

# **Webprogrammering, supplerende forløb**

## **html, css & javascript**

## **webapplikation via Github pages**

**FORLØB-OVERSIGT &**

**OPSÆTNING af Visual studio code**

## OVERSIGT OVER ELEMENTER I FORLØBET: *Webprogrammering, et supplerende forløb*

- Opsætning af **Visual studio code** samt live server
- **DOM-træet & HTML** som markup-sprog
- De primære tags: **<html>, <head>, <body>** - parent & children-noder i træet
- **Inline & block-elementer** – tager elementet en blok horisontalt ell. lægger de sig efter hinanden
- **Attributter på HTML-elementer** - fx "width" på canvas, "id" på et div-element
  
- **CSS-style, placering af css-regler:** Inline på elementet, internal i style & external via ekstern fil
- At give **id eller class på elementer** – og fange dem via CSS: id="minDims", class="mineDimser"
- **Box-modellen** til html-elementer: border, padding & margin til definition af luft i og imellem
- **Display** – via faste eller relative værdier og som block, inline, hidden eller none
- **Display** – static som default eller fixed, relative & absolute
- **CSS-grid** som display-model –
- grid opsat via kolonner med faste værdier, fx grid-template-columns: 100px 400px 100 px;
- Grid opsat med via navne, fx: grid-template-areas: 'header header header' 'menu main main'
  
- **Intro til Javascript** – placering af js-script & inspektør i fx google Chrome
- Om **dynamiske variable i js** og om typesystemet i js
- At **interagere med DOM-træet via js**, og lave CRUD-operationer på det
- **Event-håndtering** via js – fx onclick-event
- Intro til **p5.js-biblioteket** samt til **projektskabelon** til selve opgaven ☺

## TRIN 0:

- Sæt visual studio code op: <https://code.visualstudio.com/>
- Kør VS Code
- Klik på **extensions** i menuen yderst til venstre
- Installer extension **Live server** – søg den frem og klik installér
- Download evt. også ‘material icon theme’ – det giver filerne ikoner (ikke nødvendigt)
- Opret en **ny fil** i et tomt vs code-vindue
- Giv den navn, fx **index.html** – opret i samme hug **ny mappe** til øvelserne, fx **webProgBasis**
- Skriv **doc** i html-dokumentet og **klik enter** – det giver basis-koden til et html-dokument

## **LÆRINGSMÅL:** Webprogrammering, et forløb

### **NØGLESPØRGSMÅL SOM FORLØBET DÆKKER**

- Hvad kendetegner et markup-sprog til brug i en GUI?
- Hvad er styrken ved at organisere grænsefladens elementer i træstruktur?
- Hvad er forskellen på et sprog baseret på statiske og dynamiske typer? (javascript vs java)

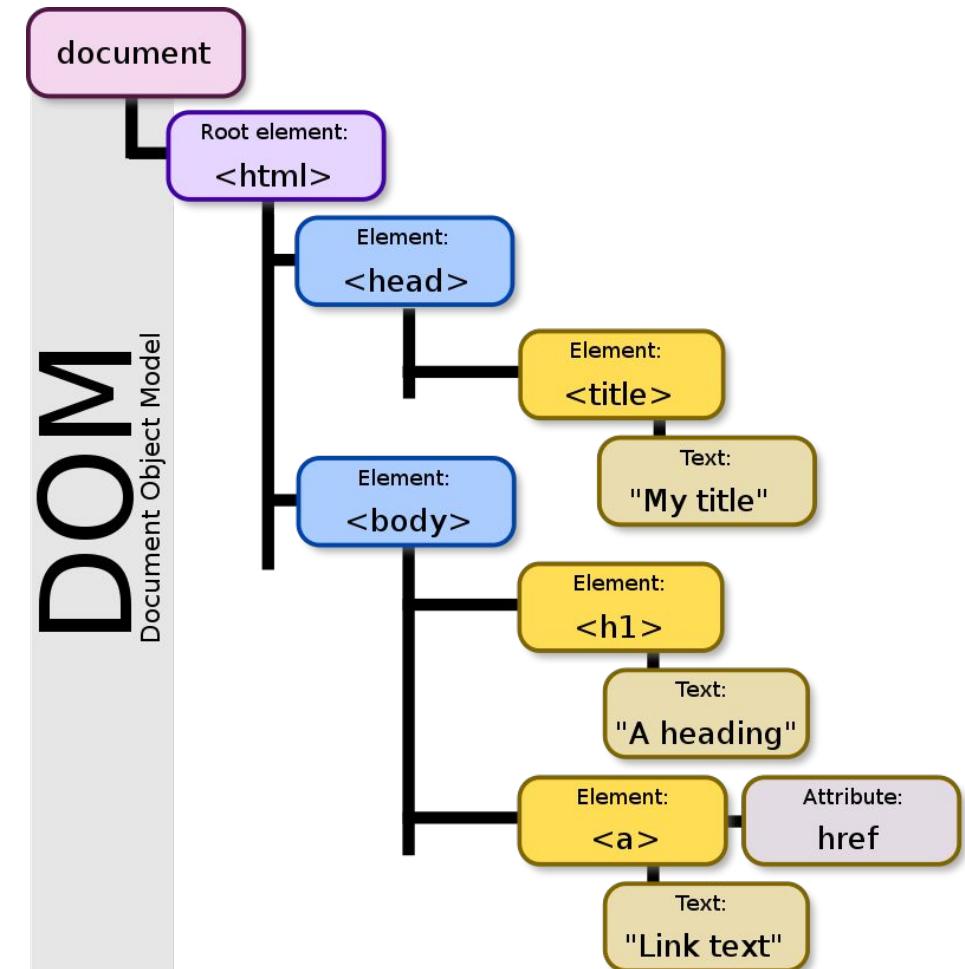
# **HTML: DOM – document object model**

**TRIN 1: Forstå et markup-sprog til opsætning af en grafisk brugergrænseflade**, en GUI, her HTML.  
Forstå først DOM-modellen og herunder diverse basale HTML-tags som fx div, head, body osv., og videre selektioner som ID og CLASS. Og forstå nestede elementer, hvor fx en div lægges i en div.

Via et **markup-language** har man en standard til opsætning & visning af elementer i et dokument, mens et rigtigt **programmeringssprog** kan kontrollere programmets gang, lave udregninger, ændre data dynamisk osv.

Via C# kan bruge XAML til desktop-programmert (wpf-frameworket)

Via Java kan man bruge FXML også til desktop-programmer (javaFX-frameworket)



# HTML: DOM – document object model

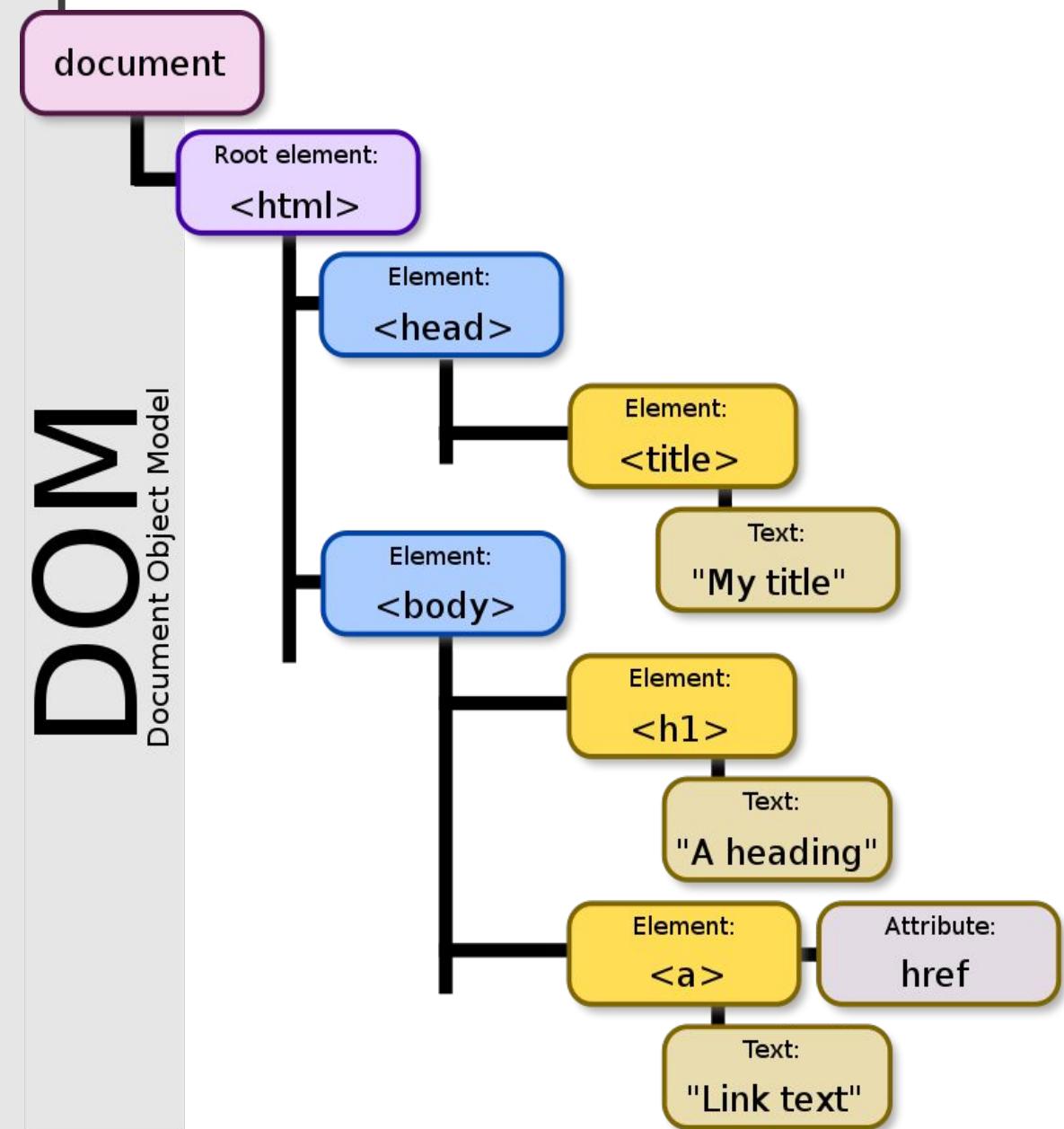
DOM repræsenterer HTML-dokumentet som en **hierarkisk træstruktur**.

Denne struktur gør det muligt via kode at tilgå og ændre i HTML-elementerne i træet – fx afhængigt af brugerinput eller andre begivenheder.

**Dokumentets rod-objekt** er hele html-siden, som igen indeholder to børn, head og body, og disse indeholder efter diverse børn.

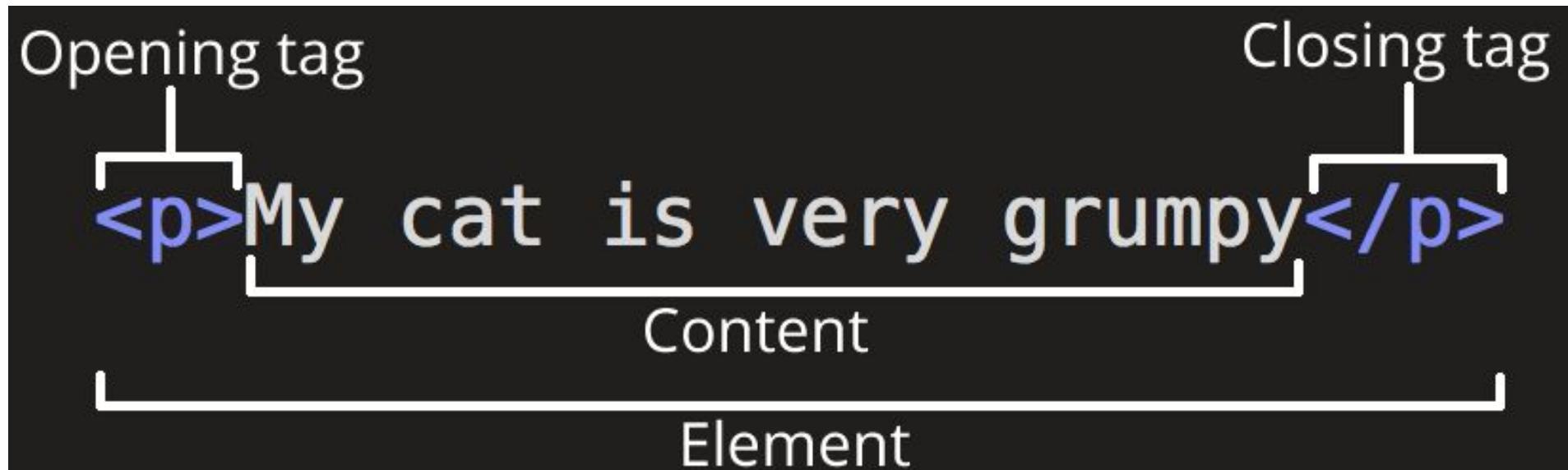
**Body** indeholder i dette tilfælde en **<h1>**, heading med værdien “A heading” samt et link-tag **<a>** med en “href”, som refererer til en adresse på www samt en værdi “Link text”, som ender med at udgøre det klikbare link.

**Head** indeholder elementet **<title>** med værdien “My title”, og dette udgør sidens titel i browseren.



## HTML: DOM – document object model

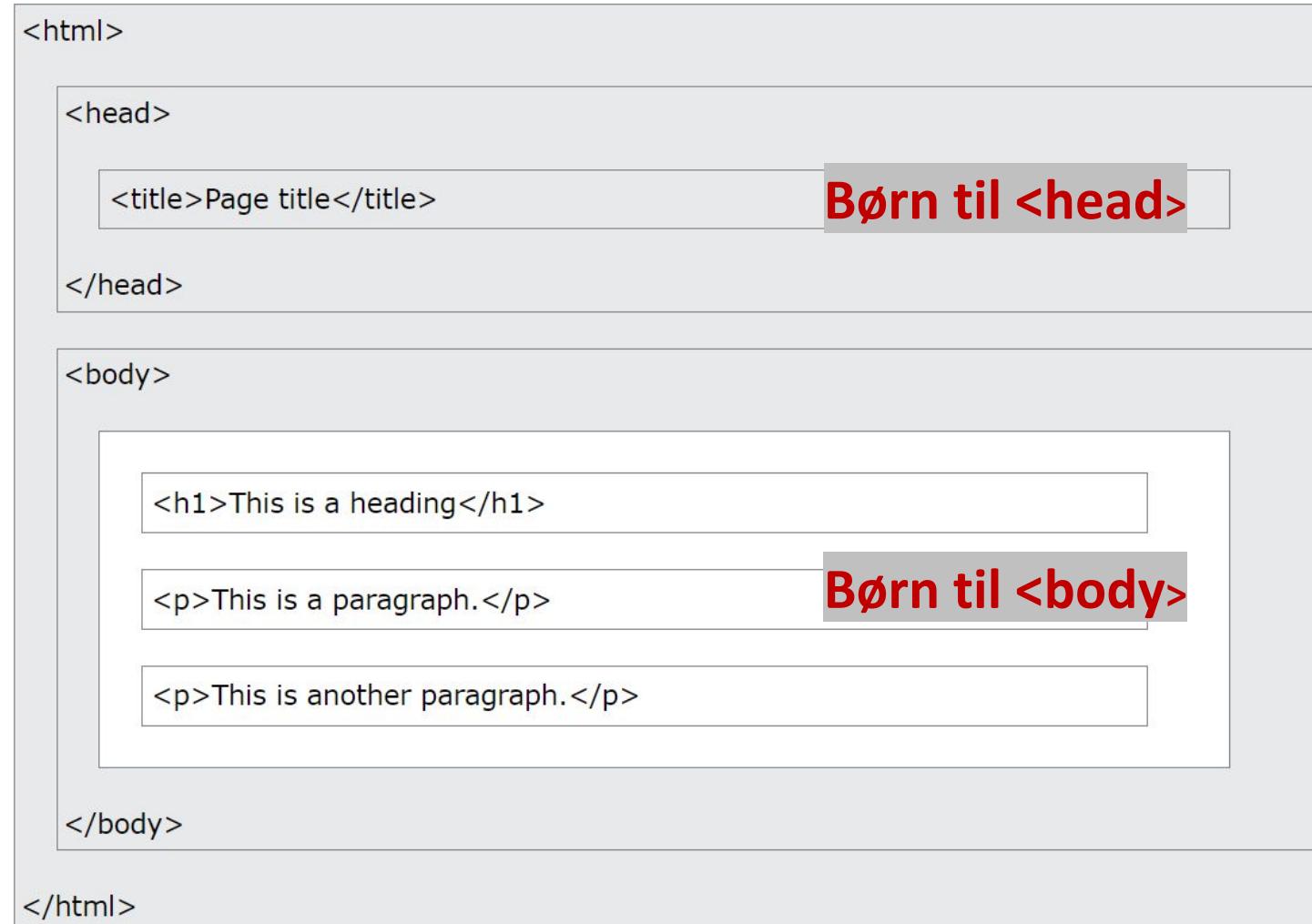
```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <p>My cat is very grumpy</p>
  </body>
</html>
```



## HTML: DOM – document object model

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Page Title</title>  
  </head>
```

```
  <body>  
    <h1>This is a heading</h1>  
    <p>This is a paragraph</p>  
    <p>This is another paragraph</p>  
  </body>  
</html>
```



// html-tagget er rod-objektet med to hovedbørn, `<head>` og `<body>`, med egne børn  
// Øverst fortæller `<!DOCTYPE html>` at det er HTML5-dokument til browseren

## HTML: DOM – document object model

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title></title>  
  </head>  
  
  <body>  
  </body>  
</html>
```

