder, kan det se ut som.
Dere skal også leke med noen verktøy for å tegne grafikk vha. kode, på litt lavere nivå enn vanlige med vanlige HTML/CSS elementer[1].

Dere skal se på to av de per i dag vanligste interaksjonsmetodene, - mus og tastatur. Men først en liten fordypning i hvordan det grafiske grensesnittet EGENTLIG dannes på skjermen og hvilke moduler eller biblioteker som brukes per i dag for å manipulere dette grensesnittet. Dagens applikasjoner er vanligvis lagbasert. Dvs. at det er et lag som er over et annet lag og det er implementert et kommunikasjonsgrensesnitt mellom disse lagene. Et brukergrensesnitt er ofte et lag i seg selv samt den består av flere lag som man ofte kaller for *layers*. Man implementerer skjeldent brukergrensesnitt helt fra bunnen av, dvs. mot maskinvare. Det er vanlig praksis å bruke eksisterende og godt gjennomtestede biblioteker (eller moduler) for å lage grensesnittet mot brukeren. Vi bygger med andre ord på eksisterende operativsystem og vindusadministrasjonssystem (og mye mer, se illustrasjon) ved hjelp av verktøy for presentasjon av det grafiske brukergrensesnittet.

[1] et eksempel på vanlig HTML / CSS “tegning” er div-elementet med dimensjoner, som høyde og bredde, posisjon i sitt foreldrevindu og en stil som gir den farge og eventuelt mønster



Den verktøykasse som vi bruker i dette kurset for å utvikle vår egen grafiske brukergrensesnitt som kan da også brukes i WWW, inneholder mange JavaScript moduler for å manipulere grensesnittet. Man kan gjøre mye med den tradisjonelle implementasjonen av DOM, HTML og CSS, men ønsker man mer "rikt" og dynamisk grensesnitt, er det vanskelig å klare seg uten JavaScript. JavaScript kan også brukes for å redusere trafikken mellom nettleser og webtjener. JavaScript er et såkalt objektorientert programmeringsspåk (eksempler på andre slike er Java, C++, Python, C#). Man kan manipulere grensesnittet med kall til forskjellige metoder[2] og attributter i JavaScript klasser. Siden klassene kan arve egenskapene fra hverandre, er det enkelt å bygge nye klasser på toppen av andre, allerede eksisterende klasser. Et eksempel er jQuery biblioteket, som bygger på en ECMAScript implementasjon i nettlesere, som kalles for JavaScript motor (eng. engine). jCanvas biblioteket, som lar oss tegne primitive grafiske elementer, er igjen bygd på jQuery.

Den grafiske presentasjonen genereres hovedsakelig basert på brukerens aktivitet og forskjellige applikasjoner. Det er tre forskjellige måter å lage denne presentasjonen på:

- ved hjelp av komponenter (som vinduer, for eksempel), som er arrangert i et hierarkisk system (eksempel er DOM, som er implementert i nettlesere, se en kort innføring her http://www.webreference.com/programming/javascript/ppk1/2.html)
- ved hjelp av streker; tegningen / presentasjonen lages vha. av høyere-nivå elementer som linjer, firkanter og buer (Canvas i HTML5 har abstrakte metoder for slike primitive elementer, som for eksempel drawLine; se under en jQuery kamuflert metode for drawLine metoden i JavaScript modellen; "stroke" er "strek" på norsk og gir dere en mening av denne metoden; SVG i HTML5 implementasjon har også en "strekmodell")

[2] Hvis man har definert et objekt my_canvas av klassen Canvas, kan man få tak i denne objekten for å kunne tegne på den med metoden getContext("2d"), og denne metoden kan kalles slik: my_canvas.getContext("2d");

