

Attention, quand on travaille avec des données server, cela se lance avec
npm start

Noémie Romano
VisualDon
Cours 01
<https://01-intro.surge.sh/#/1>

Visualisation de données

1. Introduction + SVG

Noemi Romano

noemi.romano@heig-vd.ch

Qu'est-ce que la visualisation de données ?

Représentation graphique (**visualisation**) de l'information (**données**)

Pourquoi la visualisation de données ?

Explorer: trouver les tendances

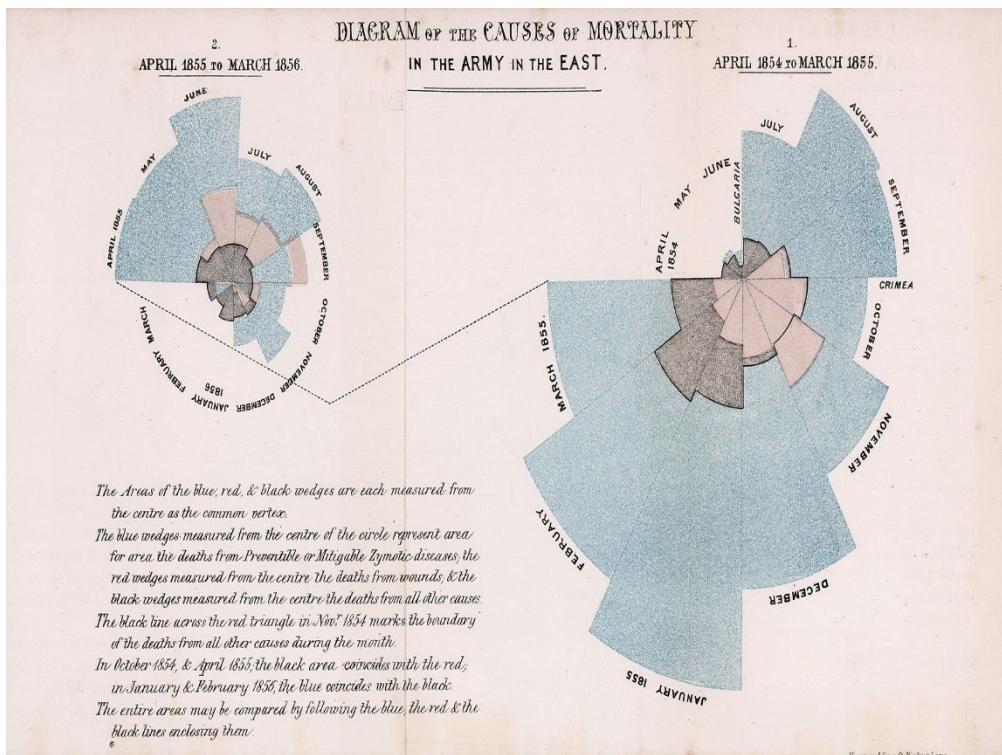
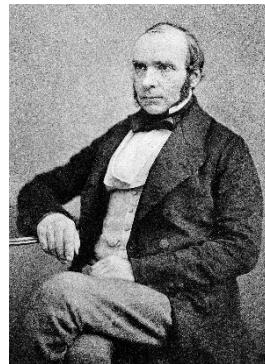
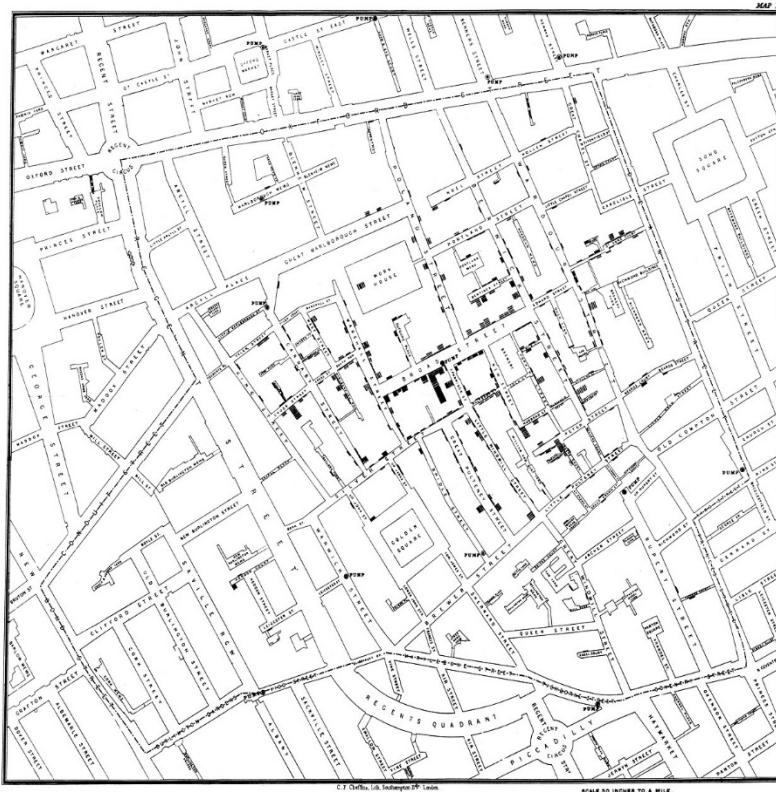


Diagram of the causes of mortality in the army in the East, Florence Nightingale, 1858

Analyser: tester une hypothèse

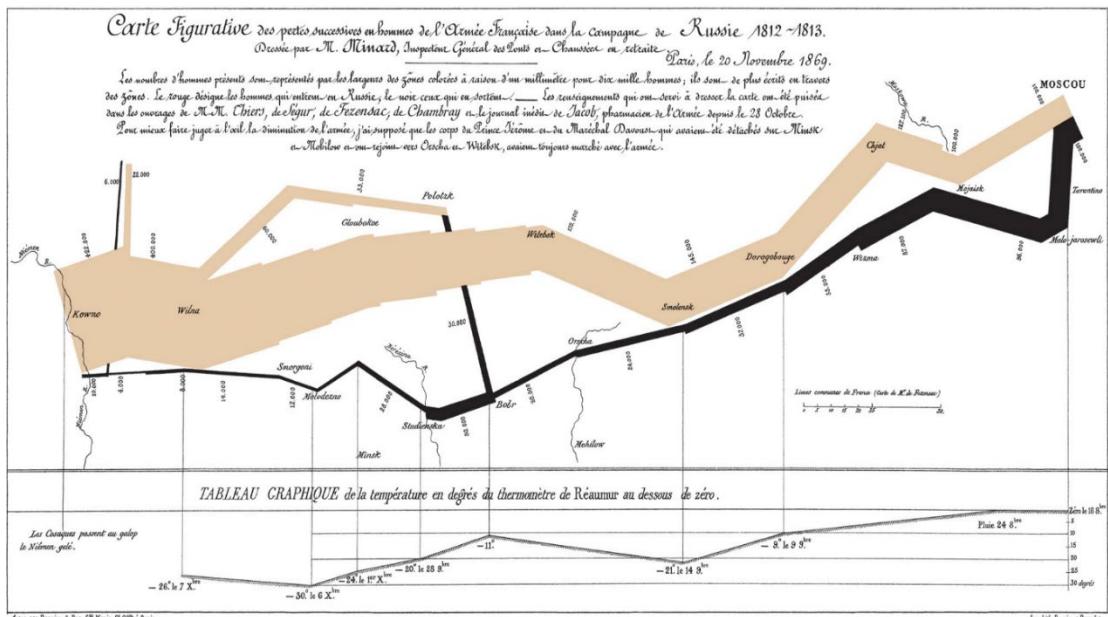




L'épidémie choléra de Broad Street, John Snow, 1854

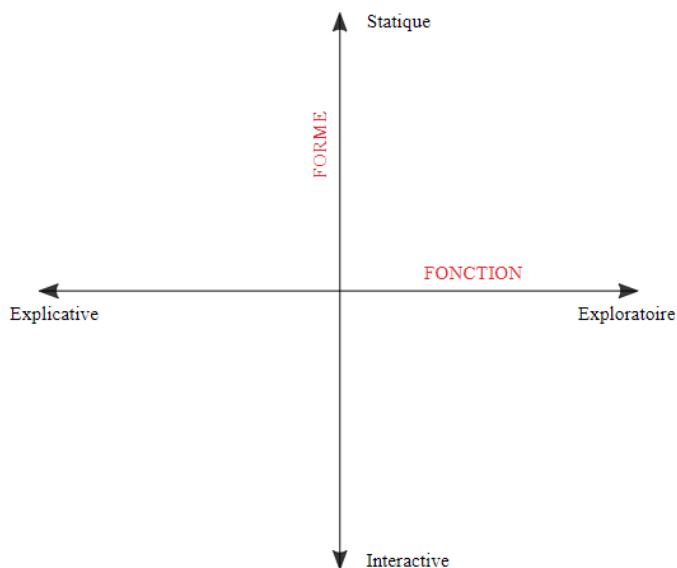
Présenter: raconter une histoire





La Carte figurative des pertes successives en hommes de l'Armée française dans la campagne de Russie en 1812-1813, Charles Joseph Minard, 1869

Fonctions et formes de visualisation

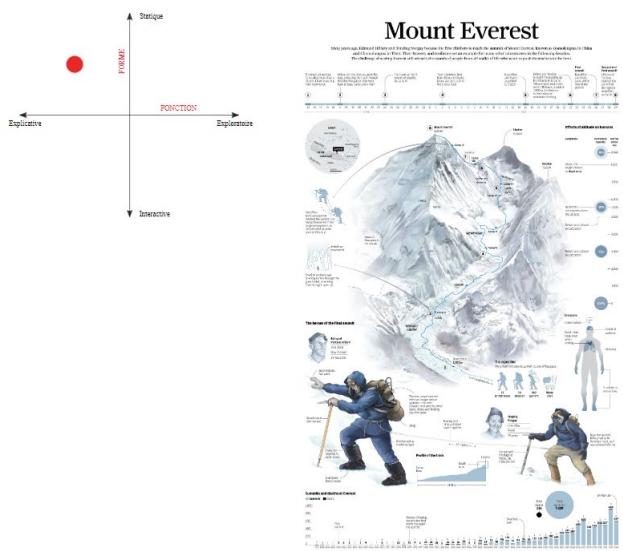


Interactive : on laisse l'utilisateur explorer l'histoire

Statique : Ne bouge pas, comme le papier.

