Webteknologi

Modul 6

d3.js

Dagens plan

- `08.30` Opfølgning på opgaver
- `08.45` Introduktion til d3.js
- `09.00` d3.js i praksis - med styling
- `10.00` *Pause*
- `10.30` d3.js + SVG + data
- `12.00` Frokost

d3.js-baggrund

Indledning

d3.js er et **JavaScript library**, dvs. det er kode man inkludere i sin egen JavaScript-kode. Det er samme princip som med den script.js-fil jeg tidligere har givet jer igennem min skabelon.

d3.js indeholder en masse funktionalitet, som helt grundlæggende kan

- transformere **data** til indhold i et HTML-dokument

Input: data (fx JSON med et JavaScript-array)

Output: indhold i HTML (fx SVG)

d3.js-baggrund

Herunder kan I se en masse eksempler på hvad der er muligt med d3.js. Det sidste link indeholder alle eksemplerne fra bogen.

https://d3js.org/

https://www.d3-graph-gallery.com/

https://webtek.krdo.dev/d3-book/

d3.js-baggrund

Sammenlignet med andre visualiserings-værktøjer, så er d3.js mere **low level**.

Det betyder at man arbejder på et lavere abstraktionsniveau. Man er tættere på data og visualisering, og skal gøre mere for at sætte tingene rigtigt sammen.

Det har både fordele og ulemper.

Fordelen ved at d3.js er mere low level:

- Man kan lave stort set alt.
- Meget få tekniske begrænsninger.
- Relativt simple byggeklodser kan sættes sammen på et utal af måder.

Ulempen ved at d3.js er mere low level:

- Ingen indbyggede visualiseringer. Man skal lave dem selv.
- Tager længere tid at blive god til d3.js.
- Der skal mere kode til at lave det samme som i et high level library.

Hvordan bruger man det?

Det er simpelt nok.

1. Lav et nyt HTML-dokument.
2. Download d3.js library og pak det ud. Så finder du en fil der hedder **d3.js**.
3. Læg d3.js-filen sammen med din html-fil.
4. Lav en script-reference til d3.js i din HTML.

Og nu kan du begynde at skrive din egen JavaScript-kode der bruger d3.js i HTML-dokumentet.

d3.js i praksis

Hvordan bruger man det?

Simpelt d3.js-kode

Her er et simpelt eksempel:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Eksempel</title>
    <script type="text/javascript" src="d3.js"></script>
  </head>
  <body>
    <script type="text/javascript">
      d3.select("body").append("p").text("Hej med jer!");
    </script>
  </body>
</html>
```

Og her er selve koden igen:

```
d3.select("body").append("p").text("Hej med jer!");
```

Koden er simpel. Den gør følgende:

- Via d3 vælges <**body**>
- På body tilføjes et nyt <**p**>
- På p sættes en ny text ("Hej med jer!").

Og det er det.

Eksemplet benytter sig dog ikke at data, blot en enkelt tekst, så det er ikke så spændende i praksis.

d3.js i praksis

Jeg vil udelade HTML-delen i de følgende eksempler. Alt koden skal skrives i **<script>** som med det første eksempel.

Her er noget kodet der får data som input:

```
const dataset = [ 5, 10, 15, 20, 25 ];

d3.select("body").selectAll("p")
  .data(dataset)
  .enter()
  .append("p")
  .text(function(d) {
    return d + " gange 100 er " + (d * 100)
  });
```

d3.js i praksis


```
const dataset = [ 5, 10, 15, 20, 25 ];

d3.select("body") // Vælg body
  .selectAll("p") // Vælg alle <p>
  .data(dataset)  // Vælg vores data
  .enter()        // Tilføj tom placeholder til hvert datapunkt
  .append("p")    // Tilføj <p> til alle placeholders
  .text(function(d) { // Udfyld teksten i hvert <p>
    // Parameter 'd' er et datapunkt fra arrayet
    // Vi bruger 'd' i output-teksten
    return d + " gange 100 er " + (d * 100);
  });
```

Hvis der allerede er nogle **<p>**, sørger **enter()** for kun at lave placeholder til dem der mangler.

d3.js i praksis

Samme eksempel som før, men med en **template string** som tekst.

```
const dataset = [ 5, 10, 15, 20, 25 ];

d3.select("body").selectAll("p")
  .data(dataset)
  .enter()
  .append("p")
  .text(function(d) {
    return `${d} gange 100 er ${d * 100}`
  });
```

Template strings er en smart feature i JavaScript hvor man kan indsætte data direkte i en string via **${}** syntaksen.

