

Introduksjon til HTML, CSS og Javascript

Å se på web-sider har de fleste gjort. For eksempel kan du lese VG på nett. Det gjorde jeg 17. april 2018. Da så det ut som til høyre.

Programmet jeg bruker kalles en nettleser. Jeg bruker Google Chrome, men andre eksempler på nettlesere er Firefox, Internet Explorer og Safari.

Du kan bruke hvilken du vil, men aller enklest blir det om du bruker samme nettleser som meg. Den er gratis, og du kan laste den ned fra www.google.com/chrome.



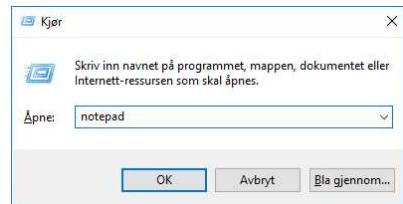
Min første web-side - ved hjelp av HTML

Når du skriver inn en adresse, som for eksempel "www.vg.no", henter nettleseren web-siden fra datamaskinen (serveren) til VG. Men du kan også åpne en web-side fra en fil på din egen PC, og det er det vi skal gjøre når du nå skal få lage din første web-side.

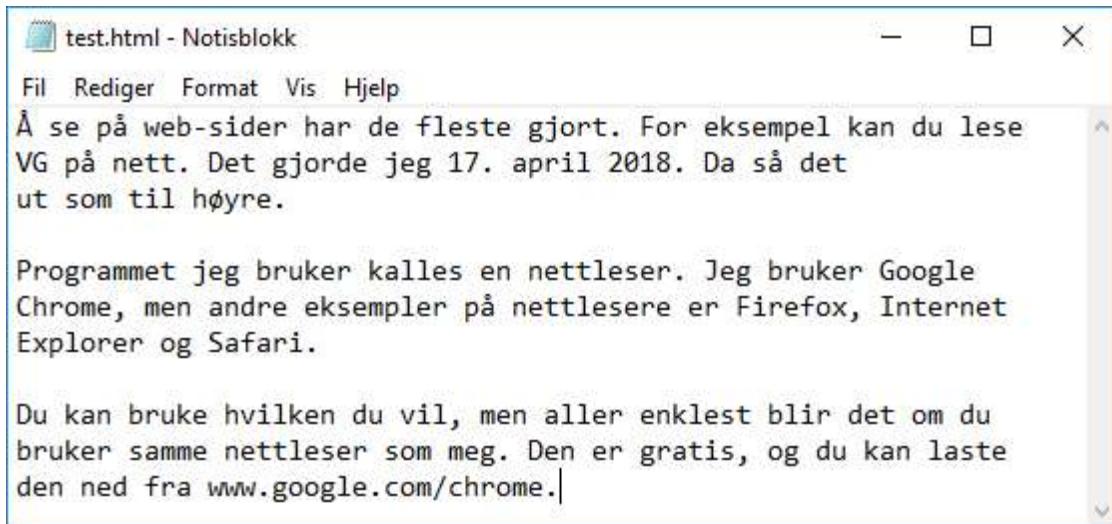
Til å lage denne filen, trenger du en teksteditor. Ikke en tekstbehandler, som word, men et enda enklere program som lar deg redigere rene tekstfiler uten formatering. I Windows er det innebygget et program som heter "Notisblokk" om du har norsk windows og "Notepad" om du har engelsk. Du kan finne det på start-menyen, men om du ikke finner det, kan du holde nede Windows-tasten og så trykke på R. Windows-tasten er vist på bildet til høyre.



Da vil du få opp en dialogboks med tittelen "Kjør" eller "Run", som vist til høyre.

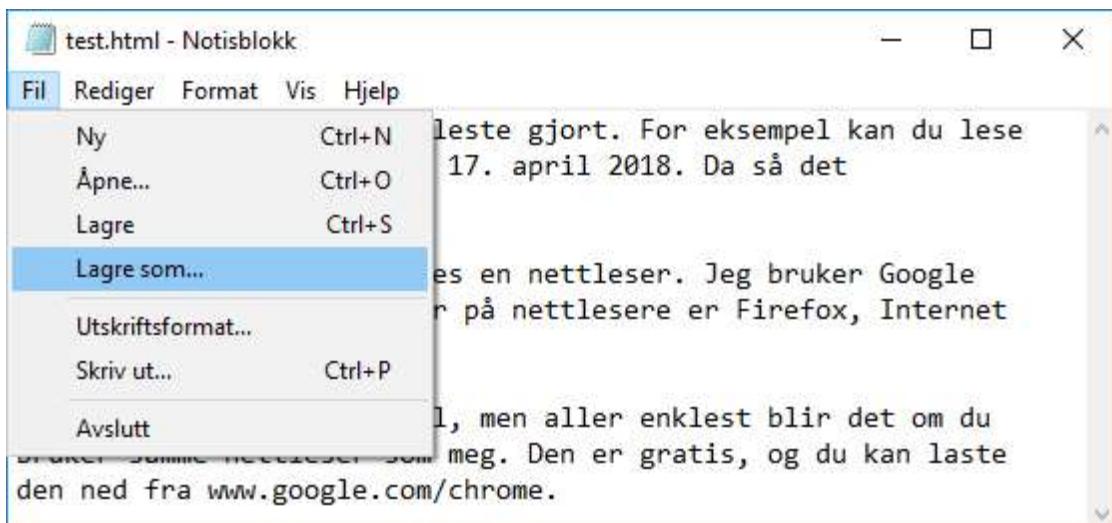


I denne skal du skrive "notepad" og trykke Enter-tasten på tastaturet, eller OK-knappen i dialogboksen. Da starter Notepad. Skriv eller kopier litt tekst i vinduet. Jeg kopierte inn litt tekst fra denne boken, og under vises hvordan det da ser ut.



Dette er din første web-side. For å se den i nettleseren, må vi først lagre den som en fil.

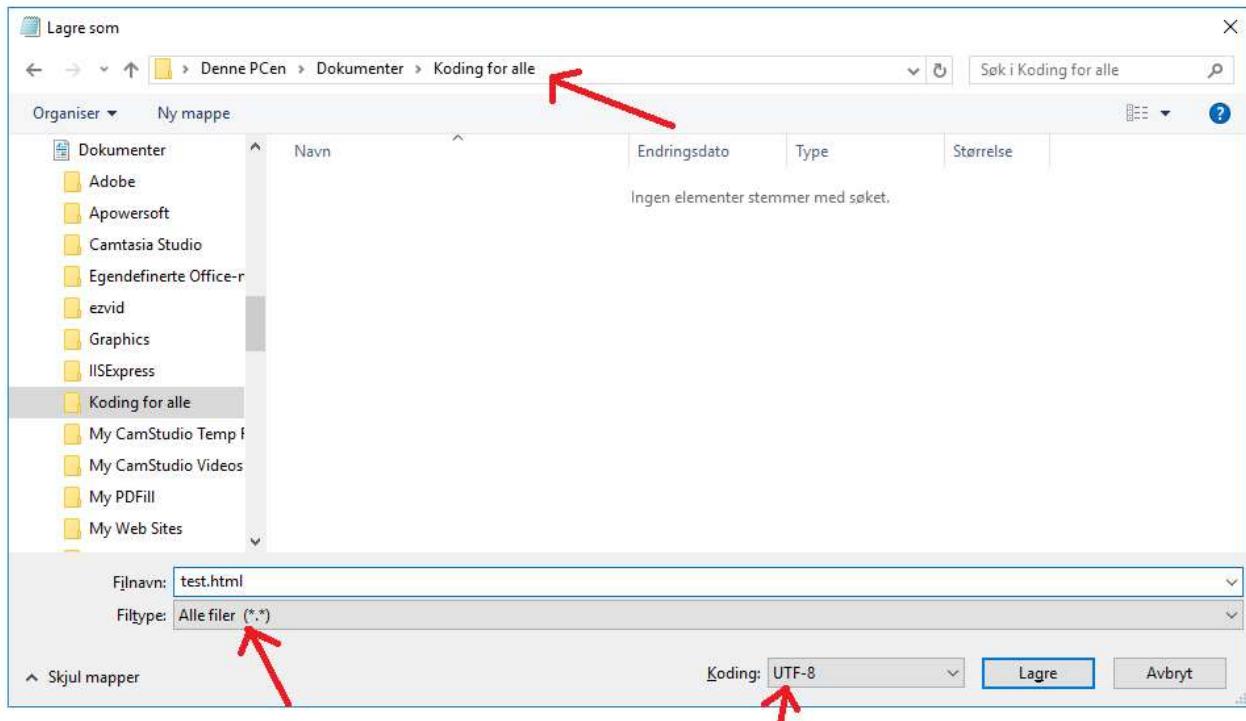
Åpne Fil-menyen og velg så "Lagre som..." som vist på bildet under.