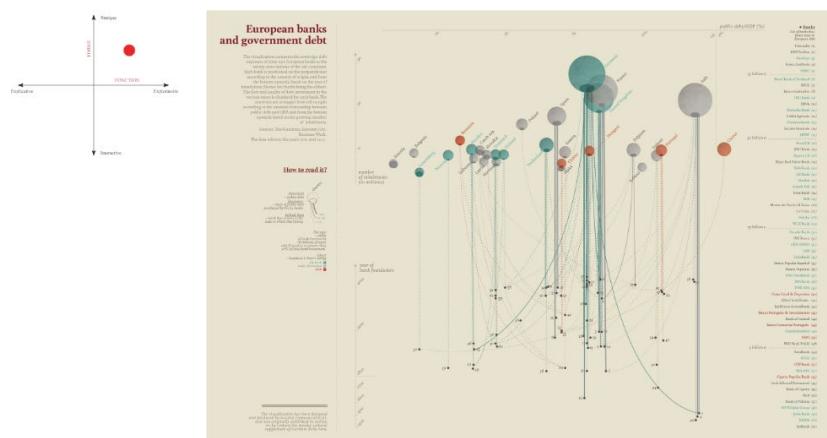
Explicative : On explique l'histoire

Exploratoire : On explique rien, on laisse à l'utilisateur comprendre ce que c'est. A lui d'aller lire ce que c'est, trouver les informations.



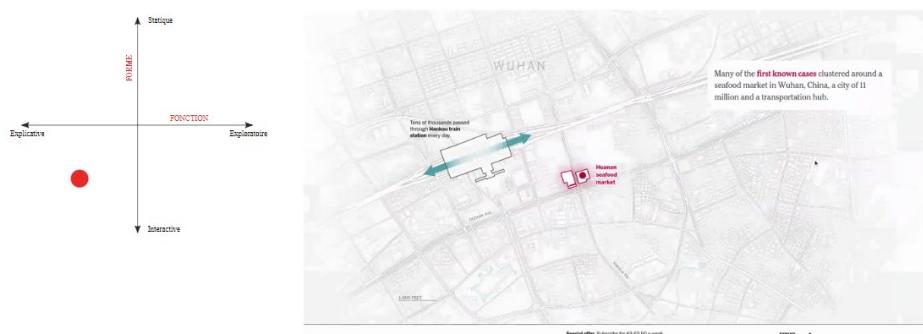
Mount Everest, South China Morning Post

Statique – Explicative



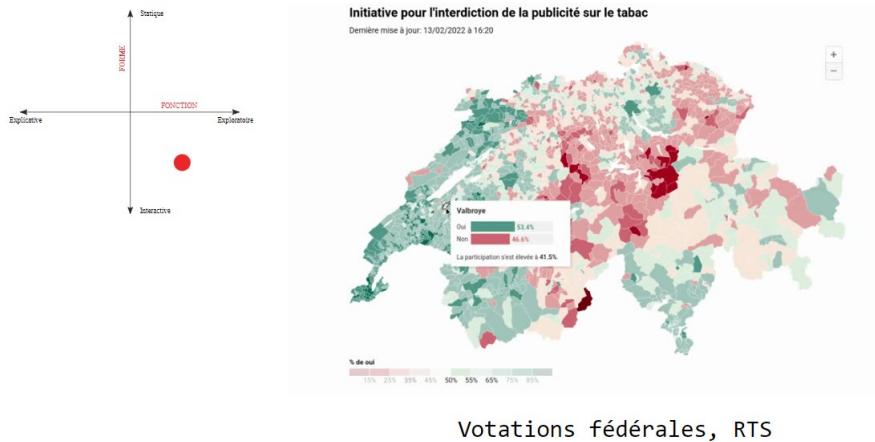
Banques européennes et dette, Giorgia Lupi

Statique - Exploratoire

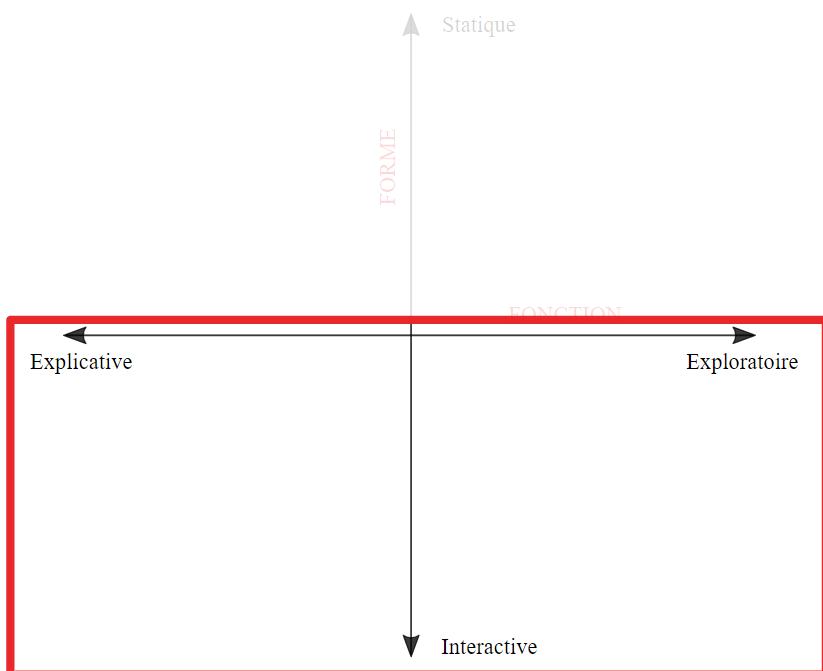


How the virus got out, New York Times

Interactif - explicatif

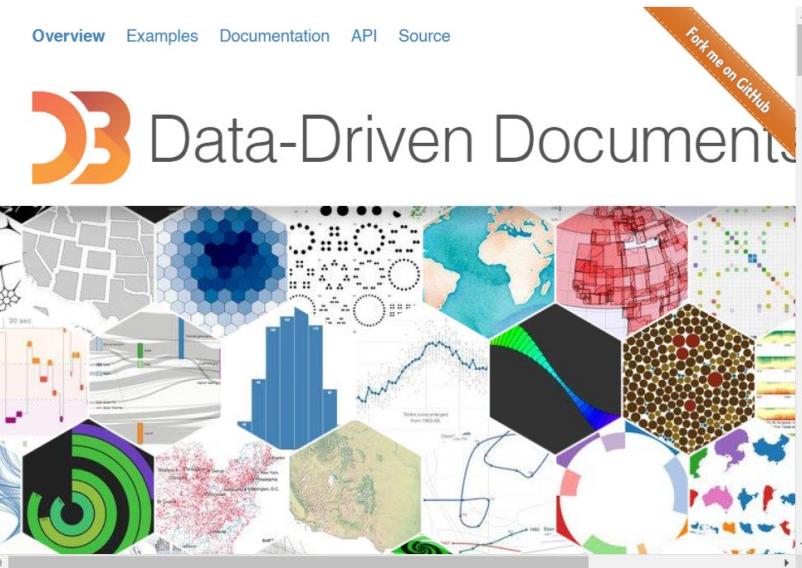


Interactif - Exploratoire



Nous on va plutôt être dans l'interactif exploratoire.

Quels sont les outils de visualisation de données interactive ?



D3 a été inventé par un journaliste, pour avoir quelque chose de plus interactif, de plus amusant. Quelque chose de visuel qui retienne plus l'attention. Il voulait créer des graphismes de manières automatiques.

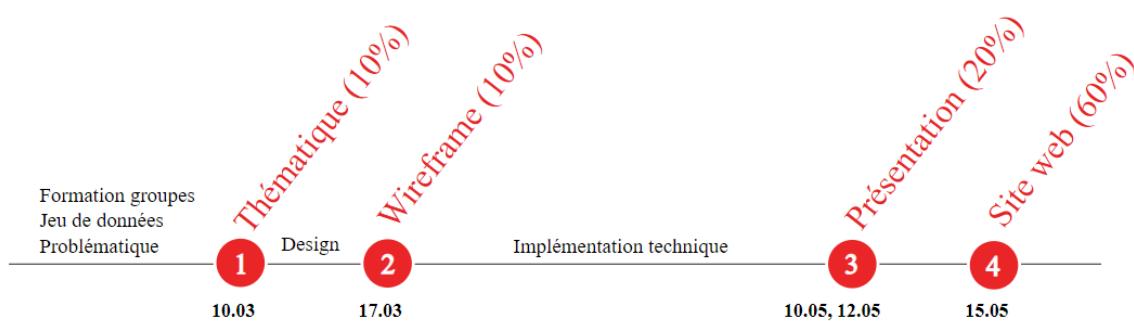
Objectifs

- Connaître différentes technologies de visualisation de données dans un navigateur Web
- Maîtriser la librairie d3.js
- Connaître une librairie de cartographie web
- Maîtriser la manipulation des données avec JS

Evaluation



Projet



Wireframe, à rendre comme on veut, à la main, en programmation, sur figma. Il faut laisser libre court à son imagination. Ne pas mettre de limite dans ce que l'on veut faire. On est pas obligé de faire exactement ce que l'on aura écrit dans le wireframe, il y aura des changements au fur et à mesure.