```
`${d} gange 100 er ${d * 100}`
```

I eksemplet herover indsættes parameteren **d**. Man kan indsætte komplekse udtryk, ikke blot variabler og værdier.

Bemærk, at citationstegnene i template strings er dem man kalder **backticks**: ` De skal pege "bagud".

Og hvis du undrer dig over d3.js-syntaksen, så kaldes den for **method chaining**:

```
d3.select("body").append("p").text("Hej med jer!");
```

Forklaret med ord: d3 er et objekt med metoder på. Metoden select() returnerer et nyt objekt. På de nye objekt findes også metoder, bl.a. append(), og den returnerer også et nyt objekt. På det nye objekt findes metoden text, osv.

Det giver en lang kæde at **metodekald**, og det er sådan d3.js fungerer.

Styling

Hvordan man styler

Styling med d3.js er lige ud af landevejen.

Man kan bruge metoden **style()** på de elementer man opretter, og her kan man vælge en **attribut** der skal styles.

Se eksemplet på næste slide.

Styling

Hvordan man styler

Kode (simpel)

```
const dataset = [ 5, 10, 15, 20, 25 ];

d3.select("body").selectAll("p")
  .data(dataset)
  .enter()
  .append("p")
  .text(function(d) {
    return `${d} gange 100 er ${d * 100}`
  })
  // Og her laves styling:
  .style("color", "red");
```

Styling

Hvordan man styler

Kode (simpel)

Kode (styling som funktion)

```
const dataset = [ 5, 10, 15, 20, 25 ];

d3.select("body").selectAll("p")
  .data(dataset)
  .enter()
  .append("p")
  .text(function(d) {
    return `${d} gange 100 er ${d * 100}`
  })
  // Styling som funktion med datapunkt som input
  .style("color", function(d) {
    if (d > 15) {
      return "red";
    } else {
      return "black";
    }
  });
```

Styling

Med d3.js kan man bruge **callback-funktioner** som værdier. En callback-funktion er en funktion der kaldes senere - i dette tilfælde når d3.js tegner. Med funktionen kan vi styre hvordan en bestemt værdi ser ud. Her afgør vi fx om en style skal være **red** eller **black** alt efter om datapunktet er over eller under 15.

```
.style("color", function(d) {
  if (d > 15) {
    return "red";
  } else {
    return "black";
  }
}}
```

Det så måske lidt mærkeligt ud at man kan give en funktion som input til en anden funktion, men det er helt normalt i JavaScript og inden for det paradigme der hedder **funktionel programmering**.

I så måske også at funktionen ikke havde noget navn. Så kaldes det for en **anonym funktion**.

Se mere om callbacks her: https://javascript.info/callbacks.

Det kaldes også en asynkron funktion, fordi den første bliver brugt **senere** (altså asynkront i forhold til resten af koden).

Øvelser

Start med opgave 1 - 4.

Resten af opgaverne gemmer vi til efter pausen.

d3.js + SVG

Baggrund

Slutmålet for i dag er at sætte d3.js og SVG sammen.

Så kan i bruge jeres viden om SVG sammen med d3.js til at genererer visualiseringer ud fra data. Nice!

Eksemplet i dag er inspireret af bogen. Lad os lave en simpel **barchart**.

d3.js + SVG

Baggrund

Kode: Data

Først noget data:

```
// Width and height
const w = 500;
const h = 100;
const barPadding = 1;

const dataset = [ 5, 10, 13, 19, 21, 25, 22, 18, 15, 13,
  11, 12, 15, 20, 18, 17, 16, 18, 23, 25 ];
```

d3.js + SVG

Så laves et SVG-element

```
// Create SVG element
const svg = d3.select("body")
  .append("svg")
  .attr("width", w)
  .attr("height", h);
```

d3.js + SVG

Og nu laver vi **firkanter** (rects):

```
svg.selectAll("rect")
   .data(dataset)
   .enter()
   .append("rect")
   .attr("x", function(d, i) {
      return i * (w / dataset.length);
   })
   .attr("y", function(d) {
      return h - (d * 4);
   })
   .attr("width", w / dataset.length - barPadding)
   .attr("height", function(d) {
      return d * 4;
   })
   .attr("fill", "teal");
```

d3.js + SVG

Da dette slideshow er skrevet i HTML + JavaScript, kan jeg nemt outputtet resultatet herunder med den kode i så.

Kør koden!

At loade JSON-data

```
// Nu henter vi data
d3.json("JEOPARDY_QUESTIONS.json").then(function(data) {
  console.log("Der er hentet data:");
  console.log(data);

  // TODO: Skriv din d3-kode til visualisering her

});
```

Dette er den nemmeste måde at loade data på. Visualiseringen skal foregår inde i den funktion, som defineres i **then**, som også er en callback-funktion.

Se **skabelon-jeopardy** i GitHub-repositoriet for det fulde eksempel.