Velg så et filnavn, for eksempel "test.html". Filnavnet må slutte på ".html". I tillegg må du velge:

- Filtype: Alle filer
- Koding: UTF-8
- en mappe som du husker

Alle de tre tingene markert med rød pil i bildet under:



Til slutt trykker du på knappen “Lagre” nederst, nest lengst til høyre. Neste skritt er å åpne denne filen i Google Chrome eller en annen nettleser. Start nettleseren din. Hold CTRL-tasten inn og trykk så O. Da får du opp en dialogboks for å velge fil. Gå til mappen du valgte over og velg filen din “test.html” eller hva annet du eventuelt kalte den. Under er vist hvordan det ser ut hos meg.



Legg merke til at formateringen fra tekstdokumentet har blitt borte. Teksten “flyter” sammen.

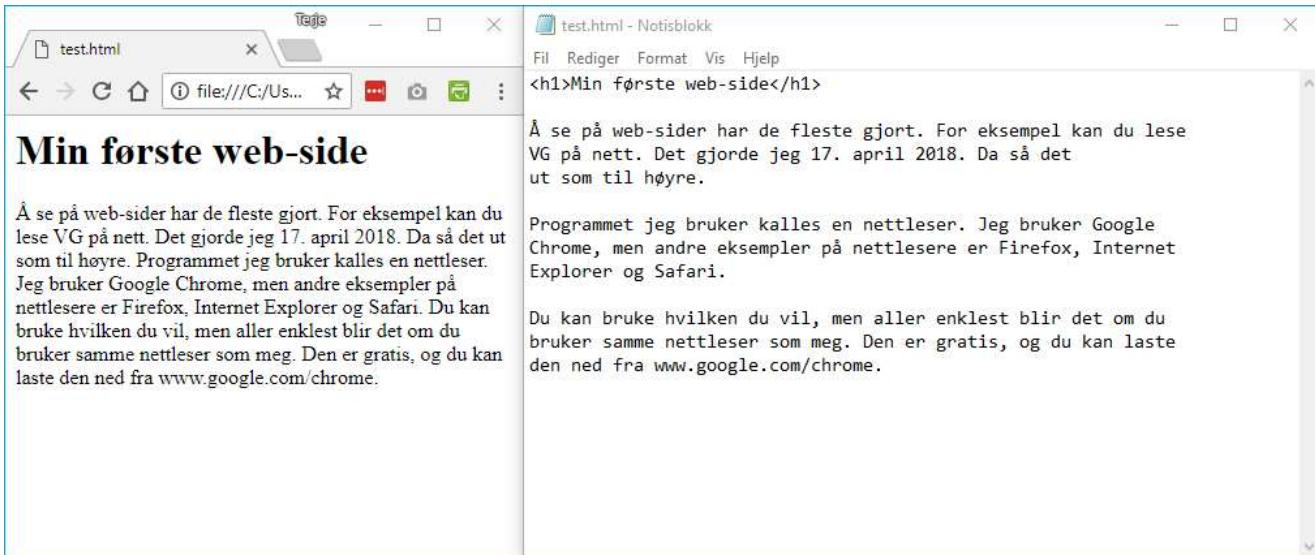
HTML

I HTML har ikke linjeskiftene i tekstdokumentet blitt bevart. Vi bruker tagger. En tag består av start-tag, innhold og slutt-tag. En tag kan for eksempel være **h1**, som vi kan tenke betyr “header 1” eller “overskrift nivå 1”.

Prøv å bytte ut første linje i html-filen din med dette:

```
<h1>Min første web-side</h1>
```

Lagre filen. Gå tilbake til nettleseren og klikk F5 for å lese inn filen på nytt. Hos meg ser det da slik ut:



```
<!DOCTYPE html>
<html>
<head>
    <title>Arkfaneoverskrift</title>
</head>
<body>
    <h1>Overskriften</h1>
    <p>Dette er et avsnitt.</p>
    <p>Dette er et avsnitt.</p>
</body>
</html>
```

Du har nå skrevet din første HTML-tag! Prøv selv tag-ene p, i og b. De står for *paragraph* (avsnitt), **bold**(fet skrift) og *italic* (kursiv).

Møt CSS

Er du klar for å endre på web-sidens farger, skriftstørrelse og mye mer? Da er du klar for CSS. Det står for *Cascading Style Sheets*. *Cascading* skal vi forklare senere, men *Style Sheets* betyr stilark, altså beskrivelse av stil, som du kan tenke på som form og farge.

Dette stilarket kan vi legge i HTML-filen. Og det hører til i en egen tag `<style>` inne i `<head>`. Alternativt kan man ha det i en egen fil med etternavnet `.css` - og referere til denne vha. en tag `<link>`, men i denne boken skal vi kun holde oss til `<style>`.

I CSS kan vi endre utseende til en tag. For eksempel kan vi endre forgrunnsfargen. Hva med blå overskrift og grønn tekst i avsnittene? Da blir det slik:

```
<!DOCTYPE html>
<html>
<head>
    <title>Arkfaneoverskrift</title>
    <style>
        h1 {
            color: blue;
        }
        p {
            color: green;
        }
    </style>
</head>
<body>
    <h1>Overskriften</h1>
    <p>
        Dette er et avsnitt 1. Dette er et
        avsnitt . Dette er et avsnitt 1.
        Dette er et avsnitt 1. Dette er et
        avsnitt 1. Dette er et avsnitt 1.
        Dette er et avsnitt 1. Dette er
        et avsnitt 1.
    </p>
    <p>
        Dette er et avsnitt 2. Dette er
        et avsnitt 2. Dette er et avsnitt 2.
        Dette er et avsnitt 2. Dette er
        et avsnitt 2. Dette er et avsnitt 2.
        Dette er et avsnitt 2. Dette er
        et avsnitt 2.
    </p>
</body>
</html>
```

Overskriften

Dette er et avsnitt 1. Dette er et avsnitt . Dette er et avsnitt 1.

Dette er et avsnitt 1. Dette er et avsnitt 1. Dette er et avsnitt 1.

Dette er et avsnitt 1. Dette er et avsnitt 1.

Dette er et avsnitt 2. Dette er et avsnitt 2. Dette er et avsnitt 2.