Supports :

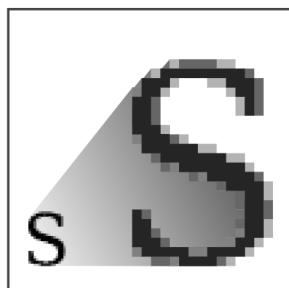
Cours : <https://github.com/romane/visualdon-cours#programme-de-cours>
Exercices : <https://github.com/romane/visualdon-exercices>

D3 & SVG

*D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, **SVG** and CSS*

SVG

Scalable **V**ector **G**raphics
(*graphique vectoriel adaptable*)



Matriciel

.jpeg .gif .png



Vectoriel

.svg

Syntaxe

<svg attributs>

Vos dessins ici

</svg>

Attributs

width [px]

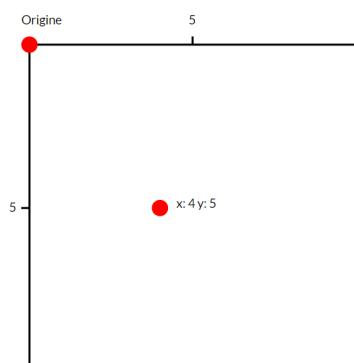
Largeur

height [px]

Hauteur

Et bien d'autres !

Système de coordonnées



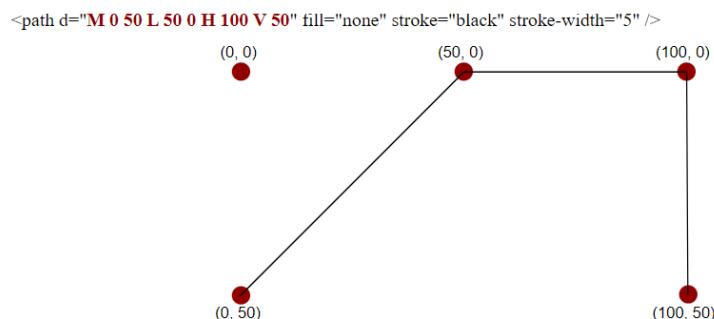
L'axe des X et Y est inversé, donc faire attention à ça.

Formes et textes



Path

Move to (M x y)
Line to (L x y)
Horizontal line (H x)
Vertical line (V y)



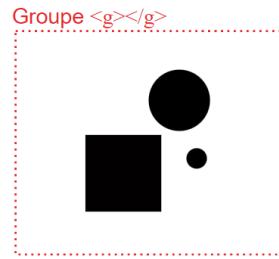
Pour chemin en français.

Toujours se rappeler qu'on part de 0, 0

M : « on lève le stylo et on le pose ailleurs sur la feuille. »

Groupe

```
<g>
<circle cx="150" cy="50" r="30" />
<rect x="100" y="60" width="50" height="50" />
<circle cx="160" cy="55" r="10" />
</g>
```

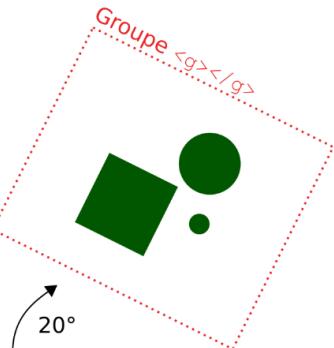


Même chose que sur Illustrator, **ctrl + g**

Transformation

```
translate(x,y)
rotate(angle, cx, cy)
scale(facteur_echelle)
```

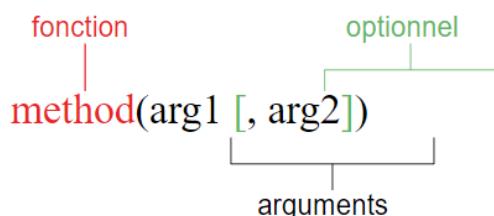
```
<g transform="rotate(20, 100, 50)" fill="green">
<circle cx="150" cy="50" r="30" />
<rect x="100" y="60" width="50" height="50" />
<circle cx="160" cy="55" r="10" />
</g>
```



Sur starckoverflow les mises à jours ne sont pas forcément faites car il y a peu de documentation sur D3

Documentation

Github: API reference



Ecriture classique de JS. **Si on a des crochets, c'est que ce sont des arguments optionnels**

Syntaxe

Chaînage de méthodes (fonctions)

```
objet.methode1()  
    .methode2()  
    .methode3()
```

```
d3.select("body")  
    .append("p")  
    .text("Nouveau paragraphe")
```

D3 on l'écrit avec cette syntaxe. La syntaxe on a un objet, une sélection et on va lui chaîner des méthodes.

d3-selection

Installation

```
npm install d3-selection
```

Tout ce qui permet de sélectionner des objets dans notre navigateur

Sélectionner

select(selector)
selectAll(selector)

	CSS	Exemple
type	element	.select('h1')
class	.class	.select('.class')
identifiant	#id	.select('#id')

↳ [d3-selection - Selecting elements](#)

Cette méthode permet de sélectionner l'élément dans la division que l'on vient de créer. (c'est un peu les sélectionneurs CSS)

On peut faire un parallèle avec *get element by ID*

Modifier

`selection.attr(name[, value])`

```
<!--index.html-->
<svg id = 'mon-svg'>
  <circle cx="50%" cy="50%" r="50"></circle>
</svg>
```



```
// index.js
import { select } from 'd3-selection';

const svg = select("#mon-svg");
const cercle = svg.select("circle");
cercle.attr("fill", "#E92528");
```

↳ [d3-selection - Modifying elements](#)

La méthode de **D3 sélection** permet aussi de modifier les éléments, les attributs de certains éléments.

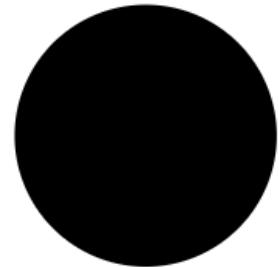
On sélectionne l'attribut, puis le cercle puis son attribut

Ajouter

`selection.append(type)`

```
const WIDTH = 500
const HEIGHT = 800

const div = select("#mon-svg")
    .append("svg")
    .attr("width", WIDTH)
    .attr("height", HEIGHT)
    .append("circle")
    .attr("cx", "30%")
    .attr("cy", "40%")
    .attr("r", "100")
```



↳ [d3-selection - Modifying elements](#)

Méthode native de JS. On a notre sélection et on appelle un type.
Notre navigateur peut être vide mais on peut lui append plein de choses. Permet de rajouter plein de choses à notre document.

Ecouter les événements

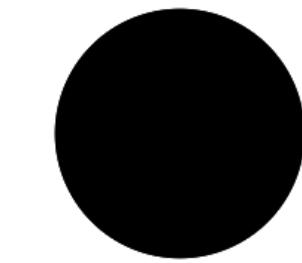
```
selection.on(typenames[, listener[, options]])
```

```
<!--index.html-->
<svg id = 'mon-svg'>
  <circle cx="50%" cy="50%" r="50"></circle>
</svg>

// index.js
import { select } from 'd3-selection';

select("#mon-svg")
  .select("circle")
  .on('click', function () {
    select(this).attr("fill", 'green')
  })
  .on('mousemove', function(e) {
    console.log("x: " + e.clientX + ", y:" + e.clientY )
  });

```



x: 0, y: 0

↳ [d3-selection - Handling events](#)

Je sélectionne le cercle à l'intérieur, en clic.

Tous ces événement seront déclencher quand je suis sur le cercle et pas ailleurs. Ils seront déclencher quand je suis sur l'élément, quand l'évènement se produit.

Joindre les données

Joindre les données aux éléments sélectionnés

```
selection.data([data[, key]])
```

```
selection.data(tableau).join(enter[, update][, exit])
```

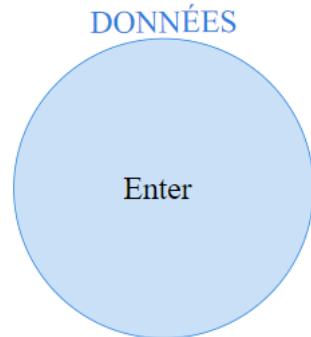
Le plus de D3, permet de joindre des données à un SVG.

Dans data il y a le tableau de données. Mais il ne sait pas trop quoi faire avec. Donc on lui dit de joindre (avec .join) ces données.

Joindre les données

Entering data

```
selection.data(données)
  .join(enter => enter
        .append('element')
        .attr('attribute', d => d.valeur)
  );
```



C'est dans la méthode enter que l'on va faire les changements.

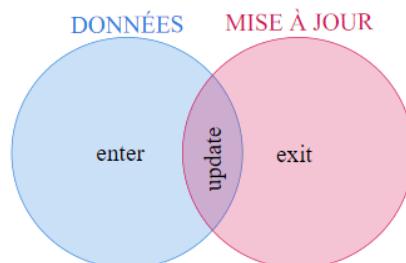
Joindre les données

Mise à jour des données

enter, update, exit

```
selection.data(données)
  .join(enter => enter
        .append('element')
        .attr('attribute', d => d.valeur),
  update => update
        .append('element')
        .attr('attribute', d => d.valeur),
  exit => exit
        .remove()
  )

.append('element') // Appliqué en tout état (enter, update, exit)
.attr('attribute', d => d.valeur), // Appliqué en tout état (enter, update, exit)
```



Enter : données qu'on a et qui existe

Join : on évalue tous les états des données

Update : pris en compte et les données updaté sont affichées grâce au .append (les éléments qui sont déjà dans la sélection)

Enter : données qui vont être appliquées quand il y a de nouveaux éléments

Exit : ce qui sort

Joindre les données

EXEMPLE

```
let lettres = tableauLettresAleatoires();  
  
svg.selectAll('text')  
  .data(lettres, d => d)  
  .join(enter => enter  
    .append('text')  
    .attr('fill', 'green')  
    .text(d => d)  
  )  
  .attr("x", (d, i) => i * 15) // Mettre à jour  
    à chaque fois  
    la position
```

d o p s t u v

.data : joindre

Enter : dessine ces lettres

Update : « tout ce qu'il y a déjà dedans, tu le changes en gris »

Joindre les données

EXEMPLE

```
let lettres = tableauLettresAleatoires();  
  
svg.selectAll('text')  
  .data(lettres, d => d)  
  .join(enter => enter  
    .append('text')  
    .attr('fill', 'green')  
    .text(d => d),  
  update => update  
    .attr('fill', 'grey')  
  )  
  .attr("x", (d, i) => i * 15) // Mettre à jour  
    à chaque fois  
    la position
```

d e g i k l m o p q t v w z

Nouveau : en vert, déjà présenter en gris

Update : a déjà le dessin, il n'aura que la mise à jour.