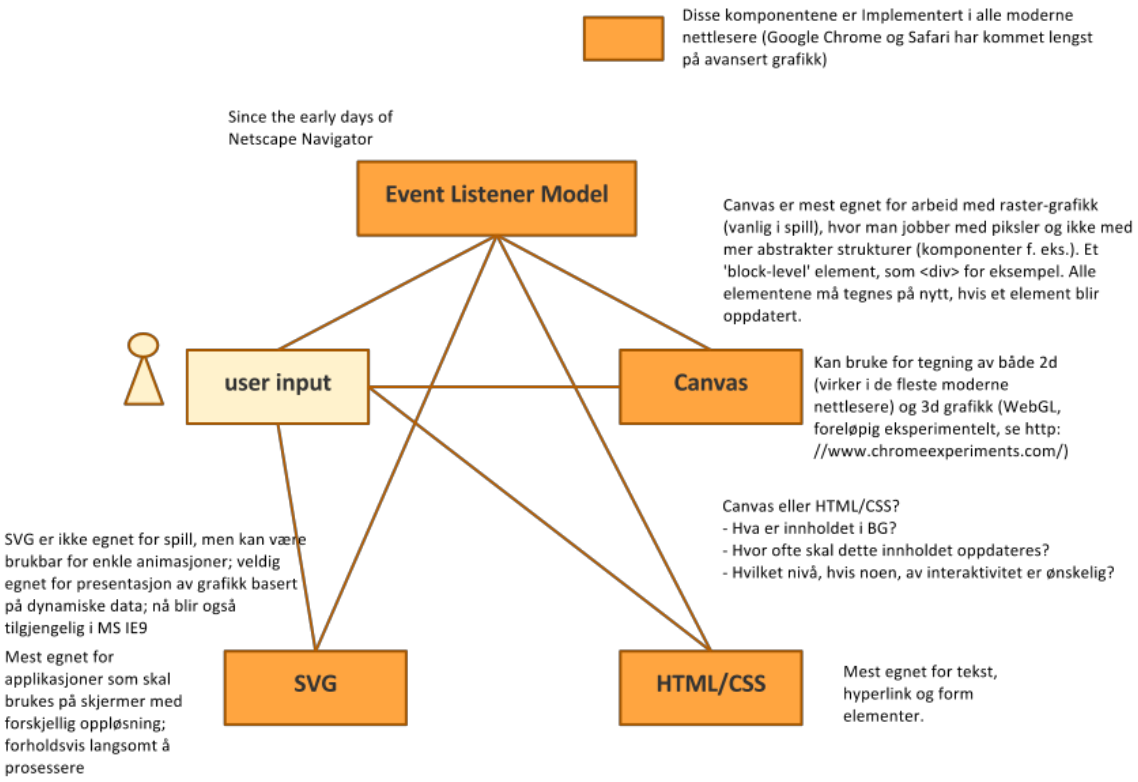
```
$("canvas").drawLine({
  strokeStyle: "#000",
  strokeWidth: 10,
  x1: 50, y1: 50,
  x2: 100, y2: 150,
  x3: 200, y3: 100,
  x4: 150, y4: 200
});
```

- ved hjelp av piksler; her er hele skjermen (eller deler av skjermen hvis man blander presentasjonsmåtene) representert som en tabell av piksler hvor størrelsen på pikselen har sammenheng med oppløsning, dvs. hvor finkornet er skjermen; man kaller det også for raster; DirectX i MS Windows systemer er en måte å manipulere piksler direkte (ikke gjennom flere lag av strek- og eller komponentpakker) for å få en bedre ytelse; Canvas i HTML er også best egnet for raske realtidsapplikasjoner hvor det er nødvendig å oppdatere skjermen fort (som i spill, for eksempel); WebGL er det eksperimentelle grafikkmotoren som tas nå gradvis i bruk med HTML5 og vil gi muligheter til å manipulere 3D grafikk direkte gjennom canvas objekter i HTML, - kult! (se http://www.chromeexperiments.com/)

Siden alle tre modeller (komponent, strek og piksler) er presentert i de fleste grafiske verktøy, spørsmålet er, når, under implementasjon av deres brukergrensesnitt, kunne det være nødvenig å bruke lavt-nivå presentasjonsmodeller som strek- og pikslermodellen?