Dette er et avsnitt 2. Dette er et avsnitt 2. Dette er et avsnitt 2.

Dette er et avsnitt 2. Dette er et avsnitt 2.

Til avsnitt med tekst er `<p>`-tag-en fin. Men den vanligste taggen for programmerere er `<div>`. Den er en generell tag, som ikke har noen egen betydning. Den blir som en slags *container* for ulikt innhold, en *container* som vi kan *style* med CSS.

I eksempelet under er innholdet i body endret til å bruke `<div>`-tag-er. Videre viser det noen flere ting vi kan endre med CSS:

```
<head>
    <title>Arkfaneoverskrift</title>
    <style>
        h1 {
            color: white;
            background-color: blue;
        }

        div {
```

Overskriften



```

        color: green;
        background-color: lightgray;
        padding: 20px;
        margin: 40px;
        border: darkblue 1px solid;
        height: 70px;
        width: 50px;
    }

```

</style>

</head>

<body>

<h1>Overskriften</h1>

<div>A</div>

<div>B</div>

<div>C</div>

</body>



`background-color` styrer bakgrunnsfargen og `color` forgrunnsfargen. Fargene kan angis ved hjelp av innebygde fargenavn i HTML - eller ved hjelp av tallkoder. For vårt bruk er det greit med fargenavnene. Alle de vanlige fargene er støttet. Og en komplett oversikt kan du finne på www.w3schools.com/colors/colors_names.asp.

`border` slår på eller av en ramme rundt et element. Man angir en farge, en tykkelse og en strektype. `solid` er heltrukken linje, mens for eksempel `dashed` er en stiplet linje.

`padding` og `margin` styrer hvor mye luft det er rundt et element. Padding er luft innenfor rammen, og margin er utenfor.

`width` og `height` setter bredde og høyde. Du kan angi det i piksler (`px`), prosent (`%`), prosent av størrelsen på vinduet (`vw` og `vh`) og mye annet.

Så langt har vi bare sett på CSS som endrer alle forekomster av en tag. Nå skal vi se på hvordan to tilfeller av samme tag kan få ulik stil.

CSS klasser

CSS-koden under introduserer en egen regel for div-er som er tagget på en spesiell måte med css klassenavnet `specialEffect` . Dette ordet velger du helt selv.

```

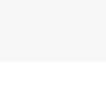
div.specialEffect {
    border: 4px dashed orange;
    padding: 5px;
}

```

```

<div>A</div>
<div class="specialEffect">B</div>
<div>C</div>

```



Så kan man altså tagge en div ved hjelp av ordet `class` på denne måten:

```
<div class="specialEffect">B</div>
```

Man kan godt angi flere CSS-klassenavn inne i `class=""`. Da må man passe på å ha mellomrom mellom dem.

Ordet *cascading* i *cascading style sheets* står nettopp for at det kan være flere regler som gjelder et bestemt element. Og disse trer i kraft fra generell til mer spesifikt. Det mer spesifikke overstyrer altså det generelle. Det vil si at den orange rammen dukker opp på alle div-er som har `class="specialEffect"`.

Merk at når vi skriver `div.specialEffect` i CSS, så kan bare `class="specialEffect"` brukes på div-er. Hvis du vil ha en klasse som kan brukes på alle tagger, så må du skrive kun `.specialEffect`.

I dette konkrete eksemplet er det forresten brukt `display: inline-block` for alle div-ene. Det gjør at de kommer på linje bortover istedenfor under hverandre. Det er også brukt en annen `width` og `height` enn i det forrige eksemplet. (Og grunnen til at padding er satt for `specialEffect` er for at firkanten i sum skal bli like stor når rammen er tykkere.)

Javascript - grunnferdigheter

Nå skal vi gå gjennom en del grunnferdigheter i Javascript-programmering - grunnferdigheter som er relevante for web, dvs. i kombinasjon med HTML og CSS.

Senere kommer en grundigere gjennomgang av de samme emnene, men i første runden introduserer vi kun helt enkle eksempler på en del nye teknikker.

Noen av ferdighetene er helt spesifikke på Javascript-programmering mot en web-side. Disse er merket mer klappende hender: 

Andre ferdigheter går på grunnleggende programmeringsteknikker som du vil finne igjen i mange andre programmeringsspråk - også i helt andre sammenhenger enn web. Disse er merket - med en flekset biceps: 

Noen grunnferdigheter dekker begge deler og er derfor merket med begge symbolene.

Ferdighet 1 - alert

Aller først trenger vi en helt enkel kommando som gjør noe - hva som helst egentlig. Og da er `alert()` et bra valg.

Legg inn følgende inne i body-taggen:

```
<script>
    alert('Hei på deg!');
</script>
```

Når du så laster html-filen din i nettleseren, vil du få en popup:



Javascript-koden inne i `script`-taggen kjøres mens siden lastes. Senere skal vi se at det derfor er viktig hvor i `body`-taggen at vi plasserer dette. Det vanlige er å ha det helt til slutt, for da er alle HTML-elementene lest inn. Men det kan forsåvidt ligge hvor som helst.

Det finnes kommandoer av mange typer. `alert` er en innebygget funksjon i Javascript.

I linjen `alert('Hei på deg!');` ser vi at vi bruker navnet på den innebygde funksjonen. Så kommer en venstre parentes. Det vil det alltid gjøre når man skal bruke en funksjon - enten funksjonen er innebygget eller ikke.

Noen funksjoner tar en eller flere såkalte parametere. Det er verdier som sendes inn til funksjonen. `alert` popper opp en boks, og som parameter kan man sende med selve teksten man vil skal vises. I Javascript bruker vi enkeltfnutter for å angi en tekst. Et annet ord er apostrof, altså `'`. Det er lov å bruke vanlig anførselstegn også, men du må alltid bruke samme tegn som start og slutt - du kan ikke starte med enkeltfnutt og avslutte med dobbeltfnutt.

I dette eksemplet er parameteren altså teksten `'Hei på deg!'`. Det går an å kalle `alert` og andre funksjoner uten parametre, men du må likevel ha med parentesene: `alert();`.

Videre avsluttes alle kommandoer i Javascript med et semikolon.

Om du ikke får opp en dialogboks, kan det hende du har skrevet noe feil. Hvis du bruker nettleseren Google Chrome, kan du trykke F12 (i Windows-versjonen) eller gå inn i menyen og velger Flere verktøy -> Utviklerverktøy. I høyre del av nettleservinduet får du da opp dette:

The screenshot shows the browser's developer tools open to the 'Console' tab. At the top, there are tabs for 'Elements', 'Console' (which is selected), 'Sources', 'Network', and 'Performance'. On the right side of the toolbar, there are icons for 'Default levels', 'Group similar', and a settings gear. Below the tabs, there is a search bar with the placeholder 'Filter' and a dropdown menu set to 'top'. A red error message is displayed: 'Uncaught SyntaxError: missing) after argument list' at line 10 of 'HTMLPage2.html'. To the right of the message is the file name 'HTMLPage2.html:10'. There is also a small 'x' icon and a 'Group similar' checkbox.

Pass på å velge "Console" i toppraden om den ikke er valgt i utgangspunktet.