Joindre les données

EXEMPLE

```
let lettres = tableauLettresAleatoires();  
  
svg.selectAll('text')  
  .data(lettres, d => d)  
  .join(enter => enter  
    .append('text')  
    .attr('fill', 'green')  
    .attr("x", (d, i) => i * 15)  
    .attr('y', 0)  
    .text(d => d),  
  update => update  
    .attr("fill", "grey")  
    .attr("y", 0)  
    .attr("x", (d, i) => i * 15)  
  exit => exit  
    .attr("fill", "brown")  
    .transition(t)  
    .attr("y", 30))
```

b c d g i m p r v w y

La méthode `exist` permet de voir tout ce qui sort

Joindre les données

```
<!--index.html-->  
  
<div id="my-div"></div>  
  
// index.js  
  
import { select, selectAll } from 'd3-selection';  
  
const data = ['Pierre', 'Charlotte', 'Jacques'];  
  
select("#ma-div")  
  .append("ul")  
  .selectAll("li")  
  .data(data)  
  .join((enter) => enter  
    .append("li")  
    .text((d, i) => `Valeur: ${d}, Index: ${i}`))  
  .style("color", "green");
```

- Valeur: Pierre,
Index: 0
- Valeur: Charlotte,
Index: 1
- Valeur: Jacques,
Index: 2

↳ [d3-selection - Joining data](#)

Exemple simple.

On peut utiliser D3 pour générer du contenu dynamique.

d : élément où on est dans le tableau (valeur du tableau)

i : index

selectAll : on sélectionne qqch qui n'existe pas encore. Mais c'est très important car sinon cela ne rajoute pas les nouveaux éléments au bon élément existant.

Projet

Thématique (10 mars)

- Trouver un jeu de données
- Créer un dossier github pour le projet
- Suivre directives [ici](#)



Types de données

Qualitatives

- Nominales
- Ordinales

Quantitatives

- Discrètes
- Continues



Qualitative :

Données qui s'écrivent comme des mots mais qui présente un ordre

- **nominale** : ex. Prénom
- **Ordinal** : ex. le 1^{er}, le 2^{ème}

Quantitative :

- **Discrètes** : ex. on ne peut pas être 35.5 dans une classe
- **Continues** : taille, poids, peut prendre un nombre infini

CSV

(Comma-Separated Values)

```
prenom,anne_naissance  
Alphonse,1932  
Béatrice,1964  
Charlotte,1988
```

JSON

(JavaScript Object Notation)

```
[  
  {  
    "prenom" : "Alphonse",  
    "annee_naissance" : 1932  
  },  
  {  
    "prenom" : "Béatrice",  
    "annee_naissance" : 1964  
  },  
  {  
    "prenom" : "Charlotte",  
    "annee_naissance" : 1988  
  }]
```

Dynamique, pour le web, ce que l'on va travailler le plus

TSV

(Tab-Separated Values)

```
prenom      anne_naissance  
Alphonse    1932  
Béatrice    1964  
Charlotte   1988
```

XML

(eXtensible Markup Language)

```
<xml version="1.0" encoding="UTF-8"?>
<Personne>
  <personne id="1">
    <prenom>Alphonse</prenom>
    <anne_naissance>1932</anne_naissance>
  </personne>
  <personne id="2">
    <prenom>Béatrice</prenom>
    <anne_naissance>1964</anne_naissance>
  </personne>
  <personne id="3">
    <prenom>Charlotte</prenom>
    <anne_naissance>1988</anne_naissance>
  </personne>
</Personne>
```

Charger les données

d3-fetch

Installation

```
npm install d3-fetch
```

D3 construit au-dessus de la méthode fetch.

CSV

```
csv(input[, init][, row])
```

JSON

```
json(input[, init][, row])
```

```
import { csv } from "d3-fetch";
csv("chemin/du/fichier.csv")
  .then( function(data) {
    // Dessiner ici
  })
  .catch(function(error){
    // Gérer les erreurs ici
  })
```

```
import { json } from "d3-fetch";
json("url/ou/chemin/du/fichier.json")
  .then( function(data) {
    // Dessiner ici
  })
  .catch(function(error){
    // Gérer les erreurs ici
  })
```

On charge les données, on dessine, s'il y a une erreur, on les cache.

Statistiques

d3-array

Installation

```
npm install d3-array
```

Statistiques

max(iterable[, accessor])
min(iterable[, accessor])
sum(iterable[, accessor])
extent(iterable[, accessor])
mean(iterable[, accessor])

```
import { max, min, sum, extent, mean } from "d3-array";
const data = [5, 10, 4, 25];
const maxValue = max(data); // Expected output: 25
const minValue = min(data); // Expected output: 5
const sumValues = sum(data); // Expected output : 44
const extentValues = extent(data); // Expected output : [4, 25]
const meanValues = mean(data); // Expected output : 11
```

↳ [d3-array - Statistics](#)

Manipuler les données avec Javascript (ES7)

Array.prototype.map()

crée un nouveau tableau avec les résultats de l'appel d'une fonction qui s'applique à l'ensemble des données

```
const data = [1, 4, 9, 16];
// Pass a function to map
const dataMapped = data.map(x => x * 2); // Expected output: Array [2, 8, 18, 32]
```

[↳ Doc MDN : Array.prototype.map\(\)](#)

Permet d'appliquer une fonction à tous les types de données. Permet d'itérer dans chaque ligne et de donner une fonction à chaque ligne.

Array.prototype.filter()

filtre les données selon une condition

```
const words = ["spray", "limit", "elite", "exuberant", "destruction", "present"];
const result = words.filter(word => word.length > 6); // Expected output: Array ["exuberant", "destruction", "present"]
```

[↳ Doc MDN: Array.prototype.filter\(\)](#)

Sort un tableau qui respecte une certaine condition.

Array.prototype.includes()

vérifie si les données contiennent une valeur spécifiée

```
const array1 = [1, 2, 3];
console.log(array1.includes(2)); // Expected output: true

const pets = ["cat", "dog", "bat"];
console.log(pets.includes("cat")); // Expected output: true

console.log(pets.includes("at")); // Expected output: false
```

↳ [Doc MDN: Array.prototype.includes\(\)](#)

Permet de vérifier si un élément existe ou pas dans un tableau, donc retourne vrai ou faux. Ça fonctionne pour un match exacte.

Array.prototype.pop()

supprime le dernier élément d'un tableau

```
const plants = ["broccoli", "cauliflower", "cabbage", "kale", "tomato"];
console.log(plants.pop()); // Expected output: "tomato"

console.log(plants); // Expected output: Array ["broccoli", "cauliflower", "cabbage", "kale"]

plants.pop();
console.log(plants); // Expected output: Array ["broccoli", "cauliflower", "cabbage"]
```

↳ [Doc MDN: Array.prototype.pop\(\)](#)

Enlève le dernier élément d'un tableau

Array.prototype.push()

ajoute un ou plusieurs éléments à la fin d'un tableau

```
const animals = ["pigs", "goats", "sheep"];
const count = animals.push("cows");
console.log(animals); // Expected output: Array ["pigs", "goats", "sheep", "cows"]

animals.push("chickens", "cats", "dogs");
console.log(animals); // Expected output: Array ["pigs", "goats", "sheep", "cows", "chickens", "cats", "dogs"]
```

↳ [Doc MDN: Array.prototype.push\(\)](#)

Rajoute à la fin du tableau un élément souhaité

Array.prototype.reduce()

applique une fonction qui est un « accumulateur » et qui traite chaque valeur d'une liste (de la gauche vers la droite) afin de la réduire à une seule valeur

```
const array1 = [1, 2, 3, 4];
// 0 + 1 + 2 + 3 + 4
const initialValue = 0;
const sumWithInitial = array1.reduce(
  (accumulator, currentValue) => accumulator + currentValue,
  initialValue
);
console.log(sumWithInitial); // Expected output: 10
```

↳ [Doc MDN: Array.prototype.reduce\(\)](#)

Permet de faire des opérations sur un tableau de données. C'est un accumulateur.

Array.prototype.forEach()

exécute une fonction donnée sur chaque élément du tableau

```
const array1 = ["a", "b", "c"];
array1.forEach(element => console.log(element));
// Expected output: "a"
// Expected output: "b"
// Expected output: "c"
```

↳ [Doc MDN: Array.prototype.forEach\(\)](#)

Permet d'itérer à partir d'un tableau sur chacun des éléments.

d= données

i = itération

(s'il n'y en a qu'un, ce sont les données) On en mai 2 pour éviter que la fonction ne prenne les valeur du tableau.