En utfordrende tilfelle kunne være hvis du skulle gi brukeren på en berøringsskjerm mulighet til å skrive inn signatur som en del av en visning i en nettleser. Da hadde det ikke holdt med kun vanlige HTML / CSS komponenter som tekstfelt, for eksempel, siden man ikke kan tegne i en teksfelt direkte. Du kunne selvfølgelig gi brukeren mulighet til å laste opp et bilde med hans signatur, men da hadde interaktivitetsnivå blitt mye lavere og brukbarhet dårligere. Så det kan hende at din kunnskap om Canvas (og eventuelt SVG) kan være nyttig.

Så hvordan henger brukergenererte hendelser (vha. av mus eller tastatur f. eks.) sammen med Canvas?



Alle hendelser som genereres i et nettleser-grensesnitt fanges opp av “Event Listener Model”, som er tilgjengelig gjennom JavaScript. For å observere et objekt med en spesifisert Id i DOM treet, kan man bruke JavaScript direkte som i dette eksemplet:

```
var my_canvas_element = document.getElementById(“my_canvas”);
if(my_canvas_element.addEventListener)
      my_canvas_element.addEventListener(‘keyup’,eventkeyup,false);
…
function eventkeyup(e) { /*hva skal skje*/ }
```

- først finner man objekt (eller node) vha. Id i DOM treet,
- så sjekker man om dette objektet (noden) kan “observere” (høre på) hendelser
- og til slutt legger man til en “observatør” (listener) med en spesifikasjon av hvilken hendelse den skal observere (‘keyup’) og hva og hvordan den skal reagere når en hendelse inntreffer, - eventkeyup er en funksjon som skal utføres når hendelsen inntreffer og false er en beskjed om hvordan man skal håndtere situasjoner hvor flere objekter i DOM treet er observatører av samme hendelse)

Hovedproblemet med å bruke "rå" JavaScript er at det er fortsett forskjeller på hvordan de forskjellige nettlesere implementerer "Event Listener Model", slik at man må eventuelt skrive egen kode for flere populære nettlesere.

I denne laben skal dere bruke jQuery biblioteket (http://jquery.org). Se på jQuery som et lag over den ordinære (i nettleseren) JavaScript implementasjonen. Hvorfor skulle vi ønsker et ekstra lag? jQuery pakken (modulen, biblioteket) gir oss en viss garanti at koden vi skriver skal virke i de mest populære nettlesere. Derfor, først og fremst. Det finnes også mange ekstra biblioteker som man kan vie sammen med jQuery og da beholde et helhetlig grensesnitt mot DOM treet, ordinære JavaScript funksjoner og eventuelt primitive grafikk metoder (som de implementert i Canvas).

Koden som tilsvarer den "rå" JavaScript koden ovenfor, kan i jQuery skrives på følgende måte:

```
$(function() {
      $("#my_canvas").on('keyup', function(e) {/*hva skal skje*/});
});
```

Se lab3_v1.html for et konkret eksempel.

For å gjennomføre denne laboppgaven trenger du følgende:

- tilgang til Internett og WWW for å finne ressurser
- en nettleser som er implementert etter XHTML 1.1 standard og støtter JavaScript og HTML5 (Google Chrome anbefales og kan lastes ned her https://www.google.com/intl/no/chrome/browser/?hl=no)
- spesifikk pakke med alle biblioteker for LAB3 (finnes i Class Fronter)

Oppgave

Oppgave består av 3 deloppgaver, 0, 1 og 2. <beskriv oppgavene kort her>

Hendelser og annet "leketøy"

I denne oppgaven går vi litt bort fra det grensesnittet som vi jobbet med i de forrige laboppgavene. Vi bruker enkle html sider hvor vi ikke tenker på brukbarhet, men som skal brukes kun for å teste ut enkelte funksjoner som finnes i alle populære nettlesere og som kan hjelpe oss å lage et "rikere" brukergrensesnitt for mange type applikasjoner.