Her står det at det er en feil på linje 10 i filen HTMLPage2.html. Du kan trykke på filnavnet og komme direkte dit hvor feilen er:

The screenshot shows a browser window with the title 'HTMLPage2.html'. The page content is the source code of the HTML file. Line 10 contains a script tag with an alert statement: 'alert('Hei')'. The closing parenthesis ')' has a red squiggly underline and a red 'x' icon, indicating a syntax error. The rest of the code is valid HTML.

```

1 <!DOCTYPE html>
2
3 <html lang="en" xmlns="http:
4 <head>
5   <meta charset="utf-8" />
6   <title></title>
7 </head>
8 <body>
9   <script>
10    alert('Hei') ×
11   </script>
12 </body>
13 </html>

```

I dette tilfellet har jeg glemt den avsluttende høyreparentesen og semikolon. Ikke alle feil er like enkle å kjenne igjen. Ofte kan det være en feil tidligere i koden, som ikke er ugyldig Javascript, men som fører til at annen kode blir feil.

Ferdighet 2 - onclick

Det er to hovedsteder vi kommer til å legge Javascript-kode. Det ene er inne i `script`-taggen, og det har vi alt sett på. Det andre er `onclick` og andre event-attributter inne i html-tagger. `onclick` er den første av disse som vi skal de på, men det finnes mange flere. (På www.w3schools.com/jsref/dom_obj_event.asp får du full oversikt.)

Her er et eksempel:

```
<h1 onclick="alert('Takk :-)');">Trykk på meg!</h1>
```

I dette eksemplet er det først når brukeren trykker på overskriften "Trykk på meg!" at dialogboksen popper opp.

Ferdighet 3 - Egne funksjoner

La oss si at du vil ha flere dialogbokser etter hverandre, f.eks.

```
alert('Hei');
alert('på');
alert('deg!');
```

Dette kan du ha inne i en `script`-tag, eller du kan ha det inne i en `onclick`. Faktisk kan du ha flere linjer med kommandoer inne i en `onclick`, men det blir vanskelig å lese. Flyten i HTML-koden forsvinner.

Da er det bedre å lage din egen funksjon, dvs. å på en måte lage sin egen kommando. Du kan deklarer en funksjon på denne måten - inne i en `script`-tag:

```
function heiPåDeg() {
    alert('Hei');
    alert('på');
    alert('deg!');
}
```

Nå kan du ha `onclick="heiPåDeg()"` og alt er bare velstand.

Når du deklarer en funksjon, må begynne med ordet `function`, deretter navnet på funksjonen. Og dette navnet velger du helt selv. Bare pass på å ikke bruke et ord som allerede er et ord i Javascript-spåket. Så en funksjon som heter "function" er altså en dårlig idé.

Etter navnet på funksjonen kommer som minimum tomme parenteser. Om funksjonen skal ha parametre, så kommer de mellom parentesene, men det skal vi lære om senere.

Så kommer det krøllparenteser (Alt Gr + 7 og 9 i Windows - Alt + Shift + 8 og 9 på Mac). Inne i krøllparentesene lister du opp alle Javascript-kommandoene du vil skal inngå i Javascript-funksjonen din.

Ferdighet 4 - Endre innholdet av en tagg ut fra id

Så langt har vi ikke funnet på så mye moro med det vi har lært, men nå skal vi endre på det. En hvilken som helst del av HTML-dokumentet ditt kan nemlig endres til hva du vil ved hjelp av Javascript.

Se på eksemplet under og test det gjerne selv - inne i en `body`-tag:

```
<h1 id="overskrift" onclick="endreTekst()">Trykk på meg!</h1>
<script>
    function endreTekst() {
        document.getElementById('overskrift').innerHTML = 'Takk :-)';
```

```
}
```

```
</script>
```

I den første linjen, `h1`-taggen, er det meste som før, men i tillegg har vi en `id`-attributt: `id="overskrift"`. Dette er en merkelapp vi velger selv og som gjør at vi kan finne tilbake til akkurat denne `h1`-taggen fra Javascript.

Og nettopp det er det vi gjør ved hjelp av `document.getElementById('overskrift').innerHTML`. Den ene kommandoen vi har inne i funksjonen vår endrer den "indre" HTML-koden i `h1`-taggen til "Takk 😊", som om vi i HTML hadde skrevet `<h1>Takk :‐)</h1>` istedenfor.

Denne kommandoen er en tilordningssetning. Vi tilordner en variabel til en bestemt verdi. Den bestemte verdien har vi på høyre side av likhetstegnet, og hvilken variabel som skal endres har vi på venstre side. Hva variabler er skal vi lære mer om litt senere. Men i denne omgang holder det å forstå at dette kan brukes til å endre innholdet i en tagg.

For dem som vil høre mer detaljer allerede nå, så er `document` et objekt som representerer HTML-dokumentet. Du vil for eksempel finne at både `document.body` og `document.head` er gyldig - og disse viser jo da til innholdet i `body` og `head` i HTML. `document`-objektet har en funksjon som heter `getElementById`. Denne funksjonen tar en tekst som parameter. Hvis det finnes en HTML-tagg i ditt dokument som inneholder en attributt `id` med verdi lik denne teksten, så vil funksjonen *returnere* et objekt som representerer denne taggen. (Hva det vil si å returnere blir forklart ordentlig først senere i teksten.) Dette objektet har et *felt* som heter `innerHTML`. (Hva et felt er forklares også senere.) Når vi endrer innholdet i `innerHTML`, endrer vi også innholdet i taggen i selve HTML-dokumentet - og dermed også hva som faktisk vises på skjermen.

I eksemplet er det samme tagg som har `onclick` og som blir endret på. Sånn trenger det absolutt ikke være. I eksemplet under er det to `h1`-tagger. Den ene er til å trykke på, og den andre er til å bli endret:

```
<h1 onclick="endreTekst()">Trykk på meg for å endre overskriften under!</h1>
<h1 id="overskrift" >Trykk på overskriften over for å endre meg!</h1>
<script>
    function endreTekst() {
        document.getElementById('overskrift').innerHTML = 'Takk :‐)';
    }
</script>
```

Et siste poeng i denne runden om `innerHTML` er at det går an å skrive `+=` istedenfor `=`. Mens `=` betyr å endre til en ny verdi, så betyr `+=` å legge til en ny verdi i tillegg til det som er der fra før. I eksemplet under legges det til en prikk for hver gang det er trykket

```
<b id="overskrift" onclick="endreTekst()">Trykk for prikker:</b>
<script>
    function endreTekst() {
        document.getElementById('overskrift').innerHTML += ' •';
    }
</script>
```

Og slik ville det da sett ut etter at brukeren har trykket tre ganger:

Trykk for prikker: •••

Ferdighet 5 - Nøkkelordet `this` og en forsøk på parametre

Se på disse to kodelinjene:

```
<button onclick="alert(this.innerHTML)">Hei!</button>
<button onclick="alert(this.innerHTML)">Hallo!</button>
```

Hva skjer om vi trykker på den første? Og den andre? Den første fører til en alert-boks med teksten "Hei!", mens den andre fører til en alert-boks med teksten "Hallo!". Koden i `onclick` er den samme. Likevel blir resultatet annerledes.

Nøkkelordet `this` har en spesialbetydning når det brukes i `onclick` eller andre eventattributter i HTML. Det betyr "denne" eller "dette", dvs. dette elementet - denne taggen. (Denne button-taggen i eksemplet over.)

På samme måte kan vi endre `innerHTML` til det elementet man trykker på:

```
<button onclick="this.innerHTML = 'x'">Hei!</button>
<button onclick="this.innerHTML = 'x'">Hallo!</button>
```

Så er det bare at om vi ønsker å gjøre mer avanserte ting, så vil vi gjerne samle flere kommandoer i en funksjon. Hvordan får vi så verdien av `this` over til funksjonen?

Nøkkelordet `this` har ikke den samme betydningen inne i `<script>`-taggen, så til vår bruk må vi ha `this` i HTML-koden.