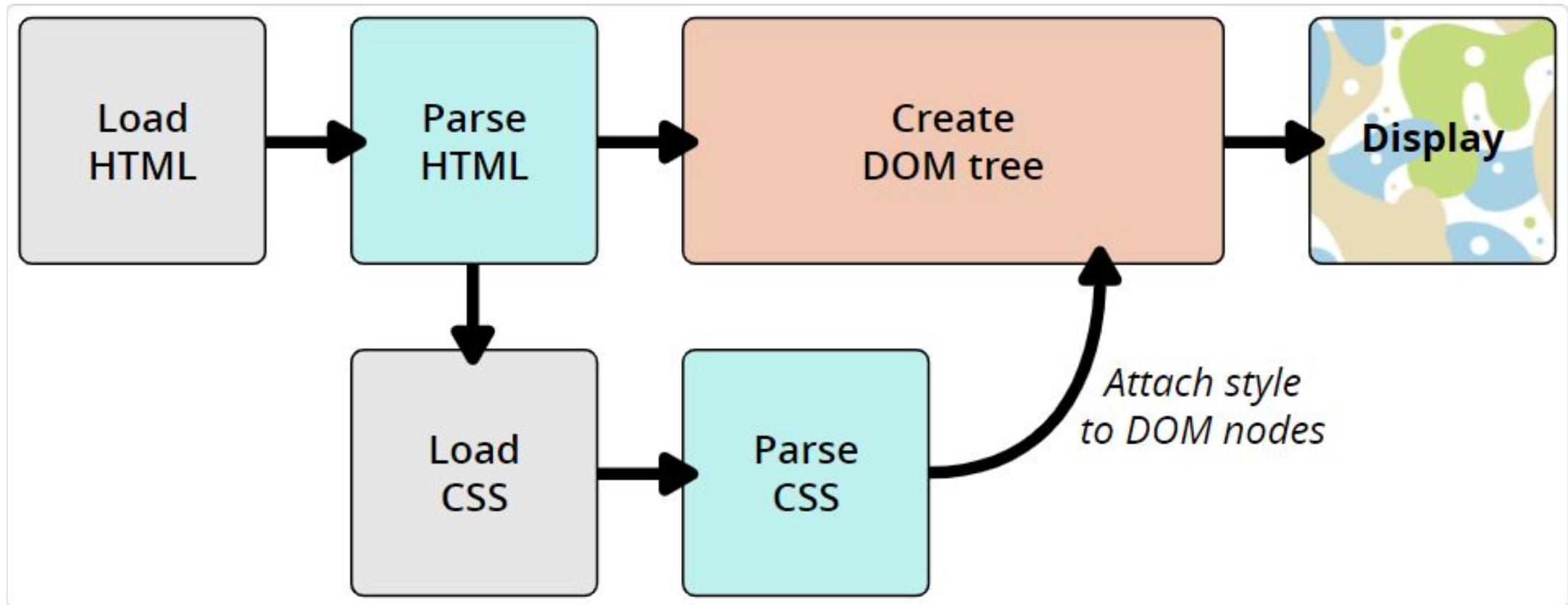
**Giv dokumentet en titel!**

**Indsæt html-elementer i body-tagget:**  
Fx h1, h2, p osv.

[https://www.w3schools.com/html/html\\_elements.asp](https://www.w3schools.com/html/html_elements.asp)

# HTML: DOM – document object model

Træstrukturen bliver dannet af browseren, og elementerne i den bliver stiliseret via CSS – og endelig vist på skærm



## **HTML: <head> og <body>**

**<head> tagget** indeholder info om html-dokumentet, men ikke selve indholdet  
Heri angiver man dokumentets titel via **<titel>**, link til diverse filer og ikoner via **<link>**  
samt evt. javascript-kode samt fx beskrivelser og nøgleord til søgemaskiner.

**<body> tagget** indeholder den synlige info til brugeren, via forskellige elementer, fx:

**<img>** - billeder

**<a>** - link

**<p>** - tekst-paragraf

**<ol>** - en ordnet liste med **<li>** list-items

**<table>** - en tabel

**<form>** - fx brugerinput i tekstboks via **<input type= "text">**

**<canvas>** til fx animationer eller spil programmeret via javascript

**<button>** til knapper (øvrige elementer kan også tildeles knap-funktion)

**<div>** en inddeling, division, af siden som igen kan indeholde øvrige elementer

**<!--...-->** man kan lægge kommentarer ind i koden

# **HTML: html-elementernes attributter & block-elementer vs inline-elementer**

Elementer inden for `<body>` tagget kan tildeles attributter/værdier, fx:

**En paragraf gøres gul via style:**

```
<p style="color: yellow;"> Denne tekst er nu gul via style </p>
```

**Et canvas gives en størrelse, på 500 x 200 pixel og via style en sort kant på 5 pixel:**

```
<canvas width="500" height="200" style="border:5px solid #000000;"></canvas>
```

**Et image får angivet sin specifikke billedfil via src samt bredde og højde:**

```

```

Visse elementer lægger sig i udgangspunktet på en ny linje, det er **block-elementer**:

`<p>` paragraf tager fx ny linje, det samme gør en `<div>`, en `<table>` og et `<canvas>` osv.

Andre elementer fortsætter så vidt muligt på samme linje, det er **inline-elementer**:

`<img>` billede fortsætter på samme linje, ligeledes med fx `<button>`, `<input>` osv.

# **HTML: html-elementernes attributter & block-elementer vs inline-elementer**

Elementer inden for `<body>` tagget kan tildeles attributter/værdier, fx:

## **Tilføj et canvas i jeres html-dokument:**

Et `canvas` gives størrelse, på 500 x 200 pixel og via `style` en sort kant på 5 pixel:

```
<canvas width="500" height="200" style="border:5px solid #000000;"></canvas>
```

## **Tilføj også et billede i html-dokumentet:**

Et `image` får angivet sin specifikke billedfil via `src` samt bredde og højde:

```

```

Visse elementer lægger sig i udgangspunktet på en **ny linje**, det er **block-elementer**:

`<p>` paragraf tager fx ny linje, det samme gør en `<div>`, en `<table>` og et `<canvas>` osv.

Andre elementer fortsætter så vidt muligt på **samme linje**, det er **inline-elementer**:

`<img>` billede fortsætter på samme linje, ligeledes med fx `<button>`, `<input>` osv.

## HTML: div-element og class & id

<div> benyttes til at lave en division, en inddeling, og den kan fungere som beholder for øvrige elementer – andre <div> eller fx bare <p> paragraf eller <img> billede

```
<div class="beholder">  
  <p>En paragraf lagt ind i en div</p>  
  </img>  
</div>
```

**class** giver html-element et navn, og så kan man tilgå den nemt via Javascript-kode eller stilisere den via CSS. Flere elementer kan høre til samme **class**.

Den samme gælder **id**, fx id="beholder", mens **class**-tagget kan deles af flere html-elementer, er **id** individuelt på det specifikke element.

# **Ikke relevante ting**

## **Noter & links**

## **Modul 1:**

- Opsætning af vs code, live server, tomt html-dokument
- forstå DOM + markup-sprog
- Tilføj egne html-elementer til tomt html-dokument – om head/body, attributter, inline vs. block, div og intro til class & id (som en overgang til css)

## **Modul 2 + 3:**

Intro til css,

Sætte regler til elementer via generelle elementnavn samt via class og id-attributterne

Kort om placering af css-kode/regler – inline, internal eller external

Forstå Borderbox-modellen – tilføj css via borderbox-modellen til egne elementer i html-dokumentet

Width og height-egenskaber, faste & relative – se eksempel og tvist på egne elementer

Display-værdier – inline, block, hidden og none – se eksempel og tvist på egne elementer

Kort om Hierarki/specifitet af en regel – hvilken regel overruler de øvrige?

## **Modul 4:**

Positionering af elementer – statisk, relativ, absolute, fixed osv.

Se eksempel på positionering og brug af class samt id – og prøv det af på eget html-dokument

Opstart på intro til css-grid som layout model til!

## **Modul 5:**

Css-grid fortsat (evt om responsivt grid også)

## **Modul 6: javascript, opstart på!**

Om let, const & var

Om js med dynamiske typer og typesvagt svagtsprog, som ikke behøver eksplisit konvertering

Intro til inspector i Chrome

Placering af js-kode

## **Modul 7: javascript – at manipulere på DOM**

CRUD på elementer via javascript

Kald af js-funktion fra html-button

## **Modul 8: javascript & p5.js**

Import af p5.js + Opsætning af canvas via p5.js

Accelerometer via p5.js samt DOM-metoder lagt ind i p5.js

## **Modul 9: I gang med opgaven om API'er**

Lav et websted til folkeskole-elever, designet til telefonen, som sætter bevægelseslege i gang i klassen

Design af websted tilpasset telefon – gerne via css-grid.

Suppler gerne med jQuery eller Bootstrap

Brug af API'er – til accelerometer via p5.js

## **Modul 17: slut, altså 8 moduler til selve hovedopgaven efter introen + samt lectio-aflevering til projektet**

## **Læringsmål:** Webprogrammering, et forløb

### **TRIN 1: Forstå et markup-sprog til opsætning af en grafisk brugergrænseflade**, en GUI, her HTML.

Forstå først DOM-modellen og herunder diverse basale HTML-tags som fx div, head, body osv., og videre selektioner som ID og CLASS

Og forstå nestede elementer, hvor fx en div lægges i en div.

**Opgave 1:** Lav et statisk websted med en header, en liste af elementer med billede ved siden af samt en footer.

### **TRIN 2: Forstå CSS** - at man kan stilisere på tværs af html-elementer og sider, og at man også kan tilføje animationer dermed, uden at bruge Javascript. Forstå herunder CSS GRID som layoutsystem til at organisere en sides elementer.

Forstå herunder positionering via CSS (absolute, relative, fixed osv.)

Forstå herunder specificitet, at fx stilisering på ID overruller CLASS, altså hierarki i forhold til hvilken stilisering som bestemmer.

**Opgave 2A:** Lav først css-script som stiliserer på en Class af html-elementer og en hover-effekt på et html-element.

**Opgave 2B:** Lav et layout via CSS-grid – gör evt. layoutet responsivt i forhold til brugerens skærmstørrelse

### **TRIN 3: Forstå basale principper i klient-server-arkitektur** og begreberne front-end og backend samt statiske og dynamiske sider.

**Opgave 3:** Læs kapitel om protokolstakken fra bogen, Datalogi.

## Læringsmål: Webprogrammering, et forløb

**TRIN 4: Forstå basal brug af Javascript** på HTML-elementer og direkte i et Canvas-element via p5.js som bibliotek.

Forstå herunder prototype-baseret klassesystem i JS versus Java's OOP.

[https://en.wikipedia.org/wiki/Prototype-based\\_programming](https://en.wikipedia.org/wiki/Prototype-based_programming)

Forstå query til DOM-træet – CRUD på html-elementer

**Opgave 4:** Lav et canvas i jeres CSS-GRID layout fra opgave 2, og tilføj animationer deri.

**TRIN 5:** Forstå basale HTML-operationer til backend: get, put osv.

**TRIN 6:** Forstå mulighederne for udviklere i Chromes konsol – se "kildekode" ved højreklik samt

<https://developer.chrome.com/docs/devtools/>

Skriv til console.log("TEST")

7: Kort intro til frameworks og biblioteker som jQuery, Bootstrap eller Vue.js

8: ? Backend-kodning, server-side scripts? Lidt php? DB?

FireBase!

## **HTML: semantiske elementer vs. div**

### **Fra Kodning til nettet, om semantiske elementer som Main, Header, Footer osv.**

Websider skal derfor udtænkes til *både* at være nemme og intuitive for mennesker at bruge, men *også* at være logiske og semantisk korrekt udformede til læse- og søgemaskiner og andre automatiserede tilgange til siderne.

Semantik drejer sig om, at du ved at bruge de forskellige tags giver udtryk for, hvordan du mener, at din side skal opfattes. Du beskriver hvilken del, der er overskrift, hvilken del, der indeholder hovedemnet, hvilken der står for navigationen, du beskriver hvor 'footer' er og hvor dine billeder befinner sig. Når du bruger de nye elementer, skaber du sidens indholdsfortegnelse. Alt dette kan du ikke gøre med 'div', da de ikke adskiller sig fra hinanden - <div> er en massebetegnelse.

# HTML: RWD, responsivt webdesign

Responsiv html-kode: max-width, textunit vw

[https://www.w3schools.com/html/html\\_responsive.asp](https://www.w3schools.com/html/html_responsive.asp)

Når du skal tilpasse koden alle skærmstørrelser, skal dit design være *fleksibelt*, og det betyder, at du skal til at bruge *relative* bredder som em og % i stedet for pixels

Medie queries

[https://www.w3schools.com/css/css3\\_mediaqueries\\_ex.asp](https://www.w3schools.com/css/css3_mediaqueries_ex.asp)

Set the viewport: In the head section of your HTML document, add the following code to set the viewport:

html

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

This tells the browser to set the width of the page to the width of the device's screen and to set the initial zoom level to 1.0

[https://developer.mozilla.org/en-US/docs/Web/CSS/Viewport\\_concepts#mobile\\_viewports](https://developer.mozilla.org/en-US/docs/Web/CSS/Viewport_concepts#mobile_viewports)

```
/* Styles for devices with a screen width of 600 pixels or less */
```

```
@media only screen and (max-width: 600px) {
```

```
    /* Styles for the body element */
```

```
    body {
```

```
        font-size: 16px;
```

```
}
```

```
.container {
```

```
    width: 100%;
```

```
    margin: 0;
```

```
}
```

```
} @media only screen and (min-width: 601px) {
```

```
    /* Styles for the body element */
```

## **HTML: Chrome inspector – at undersøge koden via browseren som redskab**

Tjek koden på <https://validator.w3.org/>

[https://developer.mozilla.org/en-US/docs/Learn/Common\\_questions/Tools\\_and\\_setup/What\\_are\\_browser\\_developer\\_tools](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Tools_and_setup/What_are_browser_developer_tools)

## HTML: Øvrige ting

At indlejre html-kode i anden html-kode via – Objekt eller Iframe

```
<iframe src="other-page.html" width="100%" height="500"></iframe>
```

```
<object data="other-page.html" width="100%" height="500"></object>
```

Eller via et serverside-script som php, <?php include 'other-page.html'; ?>

Hvis reserverede keywords i en tekst, brug: &entity\_name; eller &#entity\_number;

```
<head> <meta charset="UTF-8"> <title>My Web Page</title> </head> // dette er dog default i html5
```

Web storage vs cookies? Locale storage

**App Cache, hvis offline, ressourcer gemt i browseren**

Units, [https://www.w3schools.com/cssref/css\\_units.php](https://www.w3schools.com/cssref/css_units.php)

Absolute og relative (pixel er absolut, men relativ til pixel-dybden på brugerskærmen!)

Html-forms til input med mere: [https://www.w3schools.com/html/html\\_forms.asp](https://www.w3schools.com/html/html_forms.asp)

Video med styling af input-felter og type samt diverse attributter og submit-knap

[https://www.nemprogrammering.dk/Tutorials/HTML5/form\\_html5.php](https://www.nemprogrammering.dk/Tutorials/HTML5/form_html5.php)

[https://developer.mozilla.org/en-US/docs/Learn/Forms/How\\_to\\_structure\\_a\\_web\\_form](https://developer.mozilla.org/en-US/docs/Learn/Forms/How_to_structure_a_web_form)

[https://developer.mozilla.org/en-US/docs/Learn/Forms/HTML5\\_input\\_types](https://developer.mozilla.org/en-US/docs/Learn/Forms/HTML5_input_types)

Serverside events (via php-script): [https://www.w3schools.com/html/html5\\_serversidevents.asp](https://www.w3schools.com/html/html5_serversidevents.asp)

http, request metoder – put, get, post osv.

Webperformance: <https://developer.mozilla.org/en-US/docs/Learn/Performance>

Semantik-tags i html5, video om: [https://www.nemprogrammering.dk/Tutorials/HTML5/HTML5\\_semantik\\_struktur.php](https://www.nemprogrammering.dk/Tutorials/HTML5/HTML5_semantik_struktur.php)

At bruge dem for at gøre det mere overskueligt for udvikler og mere håndterbart for søgemaskiner at uddrage indhold

# CSS: Responsivt design, RWD, via CSS Grid

Video: <https://www.youtube.com/watch?v=sKFW3wek21Q>

Se også artikel:

<https://travishorn.com/responsive-grid-in-2-minutes-with-css-grid-layout-4842a41420fe>

<https://css-tricks.com/look-ma-no-media-queries-responsive-layouts-using-css-grid/>

// at justere griddets layout ud fra skærmstørrelse

```
@media only screen and (max-width: 500px) {  
    .item1 { grid-area: 1 / span 3 / 2 / 4; }  
    .item2 { grid-area: 3 / 3 / 4 / 4; }  
    .item3 { grid-area: 2 / 1 / 3 / 2; }  
    .item4 { grid-area: 2 / 2 / span 2 / 3; }  
    .item5 { grid-area: 3 / 1 / 4 / 2; }  
    .item6 { grid-area: 2 / 3 / 3 / 4; }  
}
```

## CSS: INTRO TIL BOOTSTRAP 5

**Bootstrap** er et populært **front-end framework** til at lave hjemmesider og webapplikationer. Det tilbyder nogle allerede byggede komponenter – **html, css & js**. Dets styrke er responsivt og visuelt, æstetisk design samt hurtighed.