0 [Formål: å bli kjent med Event Listener Modellen og jQuery samt å prøve i praksis å fange opp brukeraktivitet på tastatur]

I versjon 1, `lab3_v1.html`, er det implementert en funksjon basert på jQuery, som lar oss å

fange opp trykk på enkelte knapper på tastaturet. Det kan brukes hvis vi ønsker å gi brukeren alternativet å jobbe i grensesnittet kun vha. tastaturet (som kan være en av brukbarhetskravene).
Basert på eksemplet og tidligere laboppgaver skal du implementere følgende:

a) respons på piltastene opp / ned på samme måte som respons for piltastene høyre / venstre (du kan finne tallkodene til tastene vha. denne JavaScript-baserte siden http://www.cambiaresearch.com/articles/15/javascript-char-codes-key-codes)

b) funksjonalitet som gjør at man kan også se responsen direkte på siden (tips: bruk en `<p>` tagg og sett en id **keypressed**, ; bruk id for å skrive til innerHTML for denne elementnoden, noe som i jQuery vil se slik ut

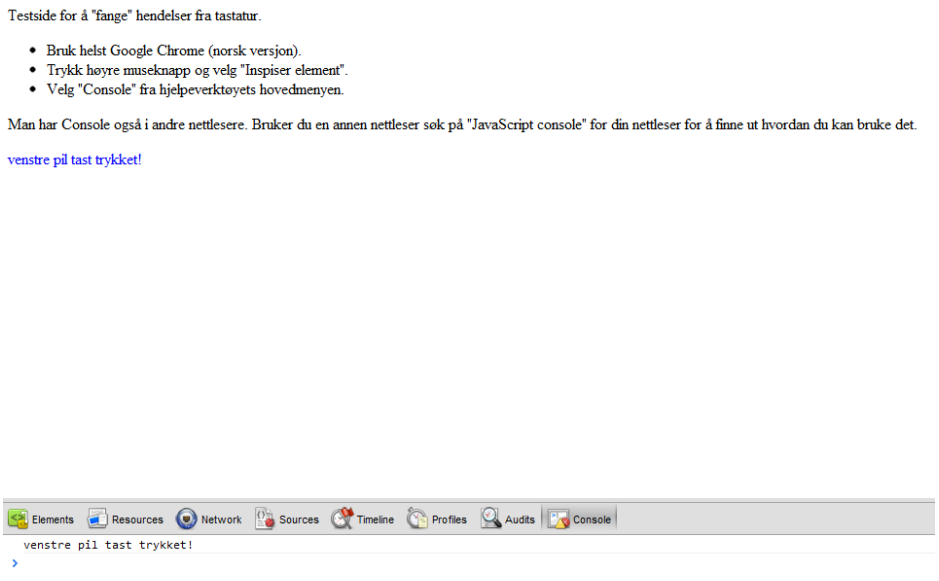
```
$("#keypressed").html("Tekst som sier hvilken tast er trykket");
```

Det er den samme teksten som skal skrives ut til både console.log og til siden. Hvordan kan du skrive kode slik at du trenger å spesifisere teksten kun en gang? (tips definer en variabel med selvvalgt navn og `var` foran og gi denne variablen verdien i form av tekst)

c) en stil som kun setter tekstfargen til en farge, blå (.blue), for eksempel (legg dette inn i stilfilen med navn lab3.css); bruk jQuery for å sette farge på teksten som du skriver ut i punkt b) slikt:

```
$("#keypressed").addClass("blue");
```

Resultatet i Google Chrome skal se omtrent slikt ut:



1 [Formål: bli kjent med jQuery og prøve i praksis å fange opp musaktivitet og muskoordinater]

I versjon 2, `lab3_v2.html` skal det fanges opp musaktivitet. I eksemplet er det implementert et **canvas** objekt (node) med **id** my_canvas og jQuery funksjoner for å observere musetrykk og beregne koordinater til musen innenfor Canvas objektet.