Exercices

Télécharger changements

`git fetch upstream`

Mettre à jour

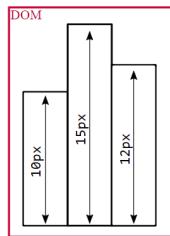
`git merge upstream/main`

Echelle : facteur qui permet de mettre à l'échelle les données (D3 scale)

Dès aujourd'hui

Données = Echelle * Pixels

```
const data = [100,150,120]
const scale = 0.1
```



d3-scale

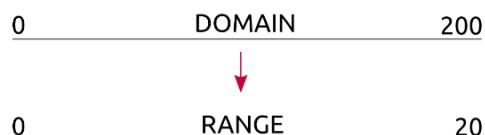
Installation

```
npm install d3-scale
```

Syntaxe

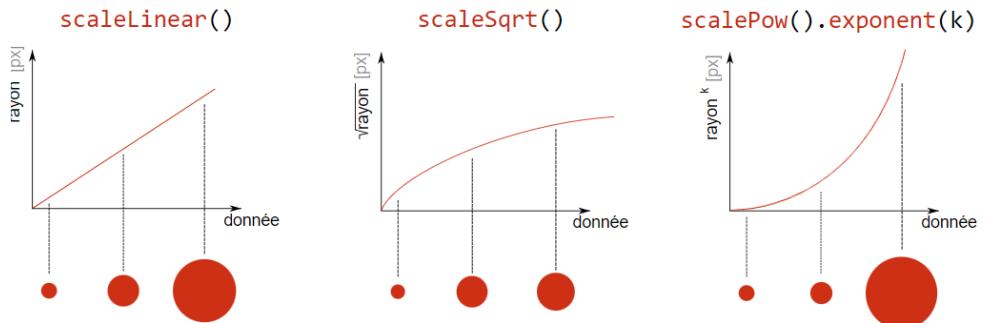
```
typeEchelle
  .domain(intervalleEntrée) // [minData, maxData]
  .range(intervalleSortie) // [0, svgWidth]
```

```
const scale = scaleLinear()
  .domain([0,200])
  .range([0,20])
```



Échelles continues

Données continues



```
const scale = scaleLinear()
  .domain([0, 100])
  .range([0, 1000])

scale(0) // Expected output 0
scale(50) // Expected output 500
scale(100) // Expected output 1000
```

D3 calcule aussi en dehors de l'échelle qu'on lui a donné.

Ici l'échelle se construit en prenant en compte le premier et le dernier jour.

Échelles continues

Données temporelles

scaleTime()

[dateDebut, dateFin] → [0, widthSvg]

domain

range

```
const firstDay = new Date(new Date().getFullYear(), 0, 1);
const lastDay = new Date(new Date().getFullYear(), 11, 31);
const christmasDay = new Date(new Date().getFullYear(), 11, 25);

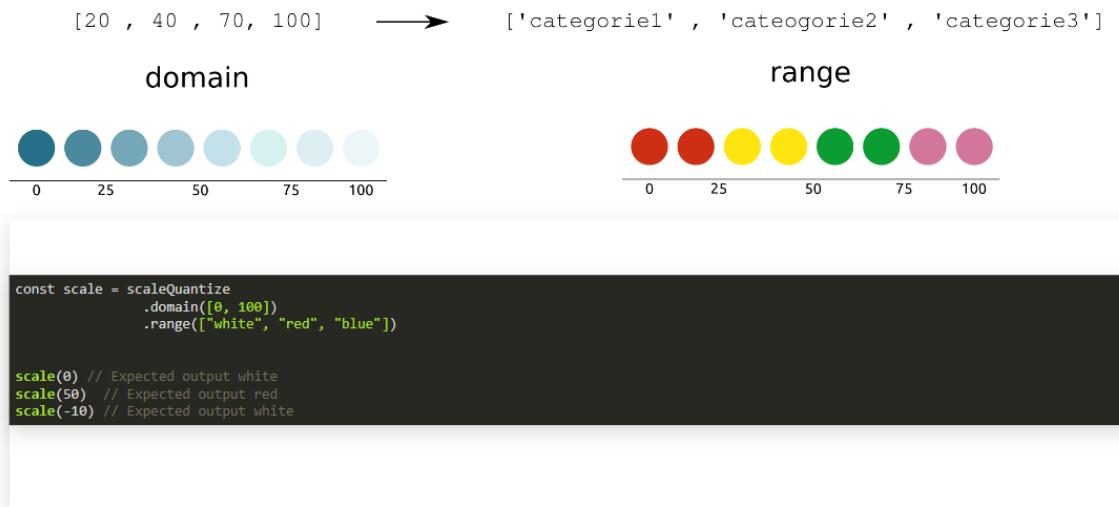
const scale = scaleTime()
  .domain([firstDay, lastDay])
  .range([0, 100])

scale(firstDay) // Expected output 0
scale(lastDay) // Expected output 100
scale(christmasDay) // Expected output 98.35164835164835
```

Échelles discrètes

Données continues

scaleQuantize()



ScaleQuantize : divise par le nombre de catégorie qu'il y a. Classe les données continues à intervalles égales.

Il comprend tout seul et associe les données.

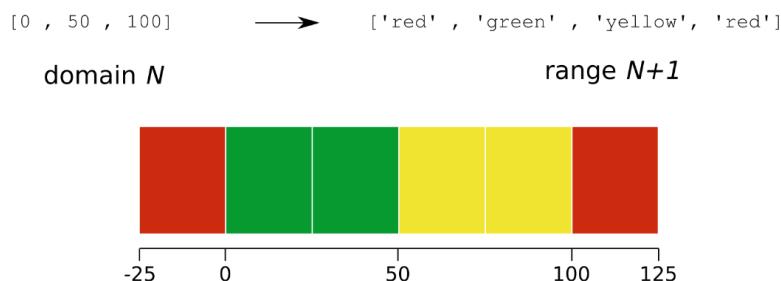
Pratique si on a des données continues mais qu'on veut colorer par une classe particulière.

Exemple graphique pour montrer qu'on passe du continu au discret.

Échelles discrètes

Données continues

scaleThreshold()



scaleThreshold : On définit nous-même nos intervalles

Quand c'est plus petit ça reste rouge, entre 2 valeur ça change de couleur (exemple avec le vert) etc.

Échelles discrètes

Données discrètes

scaleOrdinal()

["Lundi", "Mardi", "Mercredi"] → ['red', 'blue', 'green']

domain

range

Lundi Mardi Mercredi

Lundi Mardi Mercredi

Prend le tableau initiale et va mettre les classes de manière ordonnées.

Échelles discrètes

Données discrètes

scaleBand()

["Lundi", "Mardi", "Mercredi"] → [0, swgWidth]

domain

range

Lundi Mardi Mercredi

Lundi Mardi Mercredi

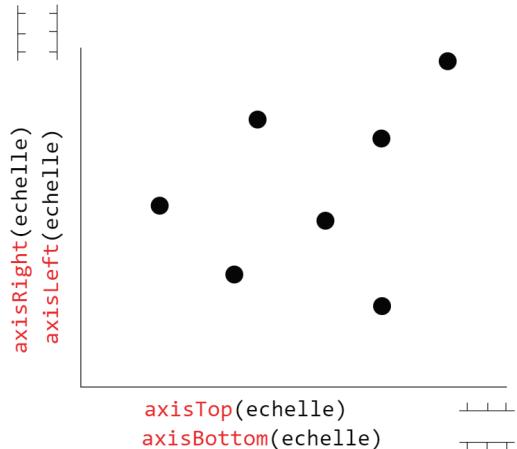
On va mapper ces données discrètes dans un domaine continu.

d3-axis

Installation

npm install d3-axis

Axes



Les axes servent à mapper de manière les données continues le visuel.
Si on doit dessiner un axe, il faut qu'il respecte l'échelle qu'on a donné avant.

Axes

Création

```
const axe =  
  axisBottom(echelle)
```

Dessin

```
selecteur  
.append('g')  
.call(axe)
```

Création d'un axe

.**call** : appelle l'axe qu'on vient de créer

On a besoin d'un groupe (**g**) car D3 va créer pleins de petits éléments donc il faut les grouper pour qu'on puisse les manipuler. Pour pouvoir faire des fonctions qui sont applicables à un groupe et pas à chaque élément.

L'axe des X et Y sont donc séparés.

Marges & Translations

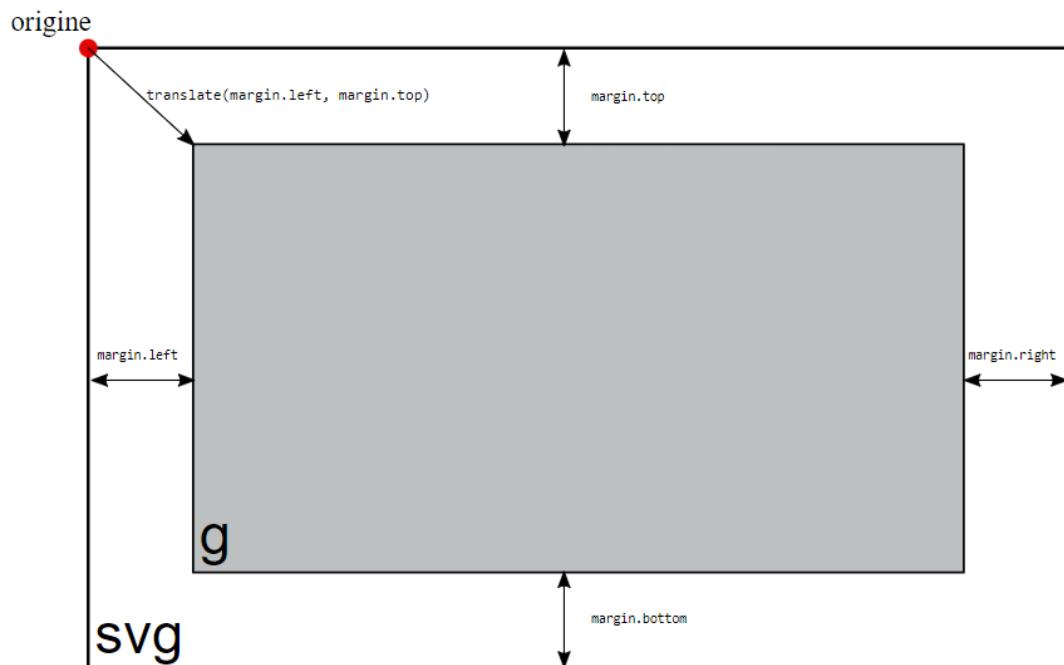


DIAGRAMME EN BATON

<https://codepen.io/romanoe/pen/oNPGzyY>

Exemple de diagramme en bâton.

```
1 // Marges et translations
2 const margin = { top : 10, right: 40, bottom: 30,
3   left: 40 },
4   width = 450 - margin.left - margin.right,
5   height = 400 - margin.top - margin.bottom;
6
7
7 // Données
8 const myData = [
9   { day : 'Mon', value: 230 },
10  { day : 'Tue', value: 40 },
11  { day : 'Wed', value: 100 },
12  { day : 'Thu', value: 60 },
13  { day : 'Fri', value: 3 }
14 ];
```