- Et gridsystem til layout
- Komponenter som fx. knapper, input-former, dropdown-menuer, navigationsmenu, billed-karusseller osv.
- Responsivt i forhold til bruger-skærmens størrelse
- Man kan tilpasse Bootstraps default-design

*Se eksempel 1 & 2!*



## Ressourcer til Bootstrap

Kom i gang med Bootstrap: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>

Dokumentation af standard-komponenter: <https://getbootstrap.com/docs/5.3/components/navbar/>

Eksempler på Bootstrap-komponenter: <https://getbootstrap.com/docs/5.3/examples/>

- Downloades og lægges ind i mappe

God video til brug af Bootstrap: <https://www.youtube.com/watch?v=-qfEOE4vtxE>

W3-schools: [https://www.w3schools.com/bootstrap5/bootstrap\\_get\\_started.php](https://www.w3schools.com/bootstrap5/bootstrap_get_started.php)

- Eksempler på en masse forskellige bootstrap-elementer

## Tilpasser Bootstraps standard-elementer:

So while it's not strictly necessary to use Sass with Bootstrap, it can make the process of customizing and extending Bootstrap's styles much more efficient and streamlined

Customize bootstraps komponenter:

You can override Bootstrap's default button styles by defining your own CSS rules that target the appropriate button classes. For example, to change the color of the primary button, you can define a new color rule for the .btn-primary class:

css

```
.btn-primary { background-color: #your-new-color; border-color: #your-new-color; }
```

2. Include your custom CSS:

3. Include your custom CSS file in your HTML document after Bootstrap's CSS file:

html

```
<head> <link rel="stylesheet" href="path/to/bootstrap.css"> <link rel="stylesheet" href="path/to/custom.css"> </head>
```

3. Ensure that your custom styles are more specific than Bootstrap's default styles: Make sure that your custom CSS rules are more specific than Bootstrap's default styles to ensure that they are applied. You can do this by using a more specific selector or by using the !important keyword. However, using !important is generally considered bad practice and should be used sparingly.

# CSS: Øvrige muligheder

## Via CSS

- mask af billeder
- Hover-effekter og pseudo-klasser
- Opdele div i kolonner
- Transformationer, overgange og animationer
- variable i CSS – kan tilgås fra javascript og kan være globale via ::root

[https://www.w3schools.com/css/css3\\_variables.asp](https://www.w3schools.com/css/css3_variables.asp)

Sass – tilføjelse til css

[https://www.w3schools.com/sass/sass\\_intro.php](https://www.w3schools.com/sass/sass_intro.php)

Teste websted på tværs af browsere:

<https://www.browserstack.com/>

# **CSS: Stilisering af html-elementer**

**- at definere regler for html-træet**

# CSS: At stilisere html-elementerne via – Cascading Style Sheets, et regelsæt

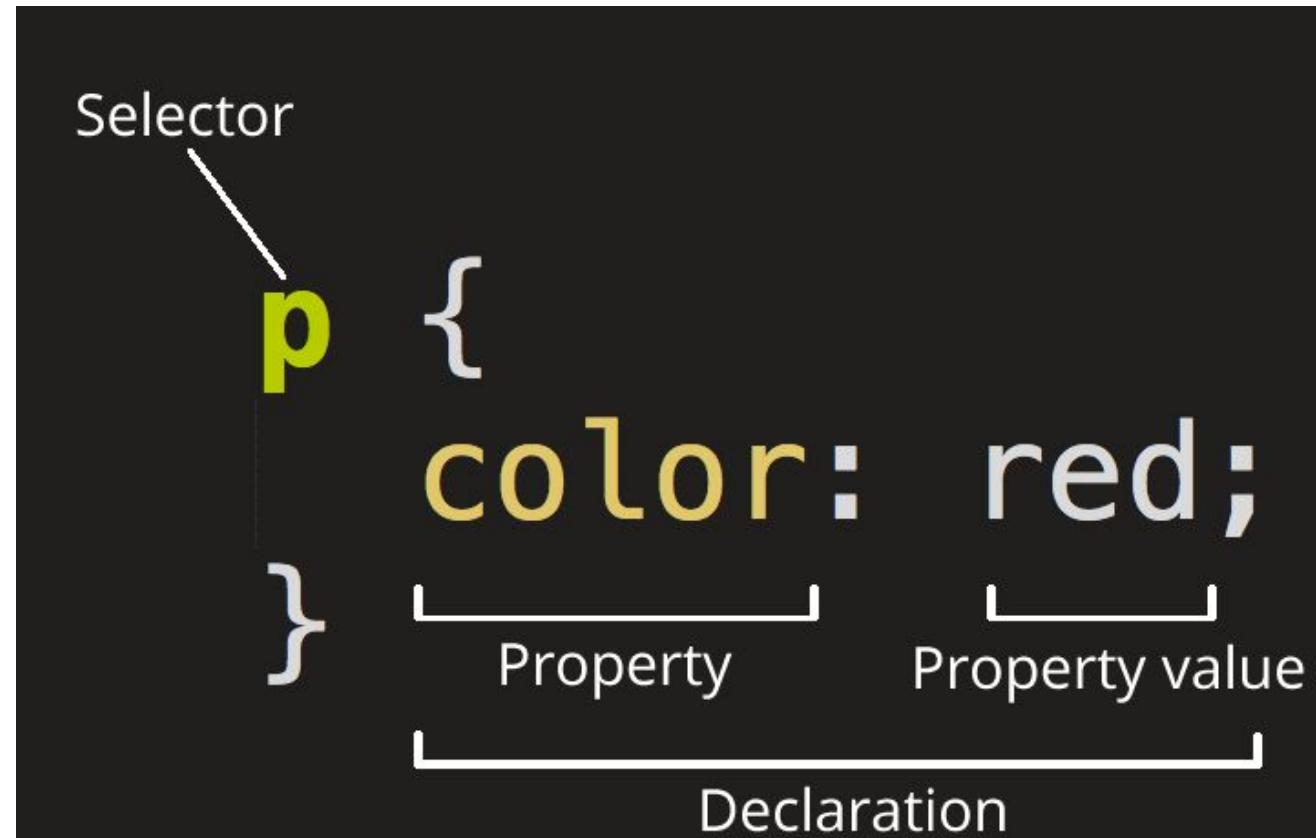
Med CSS-scripts kan vi stilisere elementerne – bestemme hvordan de ser ud og vises på brugerens skærm:  
CSS udgør et regelsæt for HTML-træet!

Her sættes alle <p>-elementerne, altså paragraffer, til farven rød:

P angiver elementet og inden for scope af de krøllede parenteser gives 'color'-attributten værdien 'red'.

Denne regel for alle paragraphs  
lægges ind i et style-tag:

```
<head>
<style>
    - sæt css-regler ind her!
</style>
</head>
<body>
</body>
```



# CSS: Placing af script – 3 måder

1: INTERNAL, CSS-scripts kan ligge inden for `<head>`-tagget via `<style>`

```
<head>
<style>
  p{border: 10px solid red;}
</style>
</head>
```

2: INLINE, CSS-scripts kan ligge inde i selve html-elementet på denne måde:

```
<p style = "border: 10px solid red;">Dette er en glad paragraf</p>
```

3: EXTERNAL, Og CSS-scripts kan ligge i selvstændig CSS-fil, som man loader ind i `<head>` via `<link>` tagget:

```
<head>
  <link rel="stylesheet" href="mitStyleSheet.css">
</head>
```

Løsning 1 og 3 er god praksis,

men 2 er problematisk, da CSS-scripts'ene så ligger spredt rundt i koden...

Scripts som ligge inline på html-tagget overruler både 1 og 2

# CSS: At stilisere html-elementerne – Cascading Style Sheets

Med CSS-scripts kan vi stilisere elementerne – bestemme hvordan de ser ud og vises på brugerens skærm:

Man kan fx definere skrifttype, størrelse, farver, margener, layout-logik, men også hover-effekt og små animationer!

```
<head> <style>
  body{
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  }
  #overskrift{
    color: blue;
    border: 3px solid rgb(49, 45, 3);
    padding: 10px;
    width: 50%;
  }
  .minParagraf{
    color: rgb(109, 133, 15);
    font-size: 2vw;
    border: 3px solid rgb(48, 47, 44);
    padding: 10px;
    width: 75%;
  }
  .minParagraf:hover{
    color: rgb(0, 0, 0);
  }
</style> </head> <body>
<h2 id="overskrift">CSS definerer stil og layout</h2>
<p class="minParagraf">Denne paragraf hører til klassen minParagraf</p>
<!-- Først element får id "overskrift", og næste element får class "minParagraf"
Dette kaldes style-scriptet via #overskrift og .minParagraf. Og så får de tildelt værdier!--&gt;
&lt;/body&gt;&lt;/html&gt;</pre>
```

Se dokument på modulet:  
cssEksempel.html

Sæt regel for baggrundsfarven for  
klassen .minParagraf  
Brug: background-color-attribut!

Sæt også baggrundsfarve-regel for  
hele body!

# CSS: Box-modellen – at styre elementets kant, margin mm.

Et html-element defineres af box-modellen

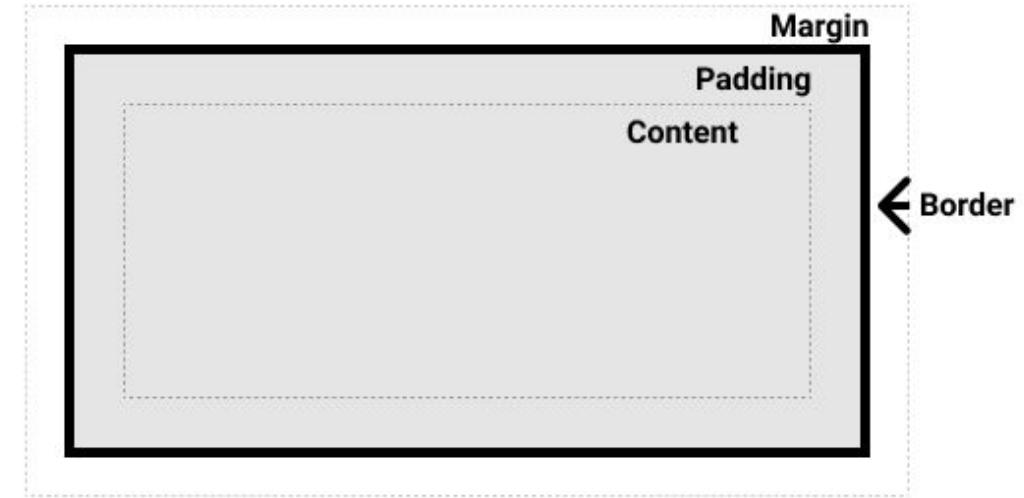
**Margin** – elementets afstand til øvrige elementer

**Border** – kantens størrelse og farve og type

**Padding** – den indre afstand fra kant til selve indholdet

Via CSS kan man definere dette, fx:

```
<head>
<style>
p {
    border-radius: 30px;
    border: 10px solid darkblue;
    padding: 10px;
    margin: 20px;
}
</style>
</head>
```



Box modellen er snedig :)

**Tilføj css til dit html-dokument via  
box-modellen: border, margin, padding**

// hvis elementets indhold fx er 100px, så  
er den samlede længde:

**$100\text{px} + 10\text{px} + 10\text{px} + 20\text{ px} = 140\text{ px}$  i alt**

# CSS: At sætte højde & bredde af element – via længde eller via % tilpasset til vinduet

## Via CSS kan man sætte bredde & højde til fast værdi

- Via fx px for pixels eller cm for cencimeter

## Eller til en relativ værdi i procent

- Det sætter højden til en procent af parent i html-træet
- Til vinduet, hvis elementet ikke er indlejret i noget...
- Det tilpasser dermed elementet til den konkrete skærmstørrelse – og det er smart!

Man kan også sætte max- og min-værdier!

1: Se dokument på modulet, [cssEksempel3.html](#)

2: Giv egne html-elementer et tvist via css, width osv.

```
<head>
<style>
#overskrift{
    color: blue;
    border: 3px solid rgb(49, 45, 3);
    padding: 10px;
    width: 800px;
}
.minParagraf{
    color: rgb(109, 133, 15);
    font-size: 2vw;
    border: 3px solid rgb(48, 47, 44);
    padding: 10px;
    width: 10%;
    max-height: 100px;
}
</style>
</head>
```

# CSS: Display-værdier – default & ændring af default!

Via CSS kan vi også styre visning og layout: Første trin er display-værdien

Se program-eksempel med:

```
display: block;      // elementet er nu block. Tager ny linje  
display: inline;    // elementet er nu inline. Vises på samme linje  
display: none;     // tager nu ikke plads, og vises ikke  
visibility: hidden; // tager nu plads, men vises ikke
```

Derudover kan man også sætte den stak-orden for visningen, som et element har via fx:

**z-index: 1;** eller  
**z-index: -1;**

PS:

Et element skal dog have givet en position-værdi for at z-index fungerer...

Se dokument på modulet:  
cssEksempel4.html

**1: Slet fx inline-regel for element**  
**2: Giv jer es egne elementer display-regel!**

# HTML: positionering & layout – statisk, relativt, absolut og fixed

Elementerne får i udgangspunktet position på skærmen **statisch**, oppefra og ned, i den rækkefølge man har skrevet det i.

Man kan også give elementerne en **relativ position** via CSS, hvor den nye position er relativ til den statiske position, som følger dokumentets flow.

Og man kan give dem en **absolut position** i forhold dens nærmeste parent i træstrukturen. Andre elementers position bliver ikke påvirket af et element med absolut position; det tæller ikke i dokumentets flow.

**Fixed position** tager også et element ud af dokumentets flow, og det er kun relativt til viewporten/vinduet (hvis brugeren fx ruller ned, scroller elementet bare med).

1: Se eks. på modulet, "**positionEksempel.html**" 😊

2: Tilføj html-elementer til eget html-dokument

- Brug de forskellige positionerings-regler!

Se: [https://www.w3schools.com/css/css\\_positioning.asp](https://www.w3schools.com/css/css_positioning.asp)

```
// 3 % margen-afstand til toppen  
// 5 % margen-afstand til højre side  
#AndersAnd{  
    position: absolute;  
    top: 3%;  
    right: 5%;  
}
```

## CSS: Specificitet – rangorden når man sætter regel for element

Når man sætter en regel for *samme* element flere steder,  
hvilken regel skal så bestemme elementets attributter?

**Hierarkiet:** Nedenfor kan man se, hvilken selektion som er mest vigtig i CSS:

At definere inline i selve html-elementet  
overruler de øvrige,  
mens fx id-selektion overruler class osv.

1. Inline styles,
2. `<div style="color: red;"`
2. Ids  
`#example`
3. Classes, pseudo-classes, and attribute  
selectors  
`.example, .example: hover, [type="text"]`
4. Element selectors and pseudo-elements  
`p, div, h1, ::before, ::after`

# CSS: Display via GRID

Via CSS kan vi også styre visning og layout: Vi skal kigge på Grid som layout-system

Grid opdeler sjovt nok skærmen i et grid: Kolonner & rækker

```
.minContainer {  
    display: grid;  
}
```

Man sætter en container-div til at fungere som et grid

Og deri kan man lægge html-elementer, hvis position og fylde man så kan jonglere rundt med i layout

```
<div class="minContainer">  
    <p>Denne paragraf er første dims</p>  
    <p>Denne paragraf er den anden</p>  
    <p>Denne paragraf er den tredie</p>  
    <p>Denne paragraf er den fjerde dims</p>  
</div>
```

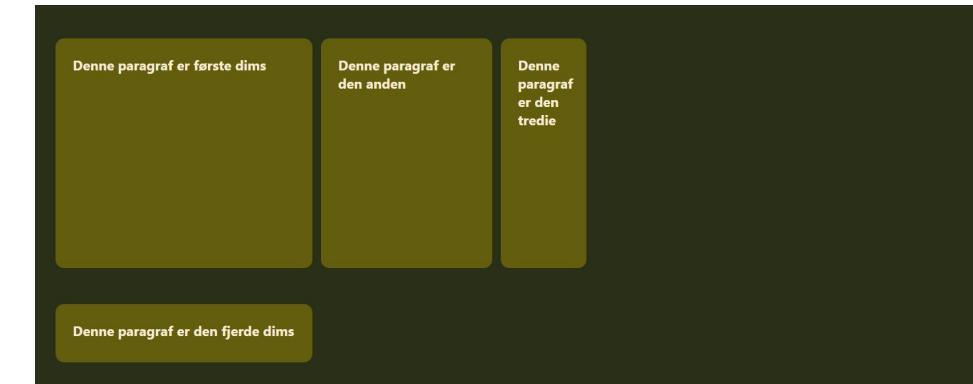
// man kan definere antal kolonner og deres fylde

//og det samme med rækkerne:

// 3 kolonner i forskellig størrelse samt 2 rækker også i forskellig størrelse:

```
grid-template-columns: 300px 200px 100px;  
grid-template-rows: 300px 100px ;
```

OPGAVE – se næste slide



# CSS: Display via GRID – mange måder at bruge grid på, men se eksempel og link

## OPGAVE:

Tilføj grid til jeres html-dokument, se eksemplet på modulet,  
[cssEksempel5.html](#)