Implementer følgende:

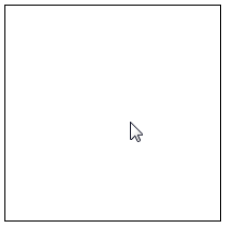
a) til console.log skal du skrive ut koordinatene til museklikk på følgende format "(x,y)=(x-verdi,y-verdi)", hvor x-verdi og y-verdi er de kalkulerte verdiene

b) til skjermen skal du skrive ut to linjer med tekst:

Avstand fra den venstre kanten av Canvas er <den kalkulerte horisontale verdien>
Avstand fra den øverste kanten av Canvas er <den kalkulerte vertikale verdien>

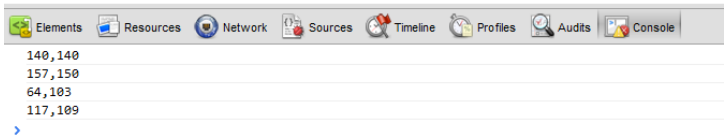
Siden det er kun de kalkulerte verdiene som endres her, bruk elementnoden med tilsvarende id for å skrive til vha. jQuery funksjoner (som i deloppgave 0). Begge beregnede verdier skal vises i fet skrift (bold), den horisontale skal være blå (blue) og den vertikale lilla (purple). Alle stilene skal settes i jQuery funksjonen.

Resultatet i Google Chrome skal se omtrent slikt ut (etter a) og b)):



Avstand fra den venstre kanten av Canvas er 117

Avstand fra den øverste kanten av Canvas er 109



2 [Formål: bli kjent med jCanvas og lære tegne tekst til et Canvas objekt]

I versjon 3, `lab3_v3.html` skal du implementere en ekstra funksjon i forhold til deloppgaven 1. Koordinatene til musetrykk skal i tillegg til console.log og innerHTML også skrives ut til Canvas objekt. jCanvas biblioteket (bygger på jQuery) skal brukes for oppgaven (se mer om jCanvas her http://calebevans.me/projects/jcanvas/index.php).

Når brukeren trykker på et punkt innenfor Canvas, skal koordinatene **mouse.x** og **mouse.y** til dette punktet skrives i Canvas der hvor brukeren trykket museknappen.

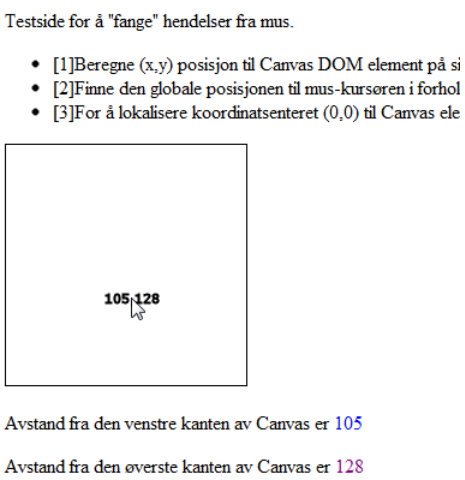
Tips:

- bygg videre på filene fra forrige deloppgaven;
- se på hvordan man kan tegne en tekst til Canvas vha. jCanvas metoder http://calebevans.me/projects/jcanvas/docs.php?p=text og prøv å gjøre det selv i din egen fil
- jeg har lagt inn metodehodet til drawText og også en metode clearCanvas som utføres før drawText, og som gjør at den forrige teksten slettes før en ny tegnes (ellers hadde Canvas blitt fort fullt opp med tilfeldig plassert tekst); legg også merke at metodene hører til **canvas** objektet i jQuery, men når de skrives etter hverandre, så kan man la være å skrive **canvas.** flere ganger

Her er kravene til tegningen av tekst i Canvas:

fillStyle: "#000", // fargen til konturene til tekstfigurene
strokeStyle: "#000", // fargen til utfylling av konturene
strokeWidth: 1, // hvor fet blir teksten
x: **mouse.x**, y: **mouse.y**, // koordinatene til museklikk; rundt dette punktet skal tekst tegnes
font: "8pt Verdana, sans-serif", // selvforklarende ...
text: **mouse.x** + "," + **mouse.y** // hva som skal skrives ut, selve teksten

Her er forventet resultat i Google Chrome:



Spørsmål

Spørsmålene besvares i README-filen i GitHub.

---

0) Hva er fordelen med et eksisterende JavaScript bibliotek som jQuery i motsetning til å skrive en "rå" JavaScript for dynamisk endring av HTML sider?