Il est conseiller de toujours définir les constantes

```

25 // Echelle
26 const bandScale = d3.scaleBand()
27   .domain(['Mon', 'Tue', 'Wed', 'Thu', 'Fri'])
28   .range([0, width])
29   .paddingInner(0.05);
30

```

.paddingInner(entre 0 et 1) : permet d'espacer les rectangles. Plus elle est petite plus ils seront rapprochés. L'échelle est entre 0 et 1

```

32
33 // Dessiner les rectangles
34 monSvg
35   .selectAll('rect')
36   .data(myData)
37   .join(enter => enter.append('rect')
38         .attr('x', d => bandScale(d.day))
39         .attr('y', d => height - d.value )
40         .attr('width',
41               bandScale.bandwidth())
42         .attr('height', d => d.value));
43
44 // Dessiner l'axe X
45 const axisBottom = d3.axisBottom(bandScale)
46 const xAxis = monSvg.append('g')
47   .attr("transform", "translate(0,"+ height + ")")
48   .call(axisBottom)

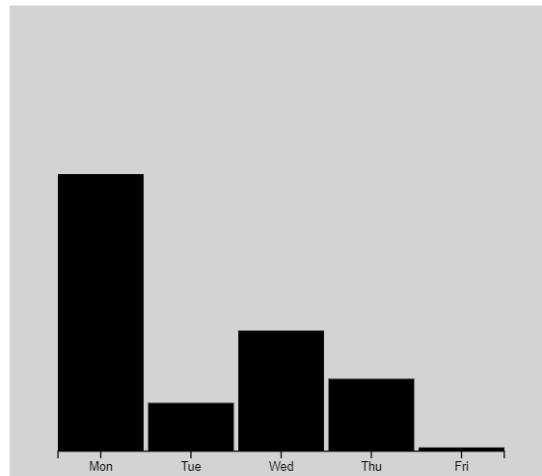
```

RAPPEL : LE SVG A SONT AXE A L'ENVERS

Axe des x :

Translate : si on ne le translate pas, il va tout construire depuis (0,0). Donc ici il faut le translater de 0 en X et de height en Y

Ne pas publier d'appeler l'axe afin de pouvoir le voir dessiner.
La largeur est calculée automatiquement grâce au padding

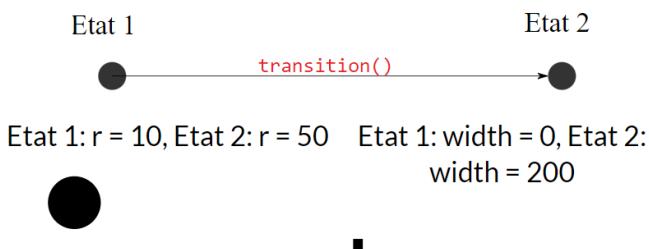


d₃-transition

Installation

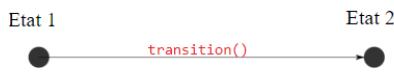
npm install d3 (installe tout le module, mais elle donne aussi toutes les versions séparées, pour que si on n'a pas besoin de tout, le projet soit plus léger)

Transitions



Transitions

```
monSvg.append('element')
    .attr('attribute', Etat_1)
    .transition()
    .attr('attribute', Etat_2)
```



1^{er} attribut : état 1 donc transition = 0

2^{ème} attribut est en état 2 et la transition n'est plus égale à 0

On n'est pas obligé de mettre état 1, c'est surtout l'explication ici

Ce qui est sympa avec D3 c'est qu'on peut appliquer ces transformations qu'aux éléments qu'on appelle.

Transitions

```
transition().ease(fonction_easing)
```

Applique une type de fonction d'accélération à la transition (défaut linéaire). Pour les fonctions Easing: [ici](#)

```
transition().delay((d,i) =>  
    i*tempsEnMillisecondes)
```

Applique un retard à chaque élément sélectionné

1^{ère} fonction : **ease()**

Elle est assez **linéaire**, c'est pour tout ce qui est accélération.

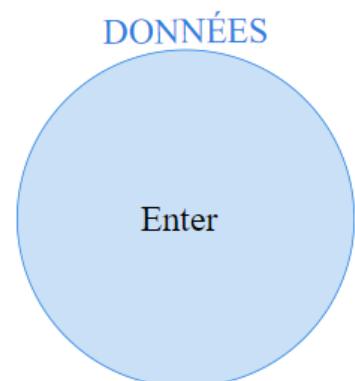
Quadratique : suit une fonction de racine carrée

Fonction **delay()** qui retarde la transition. Elle est souvent utilisé pour 1 donnée après l'autre. Elle est très intéressante quand on veut appliquer la transition à chaque élément.

Joindre les données

RAPPEL

```
selection.data(données)  
.join(enter => enter  
    .append('element')  
    .attr('attribute', d => d.valeur)  
)
```



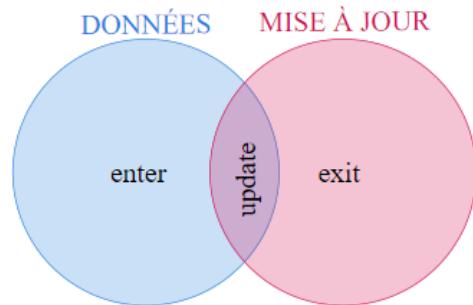
On a une sélection, on append les données puis on les joints avec la méthode **join()**

Joindre les données

Mise à jour des données

enter, update, exit

```
selection.data(données)
  .join(enter => enter
        .append('element')
        .attr('attribute', d => d.valeur),
  update => update
        .append('element')
        .attr('attribute', d => d.valeur),
  exit => exit
        .remove()
      )
.append('element') // Appliqué en tout état (enter, update, exit)
.attr('attribute', d => d.valeur) // Appliqué en tout état (enter, update, exit)
```



Les transitions vont être appliquées aux données.

La méthode `join()` va avoir 3 fonctions :

- `enter` (tout ce qu'on rajoute)
- `update` (si les données sont mises à jour elle a les données d'avant)
- `exit` (tout ce qui va partir, qui ne sera plus là)
-

Joindre les données

EXEMPLE

```
let lettres = tableauLettresAleatoires();

svg.selectAll('text')
  .data(lettres, d => d)
  .join(enter => enter
        .append('text')
        .attr('fill', 'green')
        .text(d => d)
      )
  .attr("x", (d, i) => i * 15) // Mettre à jour
                                à chaque fois
                                la position
```

a b c d e f g h l n o q r

Joindre les données

EXEMPLE

```
let lettres = tableauLettresAleatoires();  
  
svg.selectAll('text')  
  .data(lettres, d => d)  
  .join(enter => enter  
    .append('text')  
    .attr('fill', 'green')  
    .text(d => d),  
    update => update  
      .attr('fill', 'grey')  
    )  
  .attr("x", (d, i) => i * 15) // Mettre à jour  
    à chaque fois  
    la position
```

a c g j k l o x

En gris, tout ce qui est déjà là avant.
Tout ce qui est nouveau est dans enter

Joindre les données

EXEMPLE

```
let lettres = tableauLettresAleatoires();  
  
svg.selectAll('text')  
  .data(lettres, d => d)  
  .join(enter => enter  
    .append('text')  
    .attr('fill', 'green')  
    .attr("x", (d, i) => i * 15)  
    .attr('y', 0)  
    .text(d => d),  
    update => update  
      .attr("fill", "grey")  
      .attr("y", 0)  
      .attr("x", (d, i) => i * 15)  
    exit => exit  
      .attr("fill", "brown")  
      .transition(t)  
      .attr("y", 30))
```

e k l m o u y z
c d f h q r s v

En rouge. Tout ce qui n'est plus là-

d3-zoom

Installation

```
npm install d3-zoom
```

Tout ce qui est interaction de souris

Zoom & Pan

zoom()

```
function handleZoom(e) {
  monSvg.attr('transform', e.transform);
}

let zoomHandler = zoom()
  .on('zoom', handleZoom);

monSvg.call(zoom);
```

- Événements: 'zoom', 'start', 'end'
- Retourne : transform (qu'on peut injecter dans .attr()!)

Pan quand on va translater

Zoom : zoom

handleZoom : créer les translation

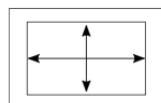
Il faut append la fonction depuis le svg après.

Zoom & Pan

Contraintes

`zoom().constraint(props)`

`scaleExtent([min, max])` `translateExtent([minX,minY],[maxX, maxY])`



Zoom & Pan

Méthodes

`selector.call(zoom.methode, props)`

Zoom

`(zoom.scaleBy, facteur_échelle)`
Multiplie l'échelle actuelle par le facteur d'échelle
`(zoom.scaleTo, échelle)`
Change l'échelle à échelle définie

Pan

`(zoom.translateBy, x, y)`
Translation de x,y
`(zoom.translateTo, x, y)`
Translation jusqu'à x,y

Le **zoom** a des contraintes qui peuvent donner le zoom max et min
translateExtent : donne jusqu'à où on va pouvoir zoomer / bouger
On peut appeler des méthodes au niveau du zoom.

ScaleBy : prend en paramètre un facteur d'échelle.

Pan : si je clique, tu me le translate ici.

Ces fonctions sont pas mal utilisées sur les cartes (zoom pays, translater sur un continent) gère les translations entre des zoom.