<https://css-tricks.com/snippets/css/complete-guide-grid/>



# CSS: Display via GRID – via grid-template-areas

Man give via **grid-template-areas** styre sine elementers layout via **navne**

```
.minContainer {  
    background-color: rgb(42, 47, 23);  
    display: grid;  
    gap: 10px;  
    padding: 30px;  
    grid-template-areas:  
        'header header header header header header'  
        'menu main main main right right'  
        'menu footer footer footer footer footer'  
        'foot foot foot foot extra';  
}  
/* her bruges navnene - 6 gange per kolonne skal følges hele vejen ned*/
```

```
.item1 {grid-area: header;}  
.item2 {grid-area: menu; }  
.item3 {grid-area: main; }  
.item4 {grid-area: right; }  
.item5 {grid-area: footer; }  
.item6 {grid-area: foot; }  
.item7 {grid-area: extra; }
```

## OPGAVE:

Tilføj endnu et grid til jeres html-dokument,

Se igen eksempel på modulet,  
[cssEksempel6.html](#)



# CSS: Display via GRID – ANDRE TEKNIKKER: unavngivne elementer via prikker

Man give via **grid-template-areas** styre sine elementers layout via **navne**

- og repræsentere unavngivne elementer med prikker

```
.minContainer {
```

```
    grid-template-areas:
```

```
        /*header-elementet fylder to */  
        /*extra-elementet fylder en*/  
        /* prikkerne repræsenterer de unavngivne elementer i griddet */  
        /* bemærk at der SKAL være et mellemrum imellem prikkerne*/  
        'header header . . extra';
```

```
}
```

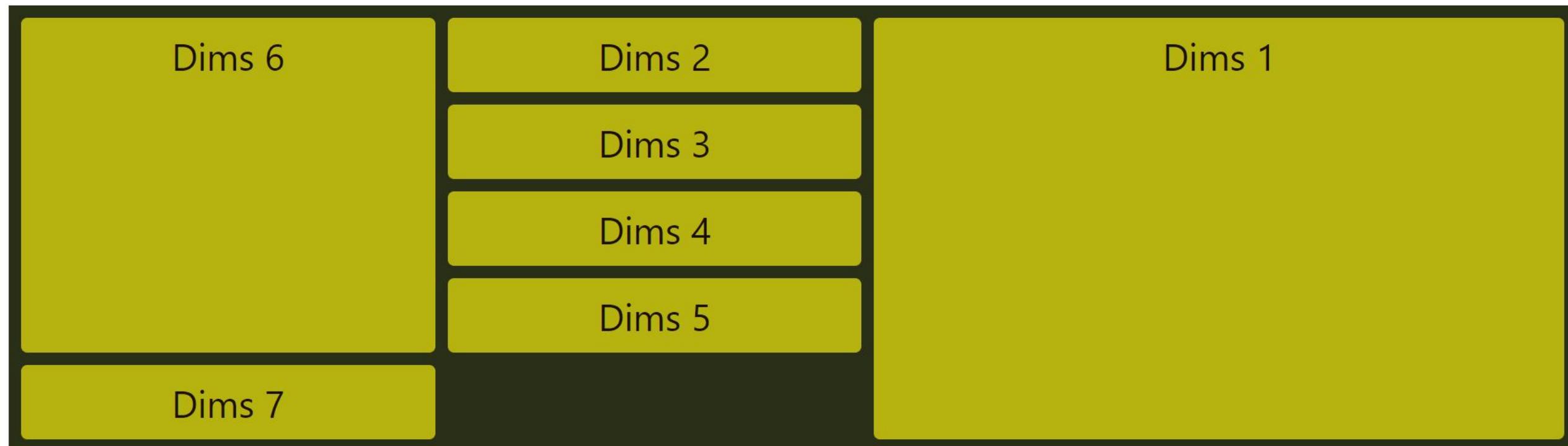


// div med teksten Menu og Main er unavngivne, men div med Extra er navngiven

# CSS: Display via GRID, ANDRE TEKNIKKER – repræsentation af linjerne i griddet

Man give via **grid-template-areas** styre sine elementers layout via **tal, som repræsenterer linjerne i griddet**

```
.minContainer {  
    .item1 { grid-area: 1 / 3 / 6 / 5; }  
    .item6 { grid-area: 1 / 1 / 5 / 2; }  
}
```



```
// dims 1 begynder på række-linje 1, kolonne-linje 3 &  
// slutter ved række-linje 6, kolonne-linje 5  
// øvrige dimser falder ind i griddet efter rækkefølgen de er sat i...
```

## CSS: Display via GRID

Læs & træn CSS GRID via disse gode links:

<https://css-tricks.com/snippets/css/complete-guide-grid/>

[https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Grid\\_Layout#guides](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout#guides)

<https://cssgridgarden.com/>

<https://mozilla-developers.github.io/playground/css-grid/>

[https://www.w3schools.com/css/css\\_grid.asp](https://www.w3schools.com/css/css_grid.asp)

## CSS: Responsivt GRID - eksempel

```
.grid-container {  
    display: grid;  
    grid-template-columns: 1fr 1fr 1fr;      /* 3 kolonner, fr står for fraction, */  
    grid-gap: 5%;                         /* %-enhed gør det også responsivt */  
}  
.grid-item {  
    /* Grid item styles */  
}  
  
@media (max-width: 768px) {  
    .grid-container {  
        grid-template-columns: 1fr 1fr;      /* 2 kolonner på små skærme */  
    }  
}  
  
@media (max-width: 480px) {  
    .grid-container {  
        grid-template-columns: 1fr;         /* 1 kolonne på ekstra små skærme */  
    }  
}
```

# Javascript

**basal syntax, p5.js  
& DOM via js**

# Javascript: BASIS

## Placering af js-kode i html-dokument –

Geerne nederst i body sådan at man ikke laver query på endnu ikke oprettede html-elementer

```
let minData = 10; // minData sættes til typen number med værdien 10  
console.log(minData + 5); // brug af konsollen til at tjekke kode
```

## Variable i js:

```
let minData = 10; // let-variable kan ændres – let som i 'lad det være det nu'...  
const minKonstant = 10; // const er til konstante variable 😊
```

**var** – block scope vs. ikke-block scope

Brug ikke **var**, er ikke block-scope, men function-scope – det kapsles ikke inde af fx if...

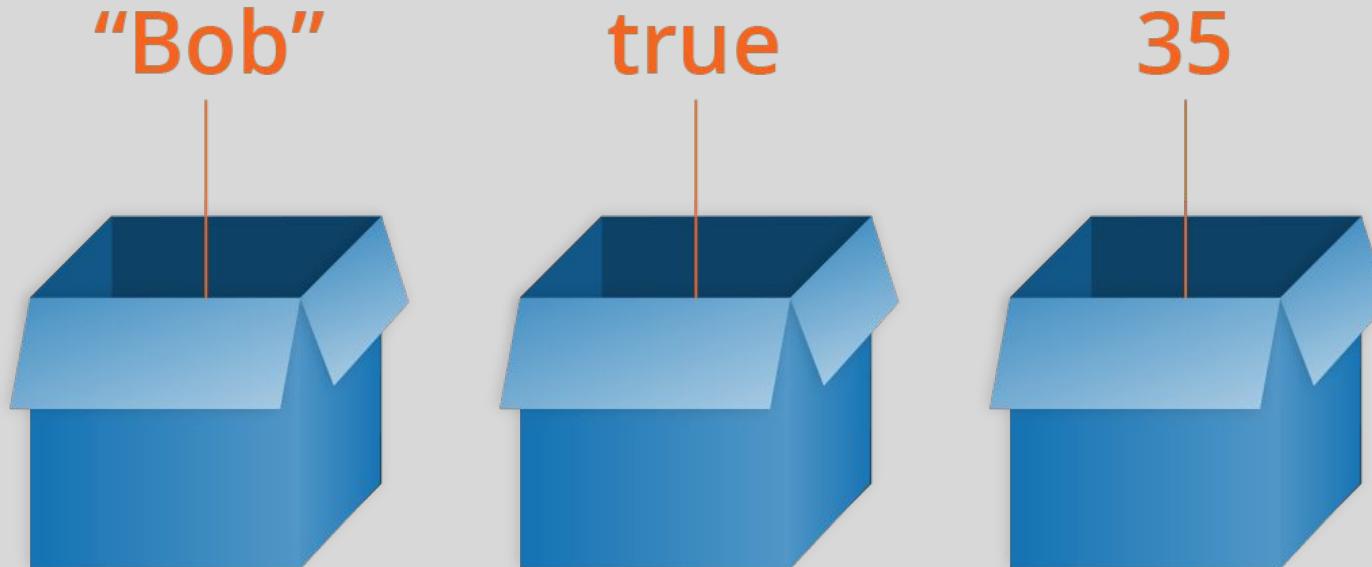
Og det knyttes an til det globale objekt, hvis det erklæres uden for en funktion

**Opgave 1:** Se introJS.html – hvilke fejl får man og hvorfor?

**Opgave 2:** Tilføj js-kode til jeres html-dokument!

# VARIABLE!

Definerer hvad man kan gøre på en værdi,  
Og hvor meget hukommelse variablen skal tildeles af computeren!



```
let kasse1 = "Bob"; // en streng - variablen kasse1 erklæres og initialiseres i samme udsagn med værdien Bob
let kasse2 = true; // en boolean – en sand eller falsk, true eller false
let kasse3 = 35; // number – heltal eller kommatal i Javascript
```

# VARIABLE i JS!

## De 7 primitive typer

- **number**: tal, heltal og kommatal!
- **bigint**: til at arbejde præcist med meget store heltal
- **string**: en række af karakterer (enkelte karakterer som 'char' findes ikke i JS)
- **boolean**: true eller false
- **null**: Står for at variablen ikke har nogen værdi, men at man netop aktivt har defineret den som 'null'
- **undefined**: Står for at variablen ikke er givet en værdi, ikke blevet initialiseret
- **symbol**: Står for en unik værdi, som kan gives til objekt

De kan alle kun rumme én værdi af gangen  
Og de er alle såkaldt værdi-variable.

# VARIABLE i JS!

## Den 8. datatype i JS er **Object-typen**

Object er ikke-primitiv, men netop sammensat!

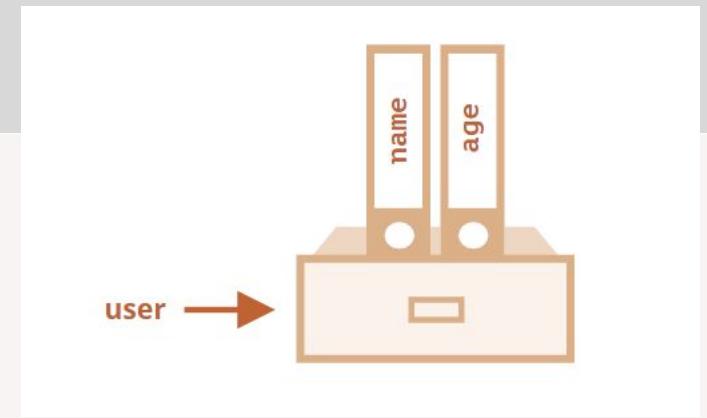
- user nedenfor en såkaldt '**object literal**' i JS , ikke lavet ud fra funktion, men netop direkte

```
1 let user = {      // an object
2   name: "John",  // by key "name" store value "John"
3   age: 30        // by key "age" store value 30
4 };
```

Se: <https://javascript.info/object>

console.log(user.name); // udskriver John i konsollen via **dot-notationen**.

Med **User**. - går vi ind i objektet user og derfra tager man fat på key'en, fx:  
user.age = 21; // objektet user's datafelt age bliver nu sat til 21.



```
Objekt-navn = {
  key: value,
  key: value
}
```

## VARIABLE i JS!

**De 7 primitive typer:** **number, bigint, string, boolean, null, undefined, symbol**

JS er et sprog med dynamiske typer – dvs. at man fx kan skrive:

```
let x = 5;      // x sættes implicit til at være et tal, typen number  
x = "Hello";    // men så sættes x implicit til typen string  
x = true;       // og nu er x en Boolean med værdien true!
```

Variablens type bliver altså defineret via den værdi, man tilskriver variablen OG man må gerne give samme variabel en ny slags værdi, mens programmet kører!

Men må ikke gøre noget på en variabel, som den ikke kan finde ud af, fx:

```
let mitTal = 1001; // variablen mitTal får værdien 1001, og er dermed af typen number
```

```
let bogstav = tal. age.charAt(0); // på mitTal bliver der kørt operationen charAt() – men virker kun på strenge!
```

Det giver en fejl, en **TypeError** – og programmet afbrydes under kørslen – *det chrasher!*

PS: Man kan dog tjekke, hvilken slags variabel man har at gøre med via –

```
if (typeof x === 'number') {...}
```

**VARIABLE i JS:** Den 8. datatype i JS er **Object-typen** - Object er ikke-primitiv, men netop sammensat!

Man kan også få fat i en key mere dynamisk ud fra brugerinput:

```
let key = name; // eller ud fra brugerinput
```

```
console.log(user[key]); // brug firkantet parentes - udskriver værdien, man har gemt  
under name i user
```

Man også tilføje ny egenskab/datafelt til sit objekt

```
user.nationality = 'danish';
```

Eller slette en egenskab via delete-keyword'et.

```
delete user.age;
```

**VARIABLE i JS:** Den 8. datatype i JS er **Object-typen** - Object er ikke-primitiv, men netop sammensat!

**Objekter lavet ud fra funktion:**

```
// to forskellige objekter laves ud fra samme funktion
let minKat = lavDyr('kat', 15, 9);
let minKanin = lavDyr('kanin', 5, 4);

// definition af funktion som netop returnerer et objekt via parametre
function lavDyr(art, vægt, alder){
    return {
        art,
        vægt,
        alder
    };
}
```

Man kan også via **new** – få en funktion til at returnere et nyt object - og definitionen skal så ikke have return  
let minHund = new lavDyr('hund', 25, 12);

## VARIABLE i JS: Den 8. datatype i JS er **Object-typen** - Object er ikke-primitiv, men netop sammensat!

### I øvrigt om objekter i JS:

- 1: Giver ikke fejl, hvis man spørger efter ikke-eksisterende egenskab/datafelt i et object – man får bare undefined!
- 2: Man kan via keywordet **in** – spørge om et object rummer en egenskab eller ej:

```
if('art' in minKat === true){  
    //...  
}
```

- 3: Man kan iterere igennem et objekts egenskaber via en speciel for-in-løkke:

```
for(key in minKat){  
    console.log(key);          // udskriver egenskabernes navne  
    console.log(minKat[key]);  // udskriver værdierne af egenskaberne  
}
```

- 4: Objekter kan tænkes som et associativt array – nøgler og værdier, komma-separeret.
- 5: Man kan også stoppe objekter ind i objekter – og metoder samt arrays!
- 6: Objekter er reference-variable – i modsætning til de primitive variable som er værdi-variable
- 7: Man kan godt ændre i et objekts egenskaber, selvom det er gemt i en const-variable
- 8: Man kan kopiere et object via `Object.assign`: `let minKatKlon = Object.assign( {}, minKat);` // giver ikke reference (hvis objektet rummer nestede objekter, skal man lave deep cloning 😊 og ikke shallow cloning)

# At interagere med DOM via JS: CRUD på html-træet

I js-koden kan vi lave **CRUD** på DOM – *altså create, read, update eller delete* – på elementer i html-træet.

## Create, eksempel: opret html-element

```
let nyParagraf = document.createElement('p');
nyParagraf.textContent = "Funky ny tekstparagraf oprettet";
document.body.appendChild(nyParagraf);           // vælger document-objektet, dets body og føjer den nye
'p' til
```

## Update, eksempel: lav html-element om

```
let minParagraf = document.querySelector("#kontaktTekst");           // vælger element via 'id'
minParagraf.textContent = "Paradisæblevej 12, Andeby 2800";           // giver nyt indhold til textContent
minParagraf.style.backgroundColor = 'white';                          // laver stilisering af background på
elementet

let mineP = document.querySelectorAll(".mineParagraffer");           // en nodeList af html-elementer
returneres...
for(let i = 0; i < mineP; i++){
    mineP[i].style.border = "5px solid";                            // itererer igennem listen og giver den en
border
}
```

# At interagere med DOM via JS: CRUD på html-træet

I js-koden kan vi lave **CRUD** på DOM – *altså create, read, opdate eller delete* – på elementer i html-træet.  
At kravle rundt i træet via parent, child osv.

```
<!DOCTYPE html>
<html>
<body>

<div>
<button id="minKnap" onclick="myFunction()">Slet det hele!</button>
<h2>Denne overskrift kan slettes sammen med hele div'en</h2>
<p>Denne smukke paragraf kan også slettes</p>
</div>

<script>
function myFunction() {
  let knap = document.querySelector("#minKnap");
  knap.parentNode.remove();
}
</script>
</body>
</html>
```

Slet det hele!

**Denne overskrift kan slettes sammen med hele div'en**

Denne smukke paragraf kan også slettes

Klikker man på knappen, så kaldes **myFunction**, hvori man tager fat i knappens **parentNode** og kalder **remove()**  
Derfor rydder man det hele, inklusiv knappen selv, da man smider div'en væk med dets children!  
**function** er et keyword som angiver, at ny defineres der en funktion/metode i koden 😊

Det samme kan man gøre på 'siblings' og på 'children' af noder i html-træet!