Løsningen blir å sende med en såkalt parameter når vi kaller en funksjon. Det har vi faktisk gjort allerede. I uttrykket `alert('Hei!');` er `'Hei!'` en parameter. Det er en verdi vi sender meg når vi kaller funksjonen `alert`. For `alert` sin del er det den teksten som skal vises vi sender med som parameter. Det kan være en helt bestemt tekst som i `'Hei!'` eller det kan være en verdi fra en variabel eller et uttrykk som i `alert(this.innerHTML)`.

Så la oss gå tilbake til et tidligere eksempel, men ikke bruke `id` og `getElementById` til å finne frem til et element. Istedentfor skal vi bruke `this`. Sammenlign hvordan det gjøres under til venstre, mens et tidligere eksempel er vist til høyre som sammenligning:

```
<h1 onclick="trykk(this)">Trykk!</h1>
<h1 onclick="trykk(this)">Trykk!</h1>
<script>
    function trykk(trykketPåTag) {
        trykketPåTag.innerHTML = 'x';
    }
</script>
```

```
<h1 onclick="trykk()">Trykk!</h1>
<h1 id="o">Trykk!</h1>
<script>
    function trykk() {
        document
            .getElementById('o')
            .innerHTML = 'x';
    }
</script>
```

Navnene på funksjonen og variablene samt noen av tekstene er endret, så begge eksemplene skulle få plass i bredden. Innholdet i funksjonen til høyre er formattert over tre linjer av samme grunn - men den fungerer likevel helt likt som om den var skrevet slik:

```
document.getElementById('o').innerHTML = 'x';
```

I eksemplet til høyre kan man klikke på det første `h1`-elementet, og teksten på det andre `h1`-elementet blir da endret. I eksemplet til venstre kan begge `h1`-elementene klikkes på, og teksten blir endret på den du trykker på.

Videre er det en forskjell mellom parentesene i deklarasjonen av vår funksjon `endreTekst`. I versjonen til venstre tar vi i mot en parameter ved å velge et navn på denne parameteren til bruk i vår funksjon. Vi velger helt fritt et navn som brukes til å referere til den verdien som sendes inn.

I eksemplet har vi valgt `taggenSomBleTrykketPå`, men det kunne vært `x`, `element` eller noe helt annet.

Det ordet vi velger kan så brukes inne i funksjonen - og den refererer da til den faktiske verdien som ble sendt med som parameter. Hvis vi trykker på det *første* `h1`-elementet i eksemplet til venstre, så vil `taggenSomBleTrykketPå` vise til nettopp det *første* `h1`-elementet. Hvis vi trykker på det *andre* `h1`-elementet, vil `taggenSomBleTrykketPå` vise til det *andre* `h1`-elementet.

Effekten er det samme som i det forrige eksemplet lenger opp. `h1`-elementet du trykker på endrer tekst til `'x'`.

Merk at i eksemplet til høyre, kan vi godt sette `onclick="endreTekst()"` på begge `h1`-taggene, men likevel vil bare den andre bli endret på, siden funksjonen endrer den `h1`-

taggen med `id="overskrift"`. Nøkkelordet `this` derimot lar oss enkelt gjøre endringer på det elementet man klikker på.

Ferdighet 6 - Sette sammen tekst av flere deler

I Javascript kalles pluss-tegnet (`+`) for en operator. Dersom du bruker den på tall, betyr den vanlig addisjon, men hvis du bruker den på tekst, betyr den å sette sammen tekst av flere deler.

Hvilken tekst tror du denne alert-kommandoen vil vise?

```
alert('Hei' + ' på' + ' deg!');
```

Den vil vise `Heipådeg!`. Den har altså *plusset* sammen de tre tekstene til én.

Vi kan ha så mange deler vi vil. Og hver av dem kan være enten en fast tekst, som i eksemplet over, eller et hvilket som helst Javascript-uttrykk. Det kan vi bruke til å lage en litt mer personlig hilsen:

```
<button onclick="alert('Hei, ' + this.innerHTML + '!')>Per</button>
<button onclick="alert('Hei, ' + this.innerHTML + '!')>Pål</button>
<button onclick="alert('Hei, ' + this.innerHTML + '!')>Espen</button>
```

Disse knappene vil vise teksten `Hei, Per!`, `Hei, Pål!` eller `Hei, Espen!`.

Ferdighet 7 - Logging til konsollet

Dialogboksene som vi lager ved hjelp av alert-kommandoen kan være ganske irriterende, fordi vi må klikke på "Ok" for å få dem bort.

Og skal vi gi beskjeder til brukeren, er det bedre å vise informasjon i en egen div - som en del av brukergrensesnittet. Denne boksen kan skjules når den ikke er i bruk.

Når vi programmerer, har vi ofte behov for å dokumentere hva som skjer i løpet av programkjøringen. Alert-kommandoen gjør det mulig, men et bedre alternativ er `console.log`. Den skriver tekst til en egen logg, som du kan se i utviklerverktøyet i nettleseren. Test det ut, for eksempel med koden under:

```
<script>
  console.log('Hei');
  console.log('på');
  console.log('deg!');
</script>
```

Lagre dette i en html-fil og åpne den i nettleseren. Siden vil være tom, men om du åpner utviklerverktøy (F12 i Chrome) og velger "Console", vil du se dette:

```

```

Til høyre står navnet på filen (HTMLPage1.html) og linjenummer (10, 11 og 12). Dette er en lenke som du kan trykke på. Da vil Chrome vise deg kildekoden og markere den aktuelle linjen. Å kunne logge underveis i en programkjøring er nyttig for å lære seg å forstå hva som foregår.

⌚ Ferdighet 8 - Lagre tall i variabler

Variabler gjør at programmet vårt kan huske. Tenk på en variabel som en skuff i en kommode:



I Javascript gjør vi tre ting med variabler:

1. Deklarere
2. Tilordne
3. Lese innhold

De to siste gir kanskje mest mening nå. Vi kan legge noe i skuffen (tilordne) og hente det ut igjen (lese innhold). Men før det må vi ha en slags merkelapp på variabelen vi vil bruke, vi vil gi den et navn. Tenk på å deklarere som å sette en merkelapp på en skuff. La oss si at jeg vil ha en variabel som skal brukes til å huske hvor mange ganger brukeren har trykket på en knapp. Jeg vil kalle variabelen `teller`, og da *deklarerer* jeg den slik:

```
var teller;
```

Tenk på ordet "var" som en forkortelse for "variabel". Linjen gjør at det holdes av en skuff til dette - med merkelappen "teller":



Så kan vi legge noe i skuffen ved hjelp av en *tilordning*:

```
teller = 0;
```

Da legger vi tallet 0 i skuffen:

I Javascript får vi også lov til å *deklarere* og *tilordne* på en gang:

```
var tall = 123;
```

Da får vi en skuff med merkelapp og innhold på en gang:



Merk at vi deklarerer en variabel kun én gang. Så denne siste skrivemåten bruker vi maksimalt én gang per variabel.

Så kommer selvsagt at vi kan lese ut innholdet av en variabel. For eksempel vi `console.log(tall)` skrive ut tallet 123. Merk forskjellen, `console.log('tall')` vil skrive ut teksten "tall", mens den første varianten altså skriver ut *innholdet* av variablene tall.

Vi kan også lese ut innholdet av en variabel og tilordne det til en annen. Anta at begge variablene er deklarert fra før:

```
a = 5;
b = a;
```

Etter dette vil begge variablene ha verdien 5.