Code PEN

<https://codepen.io/romanoe/pen/OJoZwro>

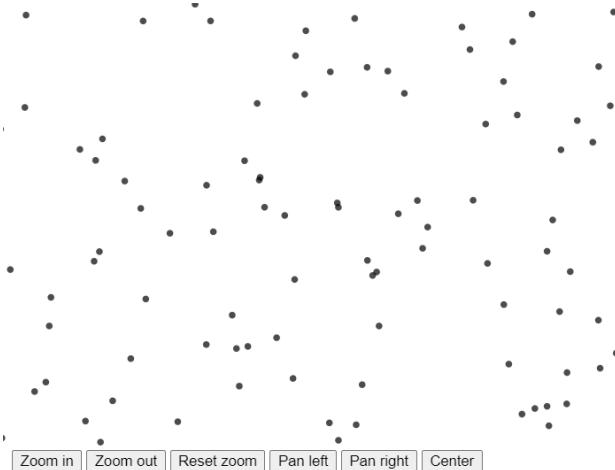
```

22
23 * function handleZoom(e) {
24     d3.select('svg')
25         .attr('transform', e.transform);
26 }
27
28 * function zoomIn() {
29     d3.select('svg g')
30         .transition()
31         .call(zoom.scaleBy, 2);
32 }
33
34 * function zoomOut() {
35     d3.select('svg')
36         .transition()
37         .call(zoom.scaleBy, 0.5);
38 }
39
40 * function resetZoom() {
41     d3.select('svg')
42         .transition()

```

handleZoom : prend un événement en paramètre
On va appliquer un transform en svg

zoomOut : On revient en arrière

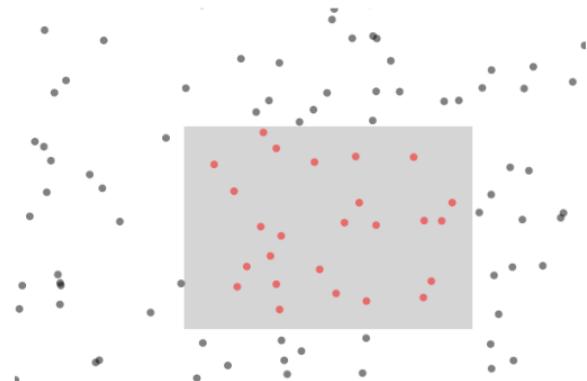


Exemple des différents zooms (explique bien !)

d3-brush

Installation

```
npm install d3-brush
```



Brush permet de faire ce genre d'action sur une donnée (rectangle) -> appliquer une fonction à un svg. Peut être aussi utiliser sur d'autres navigateur.

Brush

brush()

```
let myBrush = brush()  
  .on('brush', handleBrush)  
  
function handleBrush(e) {  
  let etendueBrush = e.selection;  
  // Faire quelque chose avec  
  // la sélection  
}  
  
svg.call(brush);
```

- Événements: 'brush', 'start', 'end'
- Retourne: Étendue du brush ([[x0,y0], [x1,y1]])

Comme le zoom, Brush possède start et end

start : quand on commence à cliquer pour faire le rectangle gris

end : quand on finit le clic du rectangle gris.

Avec l'événement source brush on va lui donner un évènement.

Pour que le brush soit accessible sur le svg il faut appeler la fonction.

CODE PEN

<https://codepen.io/romanoe/pen/BaOVQER>



The screenshot shows the CodePen interface with three panels: HTML, CSS, and JS. The HTML panel contains a simple SVG element. The CSS panel contains a single rule for circles. The JS panel contains the code for initializing a brush, handling its events, and updating the selection. Below the panels is a scatter plot where a rectangular selection is applied to a subset of points.

```
HTML:  
<svg width="600" height="400">  
  <g></g>  
</svg>  
  
CSS:  
.circle {  
  opacity: 0.5;  
}  
  
JS:  
1 let data = [], width = 600, height = 400,  
2   numPoints = 100;  
3  
4 let brush = d3.brush()  
  .on('start brush', handleBrush);  
5  
6 let brushExtent;  
7  
8 function handleBrush(e) {  
9   brushExtent = e.selection;  
10  update();  
11 }  
12  
13 function initBrush() {  
14   d3.selectAll('svg g')
```

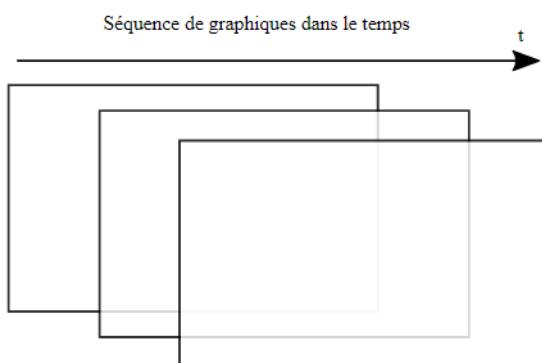
La fonction va colorer en rouge tout ce qu'il y a à l'intérieur du rectangle gris.

`update` : comment on va dessiner ces cercles.

On va appliquer la fonction `fill` qui prend en paramètres une autre fonction qui définit les points.

La `brushExtend` regarde pour chaque point si les x et y sont dans la fenêtre. Cette fonction peut être appellée dans `fill`.

Animation



Animation temporelle correspond à plusieurs animations statiques. On peut les enchaîner et avec des fonctions natives de javascript on va pouvoir afficher un graphique pour chaque donnée temporelle.

Animation

`setInterval(fonction, intervalleEnMs [, params])`

exécute une fonction de manière répétée avec une intervalle fixée. Retourne un identifiant unique (`intervalId`).

`clearInterval(intervalId)`

arrête l'intervalle

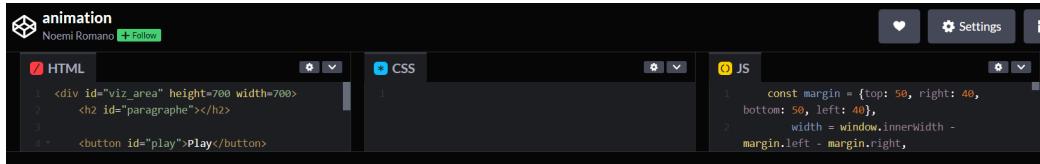
[↳ Doc MDN : setInterval\(\)](#)

Fonctions de JavaScript

L'ID est stockée dans une variable qu'on utilisera ensuite pour arrêter l'intervalle avec la fonction clearInterval

CODE PEN

<https://codepen.io/romanoe/pen/rNZKjvB>



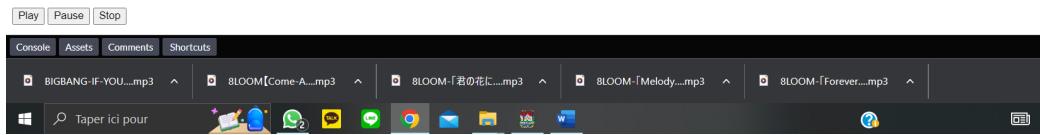
The screenshot shows the CodePen interface with three tabs: HTML, CSS, and JS. The HTML tab contains the following code:

```
<div id="viz_area" height=700 width=700>
  <h2 id="paragraphe"></h2>
<button id="play">Play</button>
```

The CSS tab is empty. The JS tab contains the following code:

```
const margin = {top: 50, right: 40,
bottom: 50, left: 40},
width = window.innerWidth -
margin.left - margin.right,
```

1960



La fonction `animate()` va démarrer une intervalle qui va enclencher une fonction qui sera itérée chaque seconde. (= exécute la même fonction, chaque seconde)

Fonction `updateShare()` : permet de dessiner le rond

Si on met stop, cela recommence depuis le début.

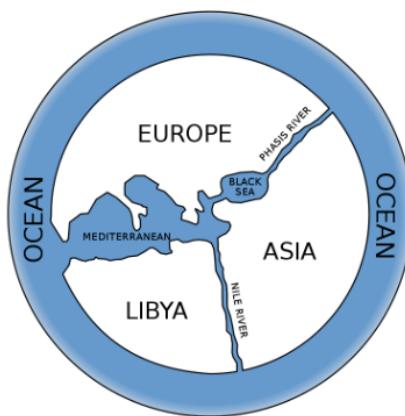
Si on met pause, la valeur est gardée et recommence au même endroit.

QUAND ON A DES DONNES TEMPORELLES, REGARDER CET CODE PEN



Carte gravée sur la défense d'un mammouth, 25.000 av. J.-C., trouvé à Pavlov, CZ (Source: Wikipedia)

Comme quoi, la cartographie ne date pas d'hier.



Reconstruction of Anaximander's Map (Source: Digital Maps of the Ancient World)

Juste avant JC

On commence à avoir des cartes qui mettent en valeur le contour des pays, le territoire



Ptolemy map of the World, II siècle après J. C. (Source: Wikipedia)

On commence à voir le système de données. Pour les faire, on se basait sur l'astrologie (position des étoiles)

On commence à visualiser le territoire à travers l'espace.

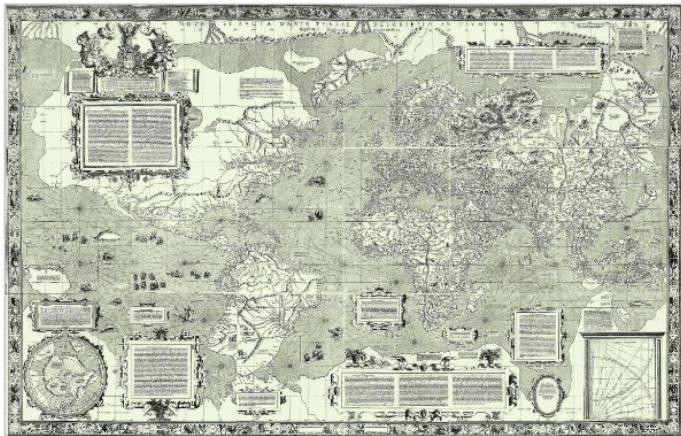


Tabula Rogeriana, Muhammad al-Idrisi, 1153 (Source: Wikipedia)

Avec l'empire romain et Ottoman, il y a les premières cartes du monde qui arrivent.