## Events via JS: At håndtere eller reagere på events!

En event er, når noget sker, fx: En klik på musen, en bevægelse over et html-element eller tryk på tast

Man kan få kaldt en js-funktion i tilfælde af en specifik event: **Klik på element, kald denne funktion**

```
<html>
<body>
<h1 onclick = "changeText(this)"> Klik på denne overskrift </h1>

<script>
function changeText(id) {    // kald denne funktion, en håndtering af event'en
    id.innerHTML = "Dette er reaktionen på en klik-event!";
}
</script>

</body>
</html>

// ved klik på h1-overskriften kaldes funktionen changeText. Elementet ryger med som parameter via 'this'
// øvrige events er fx – onmouseout, onmouseover, onkeypress, onload (når dokumentet har loaded)
```

# Events via JS: At tilføje en event-listener til et element!

Man via js få et element til at lytte på en event

Først får man i js-koden fat på elementet via **document.getElementById("myBtn");**

Og så tilføjer man en **event-listener** til elementet, hvor en funktion gives med som parameter:

Ved **mouseover**, gives **myFunction** med – dvs. når musen går over knappen, kaldes **myFunction**

```
|<button id="myBtn">Try it</button>
<p id="demo"></p>

<script>
var x = document.getElementById("myBtn");
x.addEventListener("mouseover", myFunction);
x.addEventListener("click", mySecondFunction);
x.addEventListener("mouseout", myThirdFunction);

function myFunction() {
  document.getElementById("demo").innerHTML += "Moused over!<br>";
}
function mySecondFunction() {
  document.getElementById("demo").innerHTML += "Clicked!<br>";
}
function myThirdFunction() {
  document.getElementById("demo").innerHTML += "Moused out!<br>";
}
</script>
</body>
</html>
```

## JavaScript addEventListener()

This example uses the addEventListener() method to add many events on the same button.

Try it

Moused over!  
Moused out!  
Moused over!  
Clicked!  
Moused out!

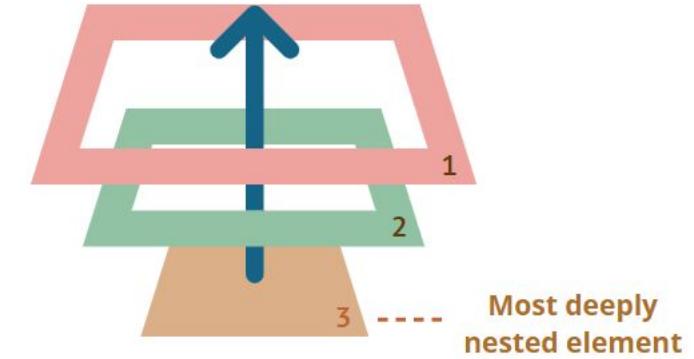
## Events via JS: Bubbling & delegation af events (muligvis overkill)

[https://www.w3schools.com/js/js\\_htmldom\\_eventlistener.asp](https://www.w3schools.com/js/js_htmldom_eventlistener.asp)

[https://www.w3schools.com/js/tryit.asp?filename=tryjs\\_addeventlistener\\_usecapture](https://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_usecapture)

Klikker man på nestet element, vil eventen, som fx et klik, bølle op og blive kørt af en parent-event-listener, hvis den er sat op dertil:

```
<form onclick="alert('form')">FORM  
  <div onclick="alert('div')">DIV  
    <p onclick="alert('p')">P</p>  
  </div>  
</form>
```



// først køres alert på p på p, dernæst på div'en og da på selve formen, hvis man har klikket på paragraffen p 😊  
// <https://javascript.info/bubbling-and-capturing>

## Delegation af event-håndtering:

Man lægger en event-listener på en parent til en gruppe af html-elementer, og så tjekker man via den kaldte funktion, hvilket element der blevet klikket på, og agerer så på det som ønsket

## ØVRIGE Javascript

Events: [https://www.w3schools.com/js/js\\_htmldom\\_events.asp](https://www.w3schools.com/js/js_htmldom_events.asp)

- <script>

```
document.getElementById("myBtn").onclick = displayDate;  
</script>
```

- var x = document.getElementById("myBtn");
- x.addEventListener("mouseover", myFunction);
- x.addEventListener("click", mySecondFunction);
- x.addEventListener("mouseout", myThirdFunction);

At operere på html-elementer via js: [https://www.w3schools.com/js/js\\_htmldom.asp](https://www.w3schools.com/js/js_htmldom.asp)

- PS: responsivt via css

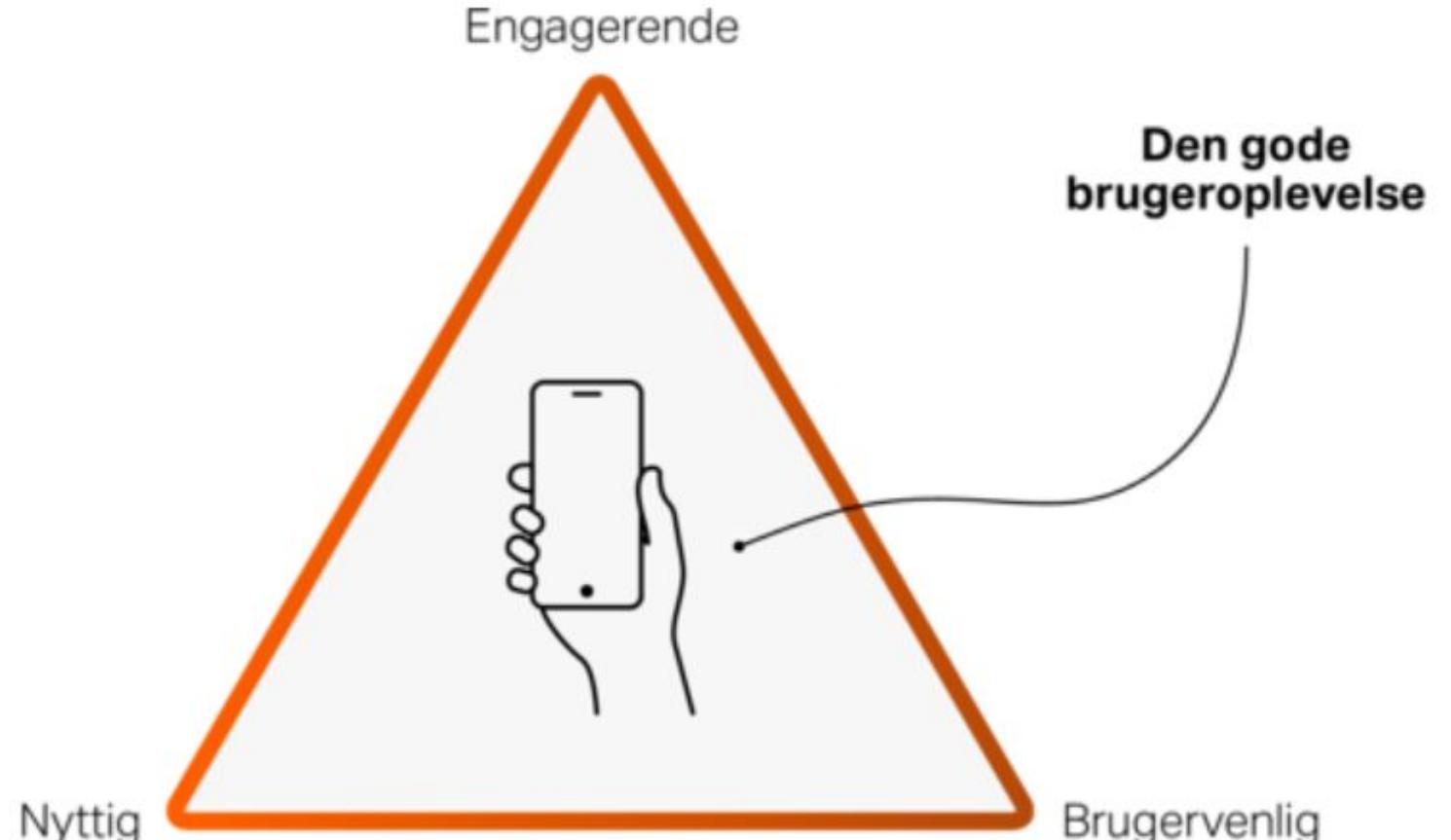
# UX: User experience

## Om brugeroplevelsen

Er systemet

- NYTTIG?
- BRUGERVENLIGT?
- & ENGAGERENDE?

## MODEL: UX-TREKANTEN



# head, body, style til css & script til js-kode

# p5-js accelerometer og canvas til animation

# p5.js

Home  
Editor  
Download  
Donate  
Get Started  
Reference  
Libraries  
Learn  
Teach  
Examples  
Contribute  
Books  
Community  
Showcase

## Reference

Search reference

### accelerationX

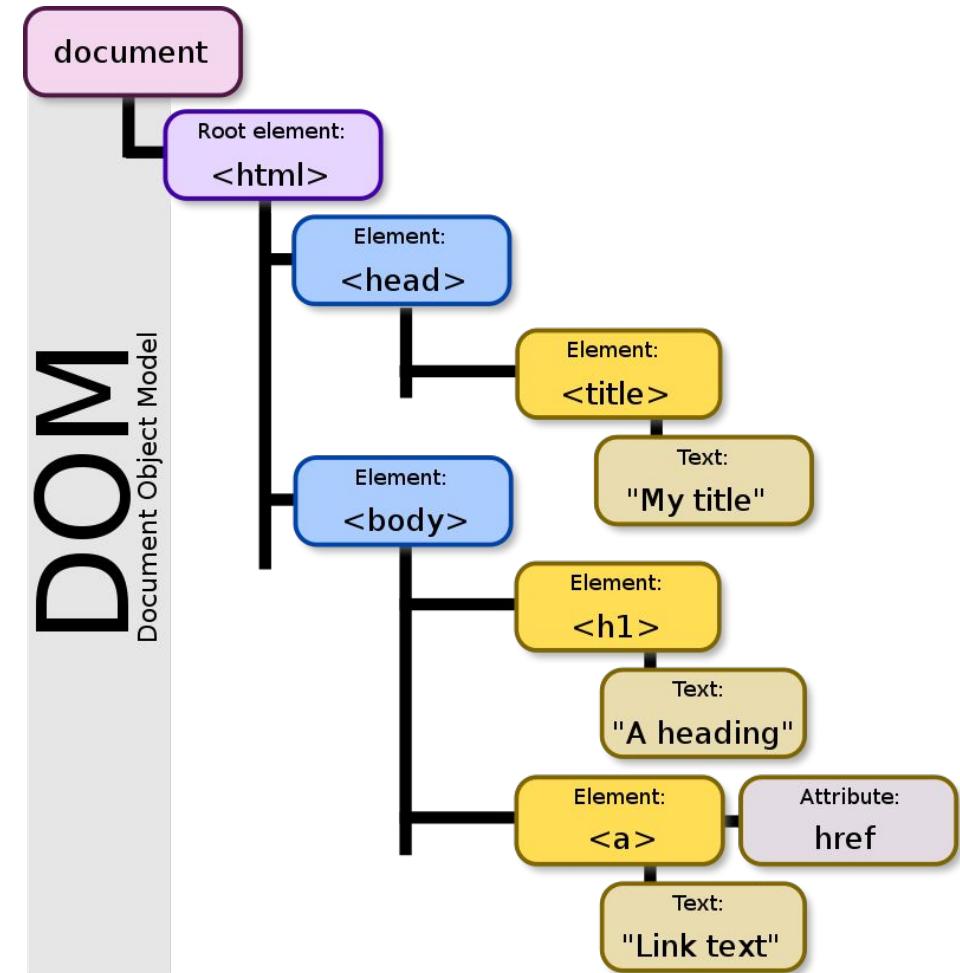
#### Description

The system variable accelerationX always contains the acceleration of the device along the x axis. Value is represented as meters per second squared.

#### Examples

Press Shift-Space to insert tab. edit reset copy

```
// Move a touchscreen device to register
// acceleration changes.
function draw() {
  background(220, 50);
  fill('magenta');
  ellipse(width / 2, height / 2, accelerationX);
  describe('Magnitude of device acceleration is
displayed as ellipse size.');
}
```



# WEBAPPLIKATION: BEVÆG FOLKESKOLEELEVER VIA TELEFONENS ACCELEROMETER

Lav et websted via **visual studio code** og udgiv det gerne på www via **github pages**

Brug **ux-trekanten** til at gøre det attraktivt som medieprodukt

Brug **p5js-biblioteket** som en nem vej til at lave dynamisk grafik i et **html-canvas**

Brug fx accelerometer-værdier til at lave selve bevægelsen (se p5js referencen på nettet)

Brug gerne **CSS-grid** som ramme for diverse sektioner, fx:

- Header med overskrift & forklaring af legen eller spillet
- Grafiske elementer til at skabe engagement og iscenesætte aktiviteten
- Både inde i canvas og rundt om via øvrige html-elementer
- input-indstillinger til canvas via button, checkbox, slider eller andet ux-element
- Footer med ekstra-info i bunden (webkonvention)

**Accelerometer-værdien** er: meter per sekund per sekund

Hvor mange meter per sekund stiger hastigheden netop per sekund 😊

Man kan måle hvor hurtigt man sætter bevægelse i gang, og hvor hurtigt man bremser den!

Alternativer: Male-app via bevægelse: [https://www.youtube.com/watch?v=DT2\\_LI8b-Vs](https://www.youtube.com/watch?v=DT2_LI8b-Vs)

<https://learn.hobye.dk/development/p5js/examples>

<https://developer.mozilla.org/en-US/docs/Web/API/Sensor> - sensor-api overordnet, js

<https://www.youtube.com/watch?v=bNmhX9464t4> – js vanilla gyro

<https://www.youtube.com/watch?v=3ls013DBcww> – js coding train – geo api

# Javascript funktioner

# JS: funktioner er ‘function object’ – et action object i js

A function is a value representing an “action” – et action-object, kan man tænke det som 😊

Regular values like strings or numbers represent the *data* vs. A function can be perceived as an *action*.

We can pass it between variables and run when we want. They can be assigned, copied or declared in any place of the code.

Er ‘first class citizens in js’ – givet til variable, til objekter, som parameter, kan laves dynamisk under runtime osv.

- If the function is declared as a separate statement in the main code flow, en “**Function Declaration**”, eller regular function!
- If the function is created as a part of an expression, it’s called a “**Function Expression**”. (ka være anonyme funktioner 😊)
- Function Declarations are processed before the code block is executed - visible everywhere in the block. **Hoisting!**
- Function Expressions are created **when the execution flow reaches them**. Ikke før.

In most cases when we need to declare a function, a Function Declaration is preferable, because it is visible prior to the declaration itself. That gives us more flexibility in code organization, and is usually more readable.

So we should use a Function Expression only when a Function Declaration is not fit for the task.

En funktion i js er en særlig slags objekt, **et function object**. Har objekt-lignende natur !

`console.log(typeof myFunction); // Outputs "function"`

- Har **length-property**, som gemmer på antal forventede parametre, fx `minFunktion.length`
- **name-property**, gemmer på funktionens navn, fx `minFunktion.name` returnerer ‘`minFunktion`’
- Har indbyggede metoder – **bind(), call() & apply()**
- Kan tildeles nye properties, fx `minFunktion.ekstraNavn = 'superFunktion'; console.log(minFunktion.ekstraNavn);`
- Kan tildeles nye metoder! Fx `minFunktion.minEkstraFunktion('udskriv super');`
- Kan **gemmes i data-strukturer** som arrays og som properties i objekter
- Kan skabe **closure**, idet de kan tilgå og huske på locale variable i den funktions scope/blok, hvorfra de blev returneret!

## JS: function declaration vs. function expressions

I dette eksempel sættes **variablen welcome** til den ene funktion, hvis svaret på prompten er under 18, og til den anden hvis over.

Kald af **welcome()** kører så den funktion, som er blevet assigget til welcome-variablen, som altså så bliver til typen **function**

Det er en **function expression**.

Og idet variablen er erklæret uden for scope af if-statementet, men defineret deri, så kan den også kaldes udenfra...

```
let age = prompt("What is your age?", 18);
```

```
let welcome;
```

```
if (age < 18) {  
    welcome = function() {  
        alert("Hello!");  
    };  
}  
else {  
    welcome = function() {  
        alert("Greetings!");  
    };  
}
```

```
welcome(); // kør velkommen
```

# JS: funktioner – callback-functions

At give en funktion med som argument til en anden funktion/metode

**1:** I dette eksempel gives funktionen **logPerson** med som argument – til metoden **forEach**, som er indbygget i array-objektet people  
logPerson tager to parametre – **person og index** – og den udskriver til konsollen værdien af dem, idet man itererer igennem array'en, man har kaldt forEach på...

```
// callbacks & foreach
let people = ['mario', 'luigi', 'ryu', 'shaun', 'chun-li'];
const logPerson = (person, index) => {
  console.log(` ${index} - hello ${person}`);
};
people.forEach(logPerson);
```

**2:** Og her definerer man funktionen i samme hug, som man indsætter den i kaldet af forEach på arrayen people....  
Via '**person**' udskrives hver værdi i array

```
let people = ['mario', 'luigi', 'ryu', 'shaun', 'chun-li'];
people.forEach(function(person){
  console.log(person);
});
```

**3:** Her får man reference til html-elementet **ul**, unordered list, med klassen **.people**, hvorefter man igen kalder foreach på arrayen people – og så føjer man til **strenge** **html** i hver iteration – en streng, som kan fungere som html-kode:

html-strenge opretter en li, **list-item**, med lilla farve  
og med værdien person  
html-strenge anvendes idet man siger **ul.innerHTML = html;**

```
// get a reference to the 'ul'
const ul = document.querySelector('.people');

const people = ['mario', 'luigi', 'ryu', 'shaun', 'chun-li'];

let html = ``;

people.forEach(person => [
  // create html template
  html += `<li style="color: purple">${person}</li>`;
]);
console.log(html);
ul.innerHTML = html;
```

# JS: funktioner – callback-functions

- Function Passed as Argument:** A callback function is defined like any other JavaScript function, but it is passed as an argument to another function. The receiving function can then invoke the callback at the appropriate time.
- Execution Deferred:** Callback functions are typically executed later, not immediately when passed as arguments. They are invoked at a specific point in the future, such as after an asynchronous operation is complete or in response to an event.
- Asynchronous Behavior:** Callbacks are often used to handle asynchronous operations like data fetching, timers, and I/O operations. Instead of blocking the program's execution, a callback allows you to specify what should happen when the operation finishes.
- Event Handling:** Callbacks are commonly used for event handling. You can register a callback function to be triggered when a specific event occurs, such as a button click or a keyboard press.
- Error Handling:** Callbacks can also be used for error handling. In many cases, callback functions take an error parameter as the first argument, allowing you to handle errors that may occur during an operation.

```
function doSomethingAsync(callback) {  
    setTimeout(function () {  
        console.log("Operation completed.");  
        callback(); // kald callback function  
    }, 1000);  
}  
  
function onComplete() {  
    console.log("Callback function executed.");  
}  
  
// Pass onComplete as a callback  
doSomethingAsync(onComplete);
```

## JS: closure – en funktion med indkapsling af egen implicitte, lokale og private variabel

Via nested funktion kan man skab closure af en variabel i js:

Den returnerede anonyme funktion gemmes i variabel **counter**. Denne funktion gemmer på sit 'lexical enviroment', som netop implicit indeholder variabel **count**.