Helt til slutt; Variabler kan selvsagt *variere*, det vil si at innholdet av en variabel kan endres mange ganger i løpet av et program.

```
a = 5;
b = a;
a = 6;
a = 7;
a = 8;
```

Etter denne koden vil ha verdien 8, mens b vil være 5, fordi a var 5 da vi leste ut verdien av a og tilordnet det til b.

Ferdighet 9 - Lagre tekst i variabler

Tekst kan lagres i variabler akkurat som tall. Forskjellen er at vi bruker *enkeltnutter* rundt verdien:

```
var navn = 'Terje';
```



Uten fnutter ville det samme bety "sett verdien av variabelen navn til innholdet av variabelen Terje":

```
var navn = Terje;
```

Javascript skiller mellom stor og små bokstaver, så `Terje` og `terje` ville referert til to forskjellige variabler.

Når vi tilordner en variabel kan vi bruke et hvilket som helst gyldig Javascript-uttrykk. I ferdighet 6 så vi at vi kan sette sammen flere tekster til én - og resultatet kan vi tilordne til en variabel:

```
var tekst = 'Hei' + ' på ' + 'deg!';
console.log(tekst);
```

Dette skriver "Hei på deg!" til konsollet. Å trekke ut verdier i en variabel gjør det lettere å lese og forstå koden.

Vi kan også hente ut innholdet i en div til en variabel:

```
var divHtml = document.getElementById('div5').innerHTML;
```

Ferdighet 10: Lagre objekter i variabler

Alle gyldige verdier i Javascript kan lagres i en variabel. Et eksempel på noe annet enn tekst og tall er et objekt som refererer til en tagg i DOM, dvs. *document object model*.

Gitt at vi har en `<div id="mainContent">A</div>`, så kan vi som kjent endre innholdet av denne ved hjelp av kommandoen:

```
document.getElementById('mainContent').innerHTML = 'B!';
```

Dette lange uttrykket kan vi gjøre med leselig ved å dele det i to:

```
var mainContentDiv = document.getElementById('mainContent');
mainContentDiv.innerHTML = 'B!';
```

Spesielt om vi har behov for å referere til `document.getElementById` flere ganger med samme id, så vil koden bli enklere ved å mellomlagre elementet i en variabel.

La oss se på et eksempel hvor vi har to div-er med id "divA" og "divB" og at vi ønsker å bytte innholdet mellom de to. Vi må sette innholdet av divA til innholdet av divB og vice versa. Men i det vi setter innholdet av divA til innholdet av divB, så har vi mistet det opprinnelige innholdet av divA. Derfor må vi begynne med å hente ut dette og lagre det i en variabel:

```
var contentTmp = document.getElementById('divA').innerHTML;
document.getElementById('divA').innerHTML = document.getElementById('divB').innerHTML;
document.getElementById('divB').innerHTML = contentTmp;
```

Denne koden kan gjøres mer lesbar og bedre ved å hente ut elementene for divA og divB i egne variabler:

```
var divA = document.getElementById('divA');
var divB = document.getElementById('divB');
var contentTmp = divA.innerHTML;
divA.innerHTML = divB.innerHTML;
divB.innerHTML = contentTmp;
```

Ferdighet 11: Endre andre ting enn innerHTML: value og style og classList

Til nå har vi kun lest og endret `innerHTML` på et element i HTML-dokumentet.

Men når vi bruker en tekstboks til å hente input fra brukeren, så er det ikke `innerHTML`. Vi er mest interessert i. For å få en tekstboks som er forhåndsutfyld med teksten "Hallo", må vi bruke denne HTML-koden: `<input id="navn" type="text" value="Hallo"/>` og ikke denne: `<input id="navn" type="text">Hallo</input>`.

Det er altså ikke `innerHTML` vi er interessert i her, men `value`. Gitt en `input`-tag med `id="navn"`, så henter vi (eller endrer) det som er fylt ut ved hjelp av denne koden:

```
var navnFraSkjema = document.getElementById('navn').value;
```

I denne sammenhengen ville ikke `innerHTML` gjøre noen nytte.

Og det er mer. Først og fremst vil vi ønske å kunne endre css-properties på et html-element. Og det kan vi gjøre via Javascript. Det er to måter. Det ene er å sette eller fjerne en css klasse, og det andre er å sette properties direkte. Kodeeksemplet under demonstrerer begge deler:

```
var minDiv = document.getElementById('minDiv');
minDiv.style.backgroundColor = '#4567ab';
minDiv.style.color = '#4567ab';
minDiv.style.padding = '4px';
minDiv.classList.add('active');
minDiv.classList.remove('old');
minDiv.classList.toggle('something');
```

Merk at alt innenfor `style` skrives på en annen måte enn i css-filen. Istedentfor `background-color` får vi `backgroundColor`, mens `color` er likt begge steder - og det er alle enkeltord. Men ting som består av flere ord har ikke bindestrek i Javascript. Bindestrekken fjernes - og neste bokstav gjøres stor.

For å legge til en css-klasse, brukes `classList` og `add`. Hvis `minDiv` før koden over var:

```
<div class="something old"></div>
```

Så ville den etter koden være slik:

```
<div class="active"></div>
```

Koden `minDiv.classList.toggle('something')` vil altså fjerne '`something`' hvis den er der fra før. Hvis den ikke er der fra før, vil den *legge den til*.

Ferdighet 12: Bygge html ved å sette sammen tekst av flere deler

Vi har sett at vi kan "plusse" sammen flere tekster til én. Dette kan vi bruker til å bygge html dynamisk ved hjelp av Javascript.

La oss lage et eksempel med en div og en knapp. Når man trykker på knappen, skal innholdet i div-en byttes ut med en punktliste med to punkter, hvor hvert punkt inneholder det div-en inneholdt fra før:

```
<div id="content">Hallo!</div>
<button onclick="wrap()">Wrap!</button>
```

```
function wrap() {
    var contentDiv = document.getElementById('content');
    var html = contentDiv.innerHTML;
    contentDiv.innerHTML = '<ul><li>' + html + '</li><li>' +
        html + '</li></ul>';
}
```

Når vi laster siden, får vi dette:

Hallo!

Etter ett trykk:

- Hallo!
- Hallo!

Og etter to trykk:

- ◦ Hallo!
- ◦ Hallo!
- ◦ Hallo!
- ◦ Hallo!

Ferdighet 13: Variable - innenfor og utenfor funksjon

En variabel som deklarerется *inne i* en funksjon lever bare mens funksjonen kjører. Det vil si at når funksjonen starter, holdes det av en skuff, og denne kan brukes av alle koden inne i funksjonen, men når funksjonen er ferdig, forsvinner variablene igjen. Merkelappen på skuffen fjernes og skuffen er ledig til annen bruk.

En slik variabel kan derfor ikke brukes til for eksempel å telle opp hvor mange ganger man har trykket på en knapp. For å få til det må man deklarere variablene *utenfor* funksjonen. Da blir det en såkalt global variabel.

Gitt en `<button onclick="tell()">Tell</button>`, så vil følgende Javascript-kode telle antall klikk på knappen:

```
var antallKlikk = 0;

function tell() {
    antallKlikk++;
    console.log(antallKlikk);
}
```

I tillegg til å øke variablene, logger vi her også verdien til konsollet. Hvis vi hadde hatt `var antallKlikk = 0;` inne i funksjonen, så ville ikke dette fungert. Da ville verdien begynt på 0 på nytt hver gang vi trykket på knappen, og den ville da alltid telle opp til 1 hver gang.