1) Les denne artikkelen http://www.ibm.com/developerworks/web/library/wa-htmlmark/index.html og svar på følgende spørsmål: Hva er sterke sider til HTML/CSS og hva er sterke sider til Canvas? I hvilke situasjoner er det meningsfult å bruke Canvas?

2) Hvilken grafisk presentasjonsmåte (se Innledning til denne oppgaven) hadde du sagt at HTML5 Canvas representerer? Forklar.

3) EKSTRA (bare for de spesielt interesserte): en mangel på resultatet fra deloppgave 2 er at hvis man trykker musen nærme kanten, kan man ikke se hele teksten siden den klippes vekk hvis den går utover Canvas grenser; hvordan kunne man "fikse" på det vha. metodene fra jCanvas? Lever i lab3_v4.html.

X) Hvis du fikk hjelp til å gjennomføre denne oppgaven av noen, samarbeidet med noen, oppgi navn på disse personene. Oppgi også ca. tid det tok deg for å gjennomføre denne oppgaven (f. eks. 2t 30m).

Hva skal leveres og hvor?

Alle filene leveres i GitHub. Unngå å legge inn jQuery og jCanvas biblioteker i GitHub, gjør kun commit av deres egen kode.

REFERANSER

- Bokmålsordboka. Sist hentet 4.9.2012 fra http://www.nob-ordbok.uio.no/perl/ordbok.cgi?OPP=tagg&bokmaal=+&ordbok=bokmaal
- Bos, B. (2011). CSS Tutorial: Starting with HTML + CSS. Sist hentet 12.09.2012 fra http://www.w3.org/Style/Examples/011/firstcss
- Browser Shots. Sist hentet 27.08.2012 fra http://browsershots.org/
- Cascading Style Sheets. Sist hentet 28.08.2012 fra http://no.wikipedia.org/wiki/Cascading_Style_Sheets
- ColorPicker 3.1. Pagetutor.com. Sist hentet 28.08.2012 fra http://www.pagetutor.com/colorpicker/index.html
- Color Worqx. www.worqx.com. Sist hentet 28.08.2012 fra http://www.worqx.com/color/index.htm
- HTML. no.wikipedia.org. Sist hentet 28.08.2012 fra http://no.wikipedia.org/wiki/HTML
- HTML5. wikipedia.org. Sist hentet 11.09.2012 fra http://en.wikipedia.org/wiki/HTML5
- Introduction to CSS 2.1. W3C Recommendation. Sist hentet 28.08.2012 fra http://www.w3.org/TR/CSS2/intro.html
- JavaScript Tutorial. (n.d.). Sist hentet 11.09.2012 fra http://www.w3schools.com/js/default.asp
- jCanvas. (n.d.). Sist hentet 19.09.2012 fra http://calebevans.me/projects/jcanvas/index.php
- jQuery. (n.d.). Sist hentet 08.10.2012 fra http://jqyery.org
- jQuery User Interface. (n.d.). Sist hentet 08.10.2012 fra http://jqueryui.com
- Kärkkäinen, S., (n.d.). Interactive jQuery selector tester. Retrieved September 10, 2012, from http://sk.kapsi.fi/interactive-jquery-tester.html
- Notepad++. Sist hentet 27.08.2012 fra http://notepad-plus-plus.org/
- The World Wide Web Consortium. Sist hentet 27.08.2012 fra http://www.w3.org/

- XHTML Basic 1.1 Cheat Sheet. Sist hentet 28.08.2012 fra http://www.w3.org/2007/07/xhtml-basic-ref.html
- W3C Markup Validation Service. Sist hentet 28.08.2012 fra http://validator.w3.org/#validate_by_uri
- W3Schools.com. Sist hentet 27.08.2012 fra http://www.w3schools.com
- WAI-ARIA. ACCESSIBLE RICH INTERNET APPLICATIONS (WAI-ARIA) CURRENT STATUS. Sist hentet 4.9.2012 fra http://www.w3.org/standards/techs/aria#w3c_all