Og hvis man kalder den, **counter()**, så returneres count og tæller dernæst count én gang op.  
(`return ++count;` tæller op og returnerer så)

MEN hvis man laver en **counter2**, så gemmer den igen på sin egen implicitte count-variabel, som ikke har effekt på counter-metodens egen count-variabel.

Det skaber closure – og en dermed **indkapsling/privat**

Se filen **jsClosure** –

også for eksempel på komplet funktion-objekt med egen closure og get og set-metoder

<https://javascript.info/closure>

```
function makeCounter() {  
  let count = 0;  
  
  return function() {  
    return count++;  
  }  
  
let counter = makeCounter();  
  
alert( counter() ); // 0  
alert( counter() ); // 1  
alert( counter() ); // 2
```

# JS: closure – eget, implicitte variabel-scope

Via nested funktion kan man skab closure af en variabel i js:

**counter** har reference til eget implicitte Enviroment-objekt, hvori erklærede variable registreres, og hvori den lokale, unikke **count** variabel bliver oprettet.

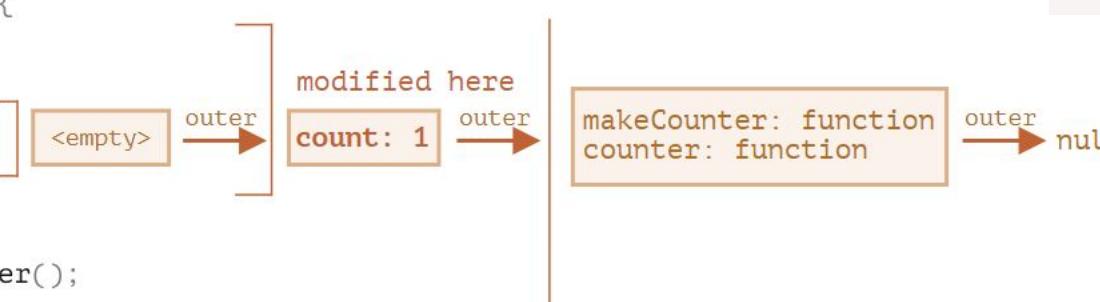
Først ser den om den inden for eget scope har en count-variabel, nej, og så kigger den i sit ydre scope.

PS: En count-variabel inden for eget scope i den returnerede funktion ville blive anvendt i stedet, hvis den havde været der...

A variable is updated in the Lexical Environment where it lives.

Here's the state after the execution:

```
function makeCounter() {  
  let count = 0;  
  
  return function() {  
    return count++;  
  };  
}  
  
let counter = makeCounter();  
  
alert( counter() ); // 0
```



```
function makeCounter() {  
  let count = 0;  
  
  return function() {  
    // let count = 77:  
    // ville træde i stedet for  
    // variablen count ovenfor  
    return count++;  
  };  
}
```

```
let counter= makeCounter();  
  
alert( counter() ); // 0  
alert( counter() ); // 1  
alert( counter() ); // 2
```

## JS: closure – flere funktioner returneret, closure vs property

Her returneres ikke bare et funktions-objekt, men et objekt med tre funktioner –

Og det indebærer closure, men man kan nu tælle op, ned samt gette på værdien

**PS:** Man kan dog godt, ret forvirrende, **parallelt** oprette en property på det returnerede objekt, fx:

```
let counter = createCounter();
counter.count = 99;
```

```
console.log(getCount()); // outputter 0 og ikke 99.
console.log(counter.count); // outputter 99, ikke 0
```

Det ene er en implicit variabel gemt i funktionens ‘enviroment’, mens det andet er en property på objektet counter.

Se igen: [filen jsClosure.html](#)

```
function createCounter() {
  let count = 0;

  function getCount() {
    return count;
  }

  function increment() {
    count++;
  }

  function decrement() {
    count--;
  }

  return {
    getCount,
    increment,
    decrement,
  };
}
```

## JS: decoration af funktion i js - call

Via wrapper funktion kan man dekorere en funktion, altså udvide dens adfærd:

<https://javascript.info/call-apply-decorators>

Se eksempel hvor man pakker en cpu-tung funktion ind i en caching-funktion

Således at det via et Map returnerer det cachede resultat, hvis det allerede har været udregnet

Og så et eksempel via call, hvor man får det samme til at virke, hvis man indpakker en metode fra et objekt via call, som netop skyder metode-kaldet ind i objektet

## JS: Simpelt eksempel på brug af call() – ændring af referencen i this via call 😊

Man kalder først metoden **fullName** på objektet ‘person’ - og den kører som forventet. This refererer til sig selv som objekt.

Men så gør man det igen, hvor man skyder videre via **call**, hvori ‘person1’-objektet indsættes, hvorfra man så finder værdierne ‘John’ ‘Doe’:

Det ændrer dermed **fullName**-metodens **this** fra ‘person’ til person1: Men kun midlertidigt.

**This** inde i **fullName** refererer dermed til først person1 og dernæst til person2.

```
const person = {  
    firstName: "Peter",  
    lastName: "Plys",  
    fullName: function() {  
        return this.firstName + " " + this.lastName;  
    }  
}  
  
const person1 = {  
    firstName: "John",  
    lastName: "Doe"  
}  
const person2 = {  
    firstName: "Mary",  
    lastName: "Doe"  
}  
  
person.fullName();  
// udskriver Peter plys  
person.fullName.call(person1);  
// udskriver "John Doe":  
person.fullName.call(person2);  
// udskriver "Mary Doe":
```

## JS: bind af funktion i js – bind() fastholder this, når funktion kaldes

Funktioner i js kan gives med som parametre og gemmes i variable,  
Men man kan miste this, altså konteksten

Obj har en funktion gemt som getX, som returnerer property x fra samme objekt.

Hvis man kopierer den funktion over i anden variabel, og kalder den, mister den sin kontekst, this.

Men man kan binde den op på dens kontekst-this via bind():  
let boundGetX = **obj.getX.bind(obj)**;

Dermed får funktionen gemt i boundGetX – referencen med til obj.x;

Hvis man bare sagde:

kopiGetX = obj.getX;

- Ville man få fejl...
- (medmindre der var en var x variabel oprettet som global variabel, dårlig ting!)

PS: I JS –

The value of this is evaluated during the run-time, depending on the context.

```
const obj = {  
    x: 42,  
    getX: function () {  
        return this.x;  
    },  
};  
  
const unboundGetX = obj.getX;  
  
// This will result in an error  
console.log(unboundGetX());  
  
const boundGetX = obj.getX.bind(obj);  
  
// Outputs 42, as it's now bound to obj  
console.log(boundGetX());
```

# JS: arrow-funktioner - korthed

Arrow funktioner kan oprette en **anonym funktion** ganske kort:  
Parametrene a og b lægges i funktion, som giver summen af de  
to, og den gemmes i variablen sum.

**Return-keywordet udelades ved én linje uden {}**

Derfra kan den kaldes efter behov, som man plejer:  
sum(1,2) giver 3, sum (1,3) giver 4 😊

Det samme, hvor kun én parameter gives: n

**Man kan da udelade parentesen.**

En funktion returneres somdobler parameteren op, og  
returnerer den: double(3) giver 6, dobbelt 4 giver 8 😊

Hvis ingen parameter, **så tom parentes**

let goddag = () => "bon jour!";

Hvis metoden har flere linjer end én, så skal den have {}

```
let nyFunktion = (a, b, c) => {  
    // gør ting med a, b og c...  
    // husk return-keyword når der er flere linjer med {}  
}
```

```
1 let sum = (a, b) => a + b;  
2  
3 /* This arrow function is a shorter form of:  
4  
5 let sum = function(a, b) {  
6     return a + b;  
7 };  
8 */  
9  
10 alert( sum(1, 2) ); // 3
```

```
1 let double = n => n * 2;  
2 // roughly the same as: let double = function(n) { return n * 2 }  
3  
4 alert( double(3) ); // 6
```

# JS: arrow-funktioner – minus egen context/this

Arrow funktioner har ikke sit eget this:

Funktionen **showList** kalder **forEach** på objektets array **students**, hvorefter der via arrow-funktionskald kører en alert med reference til **this.title**

Det giver netop ikke fejl, fordi arrow-kald ikke implicerer egen this, hvorfor **this** dermed stadig peger tilbage på objektet.  
Den arver this fra sin kontekst.

Heri kører der der en alm. funktion, men som implicerer egen this, hvorfor **this.title** ikke kan aflæses!

Dermed forbliver this reference til den funktions kontekst, hvori arrow-kaldet sker – der laves ikke en ny variabel kontekst:

De skal nemlige fungere i den kontekst, hvori de anvendes

```
1 let group = {  
2   title: "Our Group",  
3   students: ["John", "Pete", "Alice"],  
4  
5   showList() {  
6     this.students.forEach(  
7       student => alert(this.title + ': ' + student)  
8     );  
9   }  
10};  
11  
12 group.showList();
```

```
1 let group = {  
2   title: "Our Group",  
3   students: ["John", "Pete", "Alice"],  
4  
5   showList() {  
6     this.students.forEach(function(student) {  
7       // Error: Cannot read property 'title' of undefined  
8       alert(this.title + ': ' + student);  
9     });  
10   }  
11};  
12  
13 group.showList();
```

# JS: Funktioner

Om

# JS: FUNKTIONER, ØVRIGE TING AT SE PÅ

Rest-parametre og spread

Nye Function syntax

SetInterval osv.

Cookie, local storage

Class

Strict mode, - loose equality & strict equality, == & ===

Network requests, <https://javascript.info/fetch>

# **JS: window object and document object**

The `window` object and the `document` object are related but distinct objects in the browser's JavaScript environment. They are not parent-child objects in the sense of a hierarchical relationship; rather, the `document` object is a property of the `window` object. Here's the relationship:

## **1. \*\*`window` Object\*\*:**

- The `window` object represents the global context for a web page or the environment in which JavaScript code runs.
- It serves as the top-level object in the browser's JavaScript environment and is responsible for managing the entire browser window and its environment.
- The `window` object includes properties and methods related to the browser window itself (e.g., window size, location, navigation) and the global JavaScript environment (e.g., global variables and functions).
- One of the properties of the `window` object is the `document` property.

## **2. \*\*`document` Object\*\*:**

- The `document` object represents the web page's DOM (Document Object Model), which is a structured representation of the HTML content of the web page.
- It is a property of the `window` object, which means you can access it as `window.document` or simply `document`.
- The `document` object provides access to the elements and structure of the web page and allows you to interact with and manipulate the content.
- It includes properties and methods for selecting and modifying HTML elements, handling events, and making changes to the web page's structure and content.

In essence, the `document` object is a property of the `window` object, and it represents the DOM of the currently loaded web page. You can access the `document` object directly as `document`, and you can also access it through the `window` object as `window.document`. Both references point to the same `document` object.

```
console.log(this); // det udskriver hele window-objektet i konsollen!
```

# Javascript

string

Number

Array

modules

## JS: string

- There are 3 types of quotes. Backticks allow a string to span multiple lines and embed expressions \${...}.
- We can use special characters, such as a line break \n.
- To get a character, use: [] or at method.
- To get a substring, use: slice or substring.
- To lowercase/uppercase a string, use: toLowerCase/toUpperCase.
- To look for a substring, use: indexOf, or includes/startsWith/endsWith for simple checks.
- To compare strings according to the language, use: localeCompare, otherwise they are compared by character codes.

There are several other helpful methods in strings:

- str.trim() – removes (“trims”) spaces from the beginning and end of the string.
- str.repeat(n) – repeats the string n times

<https://p5js.org/reference/#/p5/split>

## JS: number

Om number som type

## JS: array - basalt

Med en array har vi en ordnet liste af værdier – et særligt slags objekt!

At erklære en array:

```
let arr = new Array(); // bruges aldrig, næsten  
let arr = [];
```

```
let frugter = ["æble", "banan", "citron"];  
let frugt = frugter[0]; // frugt er nu lig "æble"
```

```
frugter[3] = "pære"; // nyt element er føjet til!  
let j = frugter.length; // j er nu lig 4
```

```
let mixedArray = [1, "hello", true, { name: "John" }] // man kan mixe datatyper i en og same array
```

```
let arrayFunktion = [ 'Apple', { name: 'John' }, true, function() { alert('hello'); } ]; // med funktion i!  
console.log(typeof arrayFunktion[3]); // udskriver function
```

# JS: array - stack

## Array som stack i js: push, pop, shift & unshift

```
let frugter = ["æble", "banan", "citron"];
let f = frugter.pop();      // popper øverste/sidste element fra stakken
console.log(frugter);     // udskriver ["æble", "banan"]
console.log(f);            // det poppede element blev gemt i f, altså "citron"

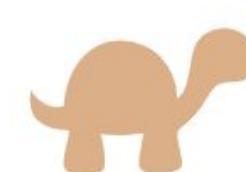
frugter.push("vindrue"); // "vindrue føjes til i enden på array'et
```

.shift() tager første element væk og  
unshift("avocado") lægger det til som første element  
frugter.unshift("avocado");

## Array er en reference variabel, ikke en primitiv datatype:

```
1 let fruits = ["Banana"]
2
3 let arr = fruits; // copy by reference (two variables reference the same array)
4
5 alert( arr === fruits ); // true
6
7 arr.push("Pear"); // modify the array by reference
8
9 alert( fruits ); // Banana, Pear - 2 items now
```

Methods `push/pop` run fast, while `shift/unshif`



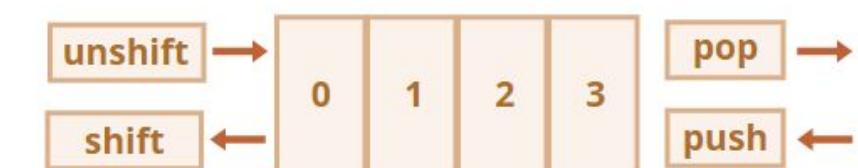
`unshift`

`shift`



`pop`

`push`



# JS: array – iterere igennem

At iterere igennem en array via klassisk for-loop:  
Man gribet fat i length-property på array-objektet

Og via for..of – loop:

Brug aldrig for..in

Husk at length-propertyen sættes som sådan:  
En position 123 vil indebære de foregående.  
Dårlig idé at gøre det sådan – memory-dyrt!