La oss se på et annet eksempel. Vi har tre knapper:

```
<button onclick="merk(this)">A</button>
<button onclick="merk(this)">B</button>
<button onclick="merk(this)">C</button>
```

Når man trykker på en knapp, skal den valgte knappen få en markering, for eksempel en stiplet oransje ramme. Det kan vi gjøre slik:

```
function merk(element){
    element.style.border = 'dashed orange 2px';
}
```

Nå vil alle knapper du trykker på, få denne effekten. *Men vi vil bare ha den på den siste knappen som ble trykket på.* For å få til det må vi lagre, i en variabel, elementet som ble trykket på forrige gang:

```
var forrigeElement;

function merk(element){
    element.style.border = 'dashed orange 2px';
    forrigeElement.style.border = 'none';
    forrigeElement = element;
}
```

Problemet med dette er at første gang vi kommer til `forrigeElement.style.border = 'none';`, så har ikke `forrigeElement` noen verdi, og linjen vil feile. (Og dermed blir heller ikke `forrigeElement` satt til en verdi til neste gang heller.)

Løsningen på dette er å gi variablen `forrigeElement` verdi fra start. Det kan vi få til slik:

```
<button id="knappA" onclick="merk(this)">A</button>
<button onclick="merk(this)">B</button>
<button onclick="merk(this)">C</button>
```

```
var forrigeElement = document.getElementById('knappA');

function merk(element){
    element.style.border = 'dashed orange 2px';
    forrigeElement.style.border = 'none';
```

```
    forrigeElement = element;
}
```

Da vil dette virke som vi ønsker, nemlig at hver gang vi trykker på en ny knapp, får denne markering *i tillegg til* at markeringen fjernes fra den knappen som var markert.

Ferdighet 14: Sette variabel til resultat av et regnestykke

I Javascript har vi tilgang til alle de vanlige matematiske operasjonene. De fire grunnleggende regneartene er tilgjengelige som *operatorer*. For eksempel vil `var x = 1 + 2` sette variabelen x til summen av 1 og 2, altså 3. `var x = a + b` vil på samme måte sette verdien av x til summen av verdiene av innholdet av variabel a og variabel b.

Merk at om en eller begge variabler inneholder tekst, så vil `+` betyr *konkatenering*, dvs. å sette sammen tekst av flere deler. `'1' + 1` vil altså ikke bli 2, men teksten `'11'`. Man kan tvinge Javascript til å tolke en verdi som tall ved hjelp av funksjonen `parseInt`. `var x = parseInt(a) + parseInt(b)` vil alltid bli addisjon av tall - og aldri konkatenering av tekst. Dersom innholdet i a eller b ikke kan konverteres til tall, vil programmet feile. Man kan selvsagt gjøre dette i flere steg og mellomlagre i egne variabler som i følgende eksempel hvor x får verdien 12:

```
var tekstA = '5';
var tekstB = '7';
var tallA = parseInt(tekstA);
var tallB = parseInt(tekstB);
var x = tallA + tallB;
```

Eksemplet under lager to sliders som lar brukeren velge to tall. Programmet viser hele tiden resultatet av alle de fire regneartene på de to tallene, altså pluss, minus, gange og dele.

```
<input id="tallA" type="range" min="1" max="100" step="1" value="2"
      oninput="calculate()" />
<input id="tallB" type="range" min="1" max="100" step="1" value="2"
      oninput="calculate()" />
<div id="resultat"></div>
<script>
    var resultatDiv = document.getElementById('resultat');
    var tallAInput = document.getElementById('tallA');
    var tallBInput = document.getElementById('tallB');
    calculate();
    function calculate() {
        var tallA = parseInt(tallAInput.value);
        var tallB = parseInt(tallBInput.value);
        var sum = tallA + tallB;
        var differense = tallA - tallB;
        var produkt = tallA * tallB;
        var divisjon = tallA / tallB;
        resultatDiv.innerHTML = `Sum: ${sum}, Differanse: ${differense}, Produkt: ${produkt}, Divisjon: ${divisjon}`;
    }
</script>
```

```

var produkt = tallA * tallB;
var kvotient = tallA / tallB;
resultatDiv.innerHTML = '' +
    tallA + '+' + tallB + '=' + sum + '<br/>' +
    tallA + '-' + tallB + '=' + differense + '<br/>' +
    tallA + '*' + tallB + '=' + produkt + '<br/>' +
    tallA + '/' + tallB + '=' + kvotient;
}
</script>

```

Merk at matematikken (også i Javascript) har regler for hvilke regneoperasjoner som gjøres først. For eksempel regnes multiplikasjon før addisjon, så $2+3*4$ blir 14. $3*4$ regnes ut først. Det blir 12. Så regnes ut $2+12$. Dersom man vil overstyre dette - eller bare være helt sikker på at rekkefølgen blir som man ønsker, så kan man bruke parenteser. Dersom man ønsker at $2+3$ blir regnet ut først i eksemplet, må man altså skrive $(2+3)*3$.

Ferdighet 15: Sette variabel til tilfeldig tall

Mer avanserte matematiske funksjoner ligger i Javascript i et objekt som heter Math. For eksempel kan du regne ut kvadratrot ved hjelp av `Math.sqrt(81)`.

Andre nyttige funksjoner derfra er Math.round, som runder av tall, Math.pow som regner ut potens, samt Math.max og Math.min som finner det høyeste eller laveste tallet av to eller flere tall.

Math.random gir oss tilfeldige tall, men den gir alltid tall mellom 0 og 1, så vi må jobbe litt med det om vi vil ha f.eks. et helt tall mellom 1 og 10. For å få til det, kan vi gjøre

`Math.random() * 10`, men vi vil fortsatt få med en masse desimaler. For å fjerne dem, kan vi runde av, men Math.round runder alt fra 0,5 oppover, så det er bedre å bruke Math.floor, som alltid runder av nedover - eller sagt på en annen måte rett og slett fjerner desimalene.

`Math.floor(Math.random() * 10)` gir oss derfor et tilfeldig tall som er minimum 0 og maksimum 9. Dersom vi ønsker minimum 1 og maksium 10, så må vi legge til 1, dvs. `1 + Math.floor(Math.random() * 10)`. Dersom vi vil ha et tilfeldig tall som er minimum 10 og maksimum 19, så må vi legge til 10, altså `10 + Math.floor(Math.random() * 10)`, mens om vi ønsker minimum 10 og maksium 99, så blir det `10 + Math.floor(Math.random() * 90)`. Det er fordi `Math.random() * 90` kan bli maksimalt 89,999999... og dermed kun 89 etter at desimalene er fjernet.

I eksemplet under er eksemplet fra forrige grunnferdighet skrevet om slik at det velges to tilfeldige tall istedenfor at brukeren velger dem vha. sliders:

```

var resultatDiv = document.getElementById('resultat');
calculate();
function calculate() {
    var tallA = 100 + Math.floor(Math.random() * 10);
    var tallB = 100 + Math.floor(Math.random() * 10);
    ...
}

```

```

var tallB = 100 + Math.floor(Math.random() * 10);
var sum = tallA + tallB;
var differense = tallA - tallB;
var produkt = tallA * tallB;
var kvotient = tallA / tallB;
resultatDiv.innerHTML = '' +
    tallA + '+' + tallB + '=' + sum + '<br/>' +
    tallA + '-' + tallB + '=' + differense + '<br/>' +
    tallA + '*' + tallB + '=' + produkt + '<br/>' +
    tallA + '/' + tallB + '=' + kvotient;
}

```

Ferdighet 16: Alternative eventhåndlere: onchange OSV.