I js kan man rydde en array således:

arr.length = 0; // længden er nu 0, ergo er der ingen værdier i!  
Eller arr.length = 1; // første element bevares

Flerdimensionale arrays:

let m = matrix[1][1]; // giver 5  
- Fordi det er andet array, og anden plads i dette  
let n = matrix[2][2]; // giver 9

```
1 let arr = ["Apple", "Orange", "Pear"];
2
3 for (let i = 0; i < arr.length; i++) {
4   alert( arr[i] );
5 }
```

```
1 let fruits = ["Apple", "Orange", "Plum"];
2
3 // iterates over array elements
4 for (let fruit of fruits) {
5   alert( fruit );
6 }
```

```
1 let fruits = [];
2 fruits[123] = "Apple";
3
4 alert( fruits.length ); // 124
```

```
1 let matrix = [
2   [1, 2, 3],
3   [4, 5, 6],
4   [7, 8, 9]
5 ];
6
7 alert( matrix[1][1] ); // 5, the central element
```

## JS: array - metoder

Med en array har vi flere metoder endnu: <https://javascript.info/array-methods>

splice – at føje elementer til på specifik position

Slice – kopierer elementer fra specifik position, uden at fjerne dem

split

concat – at svejse arrays sammen, konkatenering, at sammenkæde dem

forEach – kalder function på hvert element, uden at returnere en array

indexOff,

Includes – arr.includes('a') – true/false

find, filter

Map – smider nyt array retur, efter kald af function på hvert element: man mapper værdi via function, og returnerer det

sort

reverse

```
const numbers = [1, 2, 3, 4, 5];
```

```
// Using the map method with an arrow function to create a new array of squared numbers
```

```
const squaredNumbers = numbers.map((num) => num * num);
```

```
console.log(squaredNumbers); // Outputs: [1, 4, 9, 16, 25]
```

```
// kalder forEach på hvert element i arrayet, og kører anonym funktion derpå
```

```
numbers.forEach((num) => console.log(num)); // Outputs (1,2..) without creating new array.
```

## JS: modules

I moderne js kan man gøre et script til et modul: Et modul andre scripts kan importe 😊

Det organiserer koden, gør den mere genbrugelig, overskuelig og indkapsler funktionalitet i afgrænsede moduler

Hvis et script skal anvende **modul-systemet**, angives det inline i scriptet:

```
<script type="module">
import {dims} from "./dimser.js";
// importerer man meget, kan man skrive import * as ting from "./dimser.js"
console.log(dims.n1);      // udskriver værdi1
console.log(dims.n3[1]);    // udskriver banan
</script>
```

I filen dimser.js

```
export const dims = { 'n1': 'værdi1', 'n2': 100, 'n3': ["citron", "banan", "pære"] }
export default const person = Person() { //kode her }; // default-element, vs. navngiven eksport
```

- Moduler gør, at **strict mode** i js automatisk bliver gældende.
- Moduler har sit **eget variabelrum** – de deles kun via import & export, medmindre en variabel eksplisit er gjort global via window.minVariabel = 10; (men det er dårlig stil!)
- I js-moduler er øverste **this** undefined, vs ikke-moduler, hvor this jo refererer til det globale objekt
- Man kan **re-exportere** noget importeret, hvis man har et slags hierarki af moduler, med under-moduler
- Man kan dynamisk importere via **modul()** – men ikke bruge import inde i fx en if-forgrening:
- let {dims, Person} = await import("./dimser.js"); Se: <https://javascript.info/modules-dynamic-imports>

## JS: Promise – at arbejde med asynkrone funktionskald

I moderne js kan man via Promise på en hensigtsmæssig måde lave asynkrone funktionskald, som fx læser fra filer, over netværk eller andre evt. tidskrævende operationer.

Det asynkrone gør, at programmet fortsætter med at køre, imens den asynkrone, og måske tidskrævende operation bliver afviklet. Programmet fortsætter altså, selvom operationen tager noget tid, fx 3 sekunder 😊

**Promise** har tre tilstande –

Pending – i gang

Fulfilled – lykkes, værdi fundet

eller rejected – afvist, error objekt givet

Promise

# JS: TING AT SE PÅ

Class & prototype-inheritance (js vs java og c#)

Cookie, local storage

Class

Strict mode, - loose equality & strict equality, == & ===

Network requests, <https://javascript.info/fetch>

# Javascript

## jQuery

jQuery er et meget udbredt **javascript-bibliotek**, som gør mange ting nemmere i arbejdet med at gøre HTML-elementer og webapplikationer dynamiske, og ikke statiske.

Alt man kan gøre med jQuery, kan også gøres med Vanilla Javascript, men gøres nemmere...

Fx:

```
$(".minKlasse")          // jQuery, tager fat på alle elementer af klassen minKlasse  
document.querySelectorAll(".minKlasse")      // samme med Vanilla JS
```

```
$( "p" ).css( "color", "red" );      // jQuery, farver alle paragraffer røde  
document.querySelector("p").style.color = "red";    // samme med Vanilla JS
```

PS: vanilje som basis i ispinde-industrien, uden ekstra smage eller krymmel 😊

## JS: INTRO TIL JQUERY, ET JS-BIBLIOTEK



**\$-tegnet** er en global variabel, som refererer til jQuery-funktionen:

- den oprettes, idet man inkluderer jQuery-scriptet
- jquery-funktionen benyttes til vælge eller oprette html-elementer

// dette er der samme:

```
let paragraphs = $('p');  
let paragraphs = jQuery('p');
```

Hvis andet js-bibliotek også bruger \$-tegnet som variabel, kan man kalde noConflict()  
Hvorefter man skal skrive jQuery og ikke bare \$-tegnet

```
$.noConflict();  
jQuery(document).ready(function(){ // når dokumentet er loaded...  
    jQuery("button").click(function() {  
        jQuery("p").text("jQuery is still working!");  
    });  
}); // se eksempel på jquery
```

## JS: INTRO TIL JQUERY, ET JS-BIBLIOTEK

```
// at tage fat på jquery  
// at vælge elementet  
// at kalde en metode  
$(selector).action();
```

```
// alle paragraffer "p",  
// kald metoden hide på dem  
$("p").hide()
```



Se oversigt over jquery-selektorer:

[https://www.w3schools.com/jquery/jquery\\_ref\\_selectors.asp](https://www.w3schools.com/jquery/jquery_ref_selectors.asp)

<http://api.jquery.com/category/selectors/>

Man kan fx udvælge en klasse, id eller type **\$(".test"), \$("#test), "p", "img"** med meget mere!



## Med jquery kan man blandt andet nemt:

- Get og Set af html-elementers indhold og stil
- Tilføje og fjerne html-elementer og tilføje et klasse-navn til et html-element
- Animere elementer og slide og fade osv.
- Løbe igennem DOM-træet, fx finde en 'parent' eller et barn til et html-element
- Via callback kalde en funktion, når en effekt er færdig
- Kalde funktioner ved html-events (fx klik, mouseenter, keydown)

Se oversigt: [https://www.w3schools.com/jquery/jquery\\_ref\\_events.asp](https://www.w3schools.com/jquery/jquery_ref_events.asp)

- Lave AJAX-operationer, hvor man via en server opdaterer elementer på webstedet uden at skulle genopfriske hele webstedet (Asynchronous JavaScript and XML).
- Derudover tilbyder jQuery også – færdige UI-komponenter (fx widgets) og ikoner!

## Hjælp til jQuery:

<https://css-tricks.com/lodge/learn-jquery/>

<http://learn.jquery.com/effects/intro-to-effects/>

<https://jqueryui.com/> (færdige UI-komponenter)

<https://www.w3schools.com/jquery/default.asp>

# **Vue.js: Framework, front-end**

# VUE: framework – vue.js

## Ideen med vue.js – et frontend framework med komponenter

Et komponent består af disse byggesten:

**<template>** // til html-kode – med binding til variable og komponenter samt særlige vue-operationer

**<script>** // js-kode til at bearbejde data og erklære variable samt importere andre komponenter

**<style>** // til at stilisere elementer inden for komponentet (style scope)

**Single File Component, SFC:** At samle html-kode, javascript og css-stilisering for et komponent i én fil.

Et komponent kan konkret udgøre en side på skærmen eller bare et delelement af en side.

Nemmere at teste og udvikle på samt nemmere at skifte ud på – fordi det er komponenter 😊

Man bygger det op via .vue-filer, som bygges eller kompileres til kode, som browseren kan forstå.

**Single File Application, SPA:** At bygge webapplikationer sendt i én omgang til browser fra server, sådan at man ikke skal genopfriske siden for at skifte over til anden skærmside.

Kan også bruges til mobilapp & desktopapplikationer...

## VUE : Opsætning af rodkomponent med router-objekt og knyttet til index.html

Det begynder med **main.js** (hvis altså computeren er sat op til at køre vue-projekter)

```
// import af createApp-objekt fra vue-frameworket
import { createApp } from 'vue'
// import af App-komponent som rod-komponentet – skydes ind i app-div'en
import App from './App.vue'
// import af router-objekt fra mappen router, hvori index.js ligger, som eksporterer router
// bruges til at route eller dirigere ml. de forskellige views/skærme
import router from './router'

// mount af vue-app i html-element med id'et app
createApp(App).use(router).mount('#app')

// vue-appen, via 'App', placeres derfor inden for denne primære div, som så udgør roden
// I index.html-filen ligger netop et div-element med id="app"
<div id="app"></div>
```

## VUE : index.js til routing

De forskellige views/skærme er lagt i array i - **index.js** - lagt i mappen router

```
import Home from '../views/Home.vue';
import Create from '@/views/Create.vue';

// views at route til, objekter lagt i array - Home udgør i dette tilfælde første view
const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/create',
    name: 'Create',
    component: Create
  }
]
// Oprettelse af router-objekt med router-historik samt mulige routes og eksport af objektet
const router = createRouter({history: createWebHistory(process.env.BASE_URL),routes});
export default router;
```

## VUE : App.vue

I **App.vue.fil**en, som jo er rodkomponent eller container, lægges komponenter, som skal bruges på tværs, fx en navbar-komponent 😊

Tagget <router-view/>, som så vil huse de views, som er gemt i router-objektet, og som man netop kan skifte imellem, lægges også ind.

Dette skydes ind i html-div'en app...

```
<template>
  <NavBar />      // brug af NavBar-komponentet på tværs af views
  <router-view/>    // brug af diverse views, første view er Home
</template>

<script>
import NavBar from './components/NavBar.vue'; // import af NavBar-komponent
export default{
  components: {NavBar}                      // eksport af NavBar, så det kan bruges ovenfor
}
</script>

<style>...</style>
```

## VUE: Nav-Bar komponentet – router-links

NavBar-komponentet som er placeret i toppen af App-rodkomponentet, får så fx disse elementer ind: Først en h2-tekst, som er almindelig html.

Dernæst to router-links, som netop kan linke frem og tilbage imellem Home-siden og Create-siden.

Man benytter komponent-navnet til at route eller linke til rette komponent:  
Komponentet udfylder så <route-view> elementet i App-rodkomponentet 😊

```
<template>
  <header>
    <h2>Blog</h2>
    <nav>
      <router-link :to="{ name: 'Home' }">HJEM</router-link>
      <router-link :to="{ name: 'Create' }">NY POST</router-link>
    </nav>
  </header>
</template>
```

## VUE: Basis i et komponent, i en vue-fil: <template>

Et komponent består som sagt af disse bidder:

<template> // til html-kode – med binding til variable og komponenter samt særlige vue-operationer  
<script> // js-kode til at bearbejde data og erklære variable samt importere andre komponenter  
<style> // til at stilisere elementer inden for komponentet (style scope)

I <template>.....</template>

- Lægger html-elementer ind, fx <button>, <img>, <div>, <h1>, <input>, <label> osv.
- Lægger andre komponenter ind, fx
- en spinner-grafik <Spinner/> eller et <Kontakt/> komponent
- Diverse særlige vue-ting, som kan smides direkte ind i html-koden, fx **v-if** til en forgrening eller **v-for** til et for-loop igennem et array. Se  
[https://www.w3schools.com/vue/vue\\_directives.php](https://www.w3schools.com/vue/vue_directives.php)
- Derudover kan man danne data-binding ml. værdier,
- eller kalde metoder fra script eller fyre en event af...

## VUE : Basis i et komponent, i en vue-fil: <script> & <style>

```
<template> // til html-kode – med binding til variable og komponenter samt særlige vue-operationer  
<script> // js-kode til at bearbejde data og erklære variable samt importere andre komponenter  
<style> // til at stilisere elementer inden for komponentet (style scope)
```

### I <script>.....</script>

- Kan man importere andre komponenter, fx `import Spinner from '@/components/Spinner.vue'`
- Kan man importere funktioner fra en js-fil, fx `import getEnkeltPost from '@/composable/getEnkeltPost';`
- Kan man importere særlige vue-objekter, fx `import {computed, onMounted, ref, watch} from 'vue';`
- Kalde `setup ()` – til at oprette variable og metoder, som til brug i <template>

```
export default {  
    setup(){  
        // oprette variable (primitive, objekter & arrays) samt funktioner  
        // returnere variable/funktioner eller komponenter til brug i <template> via  
        // return {tags, title, handleKeyNed, Spinner}  
    }  
}
```

</script>

Og via style-taget kan man css-stilisere elementer inden for komponentet,  
via <style scope>....</style>

# VUE: Reaktive variable - ref

Vue-keyword **ref** kan bruges til at lave reaktive variable  
(I composition-api'en er variable ikke reaktive per default)

Se eksempel:

Først import { **ref** } from 'vue'

Så laver man en ref-variabel:

const name = **ref('mario')**

- name bliver et reaktivt objekt med værdien 'mario'

Hvis man så ændrer den via handleClick-metoden:

**name.value** = 'luigi'

- så ændrer den også værdi i template
- man skal huske **name.value**
- i selve template behøver man ikke sige DOT-value

Og man skal som altid bruge:

- **return {name}** (for at kunne bruge den i template-delen)

<https://vuejs.org/guide/essentials/reactivity-fundamentals.html#ref>

```
src > views > Home.vue > {} "Home.vue" > script > default > setup > handleClick
1  <template>
2    <div class="home">
3      <h1>Home</h1>
4      <p>My name is {{ name }} and my age is {{ age }}</p>
5      <button @click="handleClick">click me</button>
6    </div>
7  </template>
8
9  <script>
10 import { ref } from 'vue'
11
12 export default {
13   name: 'Home',
14   setup() {
15     // const p = ref(null)
16
17     const name = ref('mario')
18     const age = ref(30)
19
20     const handleClick = () => {
21       name.value = 'luigi'
22       age.value = 35
23     }
24
25     return { name, age, handleClick }
26   },
27 }
```

## VUE: Reaktive variable via automatisk udregning, **computed**

Vue-keyword **computed** kan bruges til at lave reaktive variable, som gennemgår en form for beregning eller operation:  
Hvis den reaktive variabel ændres, så kaldes computed på den automatisk!

### EKSEMPEL:

Først import { **ref**, **computed** } from 'vue'

I **setup(){.....}** laver man så en reaktiv variabel: **search = ref("")**;  
Og et reaktivt array: **navne = ref(['bob', 'tim', 'ada', 'kim']);**

Endelig en **computed-variabel**, her et array - med de strenge-elementer, som inkluderer strengen search-value:  
matchendeNavne bliver netop udregnet igen og igen, når den reaktive variabel **search.value** får ny værdi!  
**computed** består altså af en anonym metode, som automatisk køres hvis reaktiv værdi deri ændrer sig

```
const search = ref("");
```

```
const navne = ref(['bob', 'tim', 'ada', 'kim']);
```

```
const matchendeNavne = computed(() => {
  return navne.value.filter((navn) => navn.includes(search.value));
});
```

# VUE: metodekald i komponentets cyklus, life cycle hooks

Vue tilbyder en række metodenavne, som af sig selv bliver kaldt ved forskellige stadier i et komponents liv.

Fx når et komponent bliver:

- **onMounted**, læst ind og tilføjet til DOM-træet
- **onCreated**, læst ind, men endnu ikke tilføjet til DOM-træet
- **onUnmounted**, taget af
- **onUpdated**, en værdi i komponentet har ændret sig

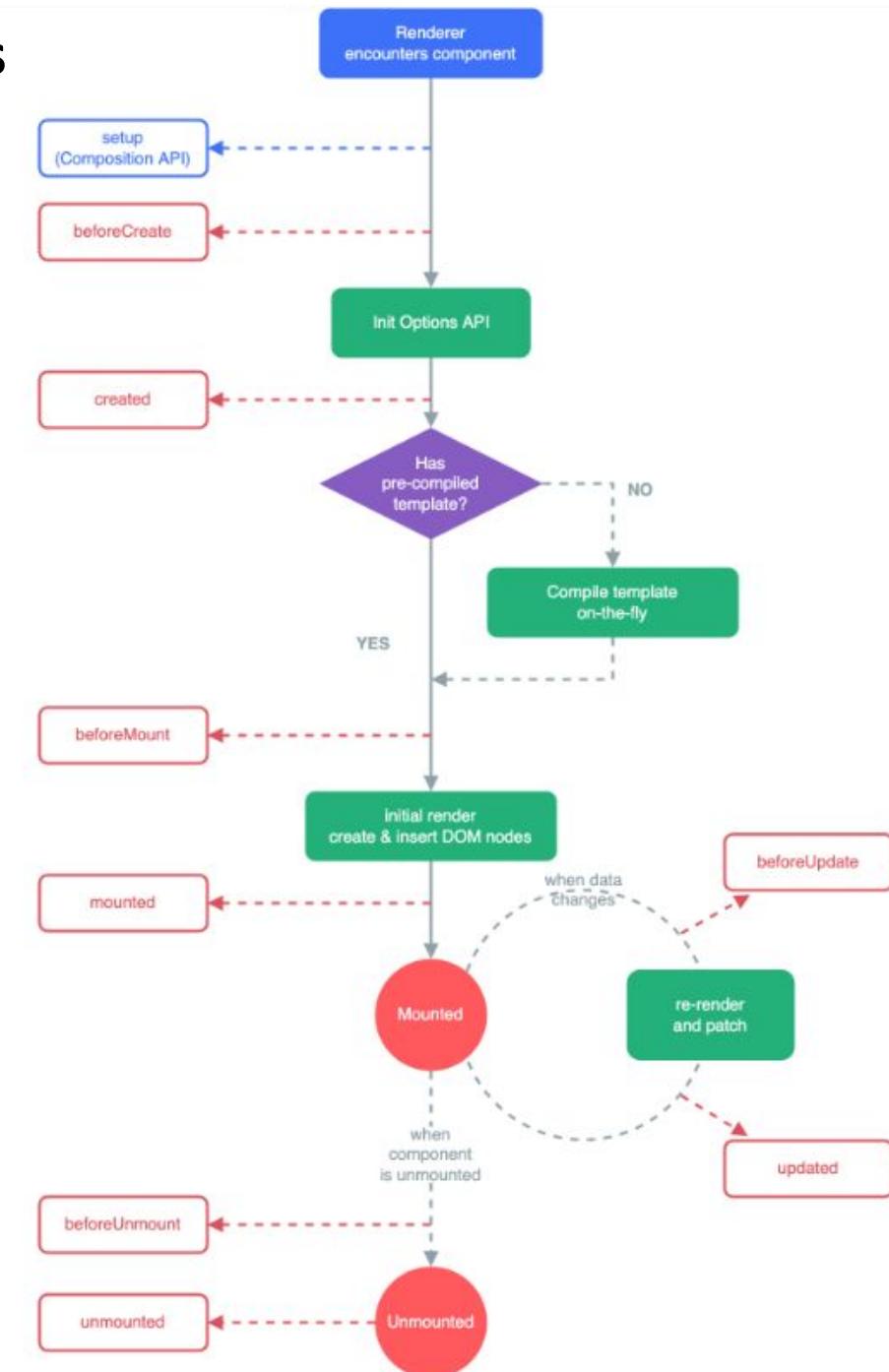
Eksempel:

```
setup(){  
  onMounted( () => console.log('Komponent er nu sat op!') );  
}
```

Se diagram:

<https://vuejs.org/guide/essentials/lifecycle.html#lifecycle-diagram>

<https://vuejs.org/guide/essentials/lifecycle.html>



## VUE: Props - at give variable med til sub-komponenter

I Vue kan man også give variable med til sub-komponenter, som de så kan benytte, fx:

**props: ['infoData'],**

```
setup(props) {  
    let nyArray = props.infoData;  
}
```

Sub-komponentet får det givet med i template-delen, fx sådan:

**<Kontakt v-bind:info = 'infoData' />**

// Kontakt-komponent ind i template, og der bindes til info, og det gives med som prop via navnet infoData

Se diagram:

<https://vuejs.org/guide/essentials/lifecycle.html#lifecycle-diagram>

<https://vuejs.org/guide/essentials/lifecycle.html>

# HTML: framework – vue.js

## Vue.js – oversigt over elementer

Opsætning via nodejs, npm og vue cli

Komponenter – main.js samt rodkomponent App med subkomponenter

Import af subkomponenter i rodkomponentet og export default af komponent-data

Komponentets byggesten – template, script og style

I template forlænger man html-syntaksen og deklarerer de elementer, som komponentet skal indeholde

I script exporterer man via 'export default {}' et objekt, som hjælper til at importere komponentets data, subkomponenter samt metoder

I script stiliserer man det enkelte komponents elementer., gerne via style scope

Global css og css scoped inden for komponentet

Methods – lagt ind i komponentets script-tag

Databinding

Props fra parent-komponent til subkomponent – envejs-ting oppefra og ned – data/properties gives videre til sub-komponentet.

Elementer i template-scriptet kan tilgås via 'ref'. Elementet gives et navn ref="ting" og via this.\$refs.ting kan man operere på elementet i script-delen, fx tilføje en ny klasse eller andet...

Life-cycle-hooks i vue:

<https://vuejs.org/guide/essentials/lifecycle.html#registering-lifecycle-hooks>

Event-emit og eventlistener

Visning af array i element, via v-for="element i elementer" :key="element"

To-vejs-databinding via v-model="variabelNavn" i fx input-felt, og ikke bare et-vejs-binding via {{variabelnavn}}

Kald af metode via @click="metodeNavn" eller @keyup="metodeNavn"

V-routing, til SPA udfoldet...routing-links og views som en slags rod-komponenter for hver enkelt skærm

Routing-links med parametre fra én skærm til en anden...

Frem og tilbage-knapper via router-historikken – knap, @click="gåTilbage", metode – this.\$router.go(-1) og frem er ...go(1)

Man kan re-directe via metode-kald – this.\$router.push({name: 'hjem'}) – altså et objekt gives med, hvor name er view-navnet registreret i index.js-filen, hvor man er listet sine views...

## **HTML: framework – vue.js**

Vue.js

[https://www.w3schools.com/vue/vue\\_intro.php](https://www.w3schools.com/vue/vue_intro.php)

[https://developer.mozilla.org/en-US/docs/Learn/Tools\\_and\\_testing/Client-side\\_JavaScript\\_frameworks/Vue\\_getting\\_started](https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Vue_getting_started)

Vue.js med bootstrap: <https://bootstrap-vue.org/>

NodeJS

[https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs)

# HTML: framework – vue.js

## Vue.js – bruger template-elementet som er et native html-element

The template is cloned using `document.importNode()` and then appended to the `<div>` element with the id of "app". Only at this point is the content of the template actually rendered in the browser.

If you don't clone and append the template's content to the DOM, it will remain hidden and not be displayed in the browser. The `<template>` element allows you to define reusable content without immediately rendering it, providing a way to generate and insert the content dynamically when needed.

```
// brug af template via dobbelt krøllede parenteser – hvis template ikke bliver appended til en div, så vises template-indholdet ikke...
// de dobbelt krøllede parenteser kan bruges til at gøre data dynamisk og til data-binding (to-vejs også)
<template id="my-template">
<h1>Hello, {{ name }}</h1> <p>Welcome to my website.</p>
</template>
<div id="app"></div>
<script>
const template = document.getElementById('my-template');
const app = document.getElementById('app');
const clone = document.importNode(template.content, true);
app.appendChild(clone);
</script>
```

# HTML: framework – vue.js – hvordan kompileres det, så browseren kan forstå det?

Vue files are typically compiled into JavaScript, HTML, and CSS code that can be understood and executed by web browsers. The compilation process involves several steps:

1. **\*\*Vue Template Compiler\*\***: Vue files contain `<template>`, `<script>`, and `<style>` sections. The Vue Template Compiler (part of the Vue.js framework) parses the `<template>` section and compiles it into a JavaScript render function. This render function generates the virtual DOM nodes required to render the component's UI.
2. **\*\*Bundler/Build Tool\*\***: Vue files are commonly processed by build tools like webpack, Parcel, or Vue CLI. These tools handle the compilation process and transform the Vue file into a format that can be understood by the browser. They may perform additional tasks like transpiling JavaScript (using Babel), optimizing the output, and handling dependencies.
3. **\*\*JavaScript Output\*\***: The compiled JavaScript code represents the logic and behavior of the Vue component. It includes the render function, component options, and any additional JavaScript code defined in the `<script>` section of the Vue file. This code is typically bundled into a single JavaScript file or multiple files for better organization and optimization.
4. **\*\*HTML and CSS Output\*\***: The HTML and CSS code defined in the `<template>` and `<style>` sections of the Vue file are extracted and processed accordingly. The HTML markup can be included directly in the JavaScript output, or it may be separated into a separate HTML file that is referenced by the JavaScript code. Similarly, the CSS styles can be extracted into a separate CSS file or included within the JavaScript output using CSS-in-JS techniques.

During development, tools like webpack-dev-server or Vue CLI's development server can dynamically serve and compile the Vue files on the fly as you make changes, allowing you to see the updates in real-time during development.

In summary, Vue files are compiled by tools like webpack or Vue CLI, which process the Vue file's template, script, and style sections and transform them into JavaScript, HTML, and CSS code that can be executed and rendered by web browsers.

# **HTML: framework – vue.js – anvendelser?**

1. **Single-Page Applications (SPAs):** Vue.js works exceptionally well for building SPAs, where the entire application is loaded once, and subsequent interactions and updates happen dynamically without full page reloads. Vue's reactive data-binding and component-based architecture make it efficient for creating interactive and responsive SPAs.
2. **Progressive Web Applications (PWAs):** Vue.js can be used to build PWAs, which are web applications that can function offline and provide a native-like experience to users. Vue's lightweight size, performance optimizations, and Vue Router's history mode make it a suitable choice for building PWAs.
3. **Mobile Applications:** Vue.js can be used to build mobile applications using frameworks like NativeScript-Vue or Quasar Framework. These frameworks enable developers to write Vue.js code that compiles into native iOS and Android applications, allowing code reuse and faster development.
4. **Desktop Applications:** Vue.js can be utilized to build desktop applications using frameworks such as Electron or NW.js. These frameworks leverage web technologies to create cross-platform desktop applications, and Vue.js can power the user interface and interactivity of these applications.
5. **Enterprise Applications:** Vue.js is suitable for building large-scale enterprise applications due to its scalability, maintainability, and robust ecosystem. It provides tools and patterns for code organization, state management (e.g., Vuex), routing (e.g., Vue Router), and server-side rendering (e.g., Nuxt.js) to handle complex application requirements.
6. **Content Management Systems (CMS):** Vue.js can be used to create custom CMS interfaces or extend existing CMS platforms like WordPress. Vue's reactivity and component-based structure can enhance the user experience and simplify the management of content.
7. **E-commerce Applications:** Vue.js can power the front-end of e-commerce applications, providing an interactive and dynamic shopping experience. Integrating Vue.js with backend platforms or APIs enables real-time updates, smooth transitions, and personalized user interfaces.

**X**

## Opgave 9: Julekalender

Du skal nu for første gang forsøge at skabe en hjemmeside selv - fra bunden - som bygger på en arkitektur, du selv planlægger og udfører. Du skal nemlig lave en hjemmeside-julekalender.

<https://www.javascript.christmas/2020>

Her er en meget sød "hjælpekalender" med Vanilla Javascript tips

<https://codepen.io/michelenl/embed/RWKaYP>

<https://codepen.io/michelenl/pen/RWKaYP> - et bud på en ren CSS julekalender. Vi vil gerne have lidt mere - dvs få fat i dine "låger" med javascript og lav indholdet sjovere og mere dynamisk

**Rammer og benspænd** Der er følgende benspænd og rammer for opgaven:

- Der skal være en splash screen som er animeret og som kører hver gang siden loader
- Siden skal have 24 "låger", dvs et eller andet man kan klikke på - og først derefter opleve indholdet
- Indholdet er frivilligt - men brug gerne en kombination af forskellige udtryk, fx billeder, tekst, musik, lyd, video, animation, api osv
- Der skal udvikles en XD skitse samt en kort "skriftlig plan" for hvordan du har tænkt dig at programmere siden - du skal med andre ord planlægge siden før du går i gang med at programmere

**Og ekstra udfordringer hvis du føler dig god..**

- Kalenderen skal sørge for at de låger der ligger før den dato man er på i december allerede er åbne
- Nogle af lågerne henter data fra et api - fx giphy.com ([https://api.giphy.com/v1/gifs/random?api\\_key=dc6zaTOxFJmzC&tag=christmas](https://api.giphy.com/v1/gifs/random?api_key=dc6zaTOxFJmzC&tag=christmas))
- Lav animationer eller lyde når man klikker på lågerne
- Man kune gåder som skulle løses før lågerne kan åbnes

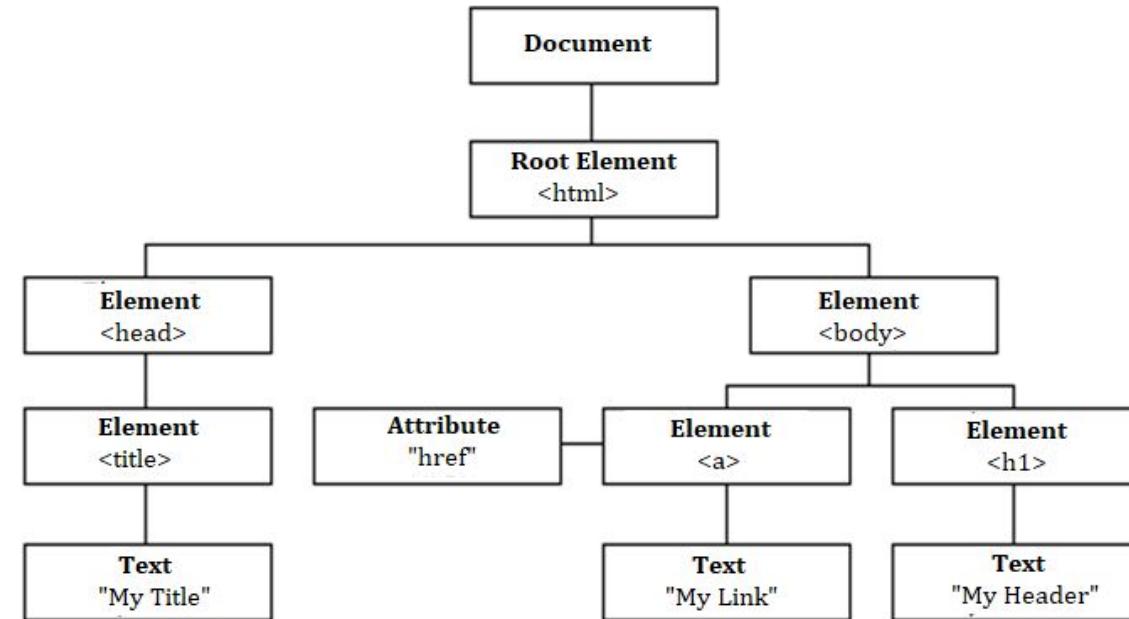
### Fremgangsmåde

- Lav først nogle papirskitser eller anden idéudvikling, hvor du overvejer hvad for en kalender du vil lave
- Lav en - lidt mere hifi - skitse i Adobe XD, hvor du viser hvordan en eller flere låger skal fungere
- Lav en liste med de **udfordringer** du ikke lige ved hvordan du skal løse, fx
  - Hvordan programmerer jeg en låge man kan klikke på med indhold bagved?
  - Hvordan holder jeg styr på de 24 låger, hvordan ved jeg hvilken man har klikket på?
  - Hvordan holder jeg styr på hvilken dag i december det er - så lågerne ikke skal åbnes to gange?
- Opret nogle små kodeeksempler hvor du løser udfordringerne en af gangen
- Til sidst er du så klar til at programmere selve kalenderen, fx ved at:
  - Lave din HTML struktur
  - Lave din datastruktur - hvis du har sådan en (lokal json fil fx)
  - Opret styles
  - Programmer - med trinvis forfining - din kalender

## HTML: DOM – document object model

JS – DOM, et interface imellem html-elementer & javascript-kode:

- add, change, and remove any of the HTML elements and attributes
- change any of the CSS styles
- react to all the existing events
- create new events



# INSPIRATION TIL PROGRESSIONSPLAN, FORLØBSPLAN

1. Javascript og webteknologi
  1. HTML, CSS og JAVASCRIPT
  2. Variabler, typer, lister, udtryk, kontrolstrukturer, funktioner
  3. Tid og dato
  4. Arrow functions
  5. Array - map, find, filter
  6. Hændelsesstyret interaktion I
2. Git og Github
  1. Hvad er git
  2. Hvad er github
  3. Hver elev har sit eget repository, som er blevet brugt til at dokumentere undervisningens progression og indhold. Vi er ikke gået i dybden med eksempelvis brach- og merge-funktionalitet
3. API og dataudveksling
  1. Hvad er et API
  2. At hente eksterne data med fetch
  3. Asynkron programmering
  4. Hvad er JSON
  5. JSON i lokale filer
  6. JSON fra et REST API
4. Javascript og biblioteket P5.JS (biblioteksmoduler)
  1. Hvad er P5.JS
  2. Introduktion til visuel programmering
  3. Hændelsesstyret interaktion II
  4. Hvordan kan objekter se ud i Javascript - vi er ikke gået i dybden med objektorienteret programmering, men har dækket de mest overordnede principper (klasse, constructor, scope, instantiering)
  5. Simpel spil- og visuel programmering i Javascript
  6. Parametrering/abstraktionsmekanismer, rekursion, polymorfi og algoritmemønstre
1. Arkitektur, netværk og prototyping (arbejdsgange, systematik, programmeringsbeskrivelser)
  1. Papirskitser og wireframes
  2. Flowcharts og overblik gennem digital skitsering
  3. Pseudokode og abstrakte programmeringsbeskrivelser
  4. Klient-Server arkitektur
  5. Netværk og protokoller (særligt MQTT og UDP)
  6. Test af web-apps - primært brugerorienterede, kvalitative testformer
2. Microcontroller forløb - M5 stack (udover kernestof)
  1. Hvad er ESP32
  2. Blokprogrammering
  3. Sensorer og data-mapping
  4. Kommunikation mellem microcontrollere og Javascript klient
3. Javascript og frameworket svelte.js
  1. Hvad er et framework
  2. Svelte.js basics
  3. Node.js introduktion
  4. Subkomponenter, bindings, reaktive variabler
  5. Byg en web-app med svelte.js og subkomponenter
4. Node.js serverarkitektur
  1. Node.js som serverarkitektur
  2. Websockets og klientkommunikation
  3. MQTT over websocket
  4. UDP protokollen - hurtig kommunikation mellem IOT/Server/klient

## **RESSOURCER TIL HTML, CSS, JS**

### **Intro til HTML**

[https://www.nemprogrammering.dk/Tutorials/HTML/oversigt\\_html.php](https://www.nemprogrammering.dk/Tutorials/HTML/oversigt_html.php)

### **INTRO til CSS**

[https://www.nemprogrammering.dk/Tutorials/CSS/oversigt\\_css.php](https://www.nemprogrammering.dk/Tutorials/CSS/oversigt_css.php)

### **CSS-GRID:**

[https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Grid\\_Layout](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout)

[https://www.w3schools.com/css/css\\_grid.asp](https://www.w3schools.com/css/css_grid.asp)

<https://css-tricks.com/snippets/css/complete-guide-grid/>

Samlet css-reference, (et monster): <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>

### **INTRO til JS**

<https://www.nemprogrammering.dk/Tutorials/javascript-v2/4-DOM.php> - god om DOM og JS