Når vi jobber med Javascript på en webside, er det ofte onclick eller andre eventhåndlere som setter igang ting. I mange sammenhenger er det ikke onclick som er mest relevant. For eksempel i et tekstmeldt (`<input type="text"/>`), så er det ikke klikk vi er ute etter, men endringer i teksten. Da kan vi bruke onchange eller oninput.

De to oppfører seg litt forskjellig. Oninput vil slå til hver gang man taster et nytt tegn (eller sletter et tegn), mens onchange først slår til når man flytter fokus bort fra tekstmeldsen (og det er endring).

Under er en tekstmeldt med eventer for onclick, onchange og oninput:

```

<input type="text"
    onclick="console.log('onclick ' + this.value)"
    oninput="console.log('oninput ' + this.value)"
    onchange="console.log('onchange ' + this.value)"/>

```

Hvis jeg først klikker i tekstmeldsen, skriver "hei" og deretter klikker et annet sted på websiden, vil jeg få følgende tekst i Console:

```

onclick
oninput h
oninput he
oninput hei
onchange hei

```

Dette viser i hvilken rekkefølge eventene skjer - samt hva verdien (`value`) av tekstmeldtet er på hvert tidspunkt.

Ferdighet 17: Valgsettninger: sammenligne tall med if

Valgsetninger gjør det mulig å kjøre en eller flere kommandoer bare i visse tilfeller. For å avgjøre om kommandoene skal kjøres eller ikke må vi angi en *logisk verdi*, også kalt et *logisk uttrykk*. Logiske uttrykk er alltid enten sanne eller usanne, og i Javascript skrives dette som `true` og `false`.

Vi skal se mange former logiske uttrykk, men i denne omgangen skal vi se på sammenligning av tall. La oss gå tilbake til eksemplet som teller hvor mange ganger brukeren har klikket på en knapp, men omskrevet slik at vi viser teksten i en div istedenfor i konsollet:

```
<button onclick="tell()">Tell</button>
<div id="info"></div>
<script>
var antallKlikk = 0;
var infoDiv = document.getElementById('info');

function tell() {
    antallKlikk++;
    infoDiv.innerHTML = 'Du har klikket ' + antallKlikk + ' ganger.';
}
</script>
```

Nå vil vi ikke lenger skrive ut hvor mange ganger brukeren har klikket, men bare en tekst som forteller at brukeren har klikket minst ti ganger. Og denne teksten skal først vises når brukeren faktisk har klikket ti ganger. For å få til dette, må vi bruke en `if`-setning. Da blir innholdet i funksjonen `tell()` slik:

```
antallKlikk++;
if( antallKlikk == 10 ) {
    infoDiv.innerHTML = 'Du har klikket 10 ganger.';
```

Som før øker vi innholdet i variabelen `antallKlikk` med én hver gang funksjonen kjøres. Så kommer `if`-setningen. Det er kommandoen `infoDiv.innerHTML = 'Du har klikket 10 ganger.'` som her bare kjøres *hvis* det er sant at `antallKlikk` har verdien 10.

En `if`-setning består altså av ordet "if", deretter en parentes med et logisk uttrykk inni, samt en *kropp* med krøllparenteser og inni dem alle de kommandoene vi eventuelt ønsker skal kjøres hvis, og bare hvis, det logiske uttrykket er sant. Utrykket `antallKlikk == 10` vil være sant hvis `antallKlikk` er 10 - og usant ellers. (`==` betyr "er lik", men som sammenligning. Ett likhetstege er *tilordning* mens to likhetstege er *sammenligning*.)

Variabelen `antallKlikk` begynner med en verdi på 0, så første gang vi trykker på knappen vil den få verdien 1. Da vil det logisk uttrykket `antallKlikk == 10` ikke være sant, og ingen tekst blir vist. Først den tiende gangen blir uttrykket sant og teksten vises.

Ferdighet 18: Valgsetninger: sammenligne tall med if og else

Vi fortsetter å se på det samme eksemplet. Den ellevte gangen og alle gangene deretter slår ikke if-setningen til, men teksten står likevel på skjermen fra det tiende klikket. Derfor kan det være lurt at vi hver gang blunker teksten hvis if-setningen ikke slår til. Det kan vi få til med en *else-gren*:

```
antallKlikk++;
if( antallKlikk == 10 ) {
    infoDiv.innerHTML = 'Du har klikket 10 ganger.';
} else {
    infoDiv.innerHTML = '';
}
```

Hver gang vi klikker, vil nå alltid en av de to `infoDiv.innerHTML`-kommandoene kjøres. Da er det kun etter det tiende klikket at teksten vises. Etter det ellevte klikket (og alle deretter - og alle før det tiende) vil else-grenen slå til og blank tekst vises.

Når vi har en if-setning med else-gren, vil *alltid* en av grenene utføres.

Som en liten digresjon bør det nevnes at i akkurat dette eksemplet kunne vi fått til det samme uten en else-gren. Vi kunne nemlig blanket teksten før if-setningen:

```
antallKlikk++;
infoDiv.innerHTML = '';
if( antallKlikk == 10 ) {
    infoDiv.innerHTML = 'Du har klikket 10 ganger.';
}
```

På det tiende klikket vil da teksten først blankes og så deretter settes til "Du har klikket 10 ganger.". Alle andre ganger vil den kun blankes.

Hittil har vi bare sammenlignet om tallene er like eller ikke. Vi kan også sammenligne med større enn og mindre enn - samt ulik, som er akkurat det motsatte av å være like:

Uttrykk	Betydning
<code>a < 10</code>	a er mindre enn 10, altså opp til og med 9
<code>a <= 10</code>	a er mindre enn eller lik 10, altså opp til og med 10
<code>a > 10</code>	a er større enn 10, altså fra og med 11
<code>a >= 10</code>	a er større enn eller lik 10, altså fra og med 10

Uttrykk	Betydning
a != 10	a er ikke lik 10

Merk at tabellen over forutsetter at vi jobber med *hele* tall og ikke desimaltall!

Dette kan vi for eksempel bruke til å gi en tilbakemelding på om brukeren har klikket mindre enn 10 ganger eller ikke:

```
antallKlikk++;
if( antallKlikk < 10 ) {
    infoDiv.innerHTML = 'Du har klikket 9 ganger eller mindre.';
} else {
    infoDiv.innerHTML = 'Du har klikket 10 ganger eller mer.';
}
```

Merk else-grenen kjøres når det *motsatte* kriteriet er tilfellet. Det motsatte av `a == b` er `a != b`, og det motsatte av `a < b` er `a >= b`. Dermed kan vi omskrive forrige eksempel til en variant som ser annerledes ut men som oppfører seg helt likt:

```
antallKlikk++;
if( antallKlikk >= 10 ) {
    infoDiv.innerHTML = 'Du har klikket 10 ganger eller mer.';
} else {
    infoDiv.innerHTML = 'Du har klikket 9 ganger eller mindre.';
}
```

⌚ Ferdighet 19: Valgsetninger: sammenligne tall med if, else og else if

En if-setning kan ha maksimalt én if-gren og én else-gren, men vi kan ha så ingen, én eller mange else if-grener.

⌚ Ferdighet 20: Valgsetninger: sammenligne tekster

⌚ Ferdighet 21: Valgsetninger: sammenligne objekter

⌚ Ferdighet 22: Valgsetninger: sammenligne flere ting på en gang (og og eller)

⌚ Ferdighet 23: Funksjoner med parametere

Ferdighet 24: Funksjoner med returverdi

Served by [MarkServ](#) | PID: 18716