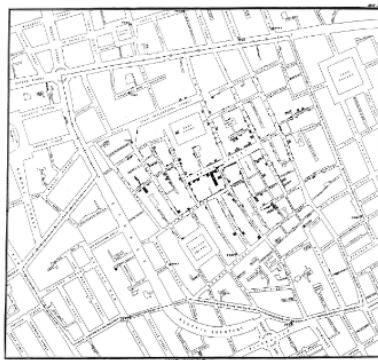
On commence à avoir des cartes 2D qui ressemble de plus en plus à celle que l'on a maintenant. Sur ces cartes tout est encore aplatie.

Avec ces cartes, plus on s'éloigne du centre, plus cela sera déformé. Elle est d'ailleurs à l'envers (car sinon on ne comprend pas ce c'est une carte). A l'époque on s'orientait pas nous-même. Ici la notion nord-sud n'était pas encore très claire.



Mercator Map (Source: Wikipedia)

C'est la projection de Mercator qui concerne les ancre, pour la navigation bateau. Ce sont des formules mathématiques qui permet de garder les endroits. C'est une carte connue.



L'épidémie choléra de Broad Street, John Snow, 1854

Carte de la ville créée pour zoner le choléra



How the virus got out, New York Times

Carte plus interactive grâce à l'arrivée du web.

Processus

But Pourquoi ? Quelle est l'histoire que je veux raconter ?

Public Pour qui ? Des experts ? Des novices ?

Type Quel type de carte pour quelles données ?

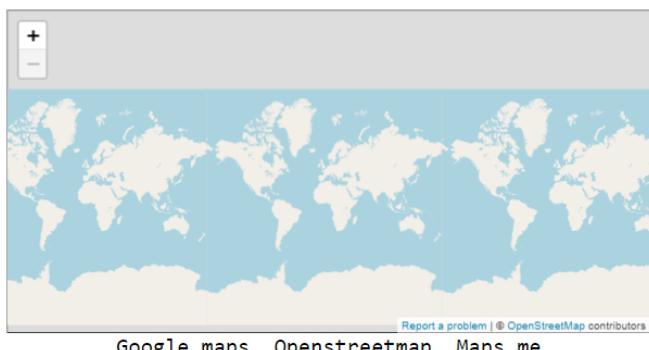
Design Lisibilité, interprétabilité, visuel

Le **but** est une question à poser dans tout élément visuel.

Le **type** de carte dépend de la données que l'on a à disposition et du message que l'on veut passer.

Type de carte

Cartes de référence



Google maps, Openstreetmap, Maps.me

Carte que l'on voit tous les jours (google map) Ici on n'a pas vraiment de données à interrogées, c'est une sorte de carte sur tuilage (comme les tuiles d'un toit)
Il y a donc plusieurs cartes à différentes échelles qui sont divisées en plusieurs part

Cartes thématiques

Qualitative

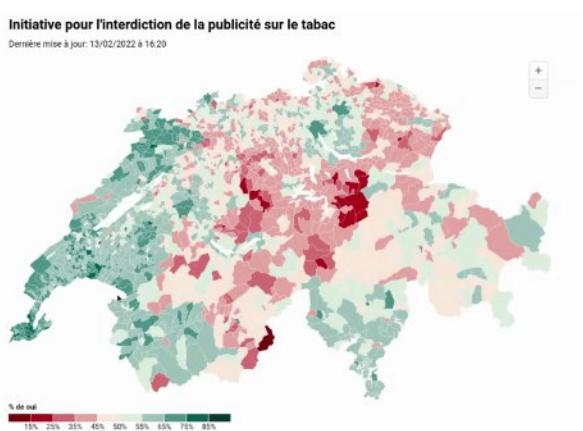


Les cantons (Source: Bonjour Genève)

On va donner une couleur aux classes.

Cartes thématiques

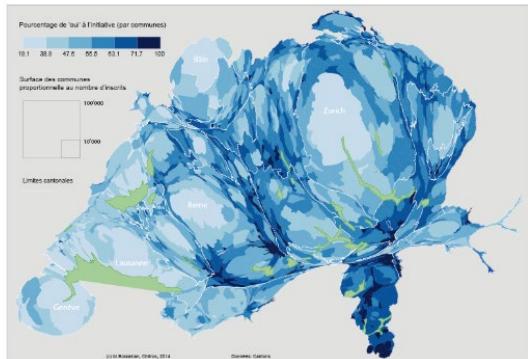
Choroplète



Votations fédérales 2022 (Source: RTS)

Données plutôt quantitative. On voit l'échelle continue des données. Par exemple cela peut être pour les cartes de votations.

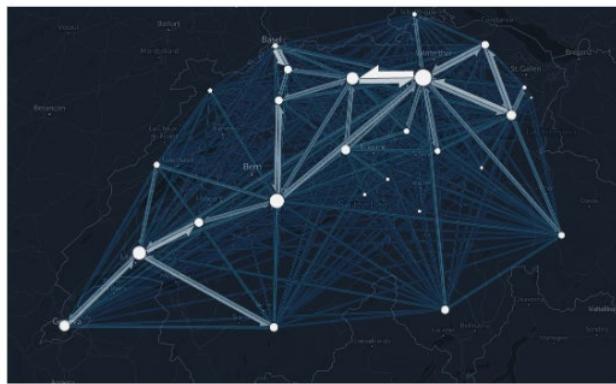
Cartogrammes



Vraies et fausses évidences de la géographie électorale suisse, Manouk Borzakian

On veut faire passer un message avec ce type de cartes. On voit le nombre de lecteur et combien de pourcentage par région. Les grandes villes sont plus grandes, cela donne la proportions des lecteurs et cela met en évidence des villes qui ont en général des avis assez similaires. Cela montre le clivage (c'est qqch souvent oublié alors qu'elle est plus précise que la précédente qui nous ferait dire par erreur suisse romand VS suisse allemand) On n'en voit pas trop, car elles sont difficiles à faire. C'est une manière de visualisé des données quantitatives géographique sur 3 axes.

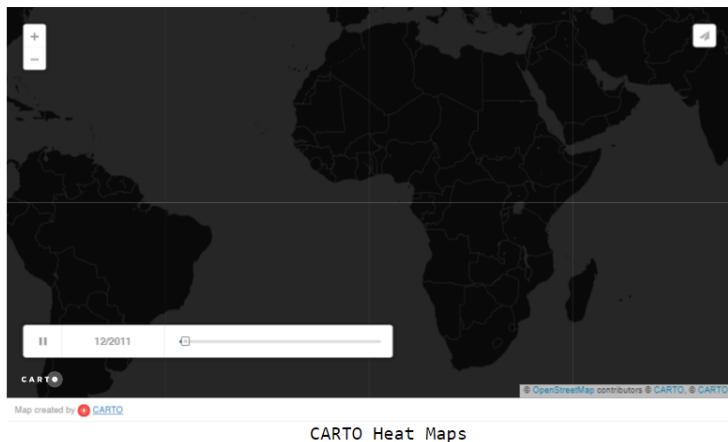
Flowmap



Migration en Suisse 2016, [flowmap.blue](#)

Permet de visualiser des flux entre différent endroit (exemple : transport)

Cartes de chaleur



Selon un nombre de point à un certains endroit, on va créer une carte d'une certaine densités sur le graphique. C'est beaucoup plus direct, strict to the point.

Cartes topographiques



Images relief, [Reddit](#)

Ce genre de données en Suisse sont en libre accès depuis 2 ans. C'est une énorme image de la suisse. On peut le créer dynamiquement mais le vrai effet 3D est fait en design.

Suisse topo a pleins de données graphique sur notre territoire sur et au-dessus).
Données gratuites

Cartes topologiques



Représente souvent les cartes des transports publics. Ces cartes ne représentent pas vraiment les distances entre les points.

Design

Projection

Comment on passe du 3d au 2d

Symbologie

Variables visuelles: symboles, couleurs

Echelle

Dépendante des détails qu'on souhaite montrer

Texte

Légendes, labels

C'est important de mettre un contexte et également de toujours mettre un titre et des labels pour faire référence à différents détails.

Formats de données

- **geojson** Comme du .json mais avec des données géographiques
- **OSM (xml)** OpenStreetMap
- **shp** Shapefile
- **WKT** Well Known Text

Ce qu'on va utiliser le plus en cours c'est le geojson

Sources de données

- Mondiales: [Natural Earth](#)
- Suisse: [opendata.swiss](#)
- OpenStreetMap (OSM)

OpenStreetMap (OSM)



<https://www.openstreetmap.org/#map=15/46.7841/6.6392>

Ils sont un peu le concurrent de Google map.
Suisse topo

OSM

vs

Google Map

"Google Map is a closed system, and every information is property of Google. OpenStreetMap is an open data source, and its information is available to every organisation and user."

Récupérer les données

- Par région :
`curl "https://api.openstreetmap.org/api/0.6/map?bbox=6.645,46.779,6.65,46.783" > heig.osm`
- Par types d'objets : Overpass API
Objets: https://wiki.openstreetmap.org/wiki/Map_features
- Géocodage: Nominatim
<https://nominatim.openstreetmap.org/search?city=yverdon&format=json>

Overpass API

Exemple

- Cherchez la ville, village, région qui vous intéresse
- Cliquez sur le bouton **Wizard**
- Cherchez un mot clef comme *pub*, *bar*, *tree* etc.
- Cliquez sur **Export** et choisissez le format que vous souhaitez

<https://overpass-turbo.osm.ch/>

Librairies javascript

- d3
- Leaflet
- OpenLayers

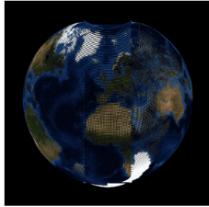
... ET BEAUCOUP D'AUTRES!

d3-geo

Installation

`npm install d3-geo`

Projections



```
let projection = nom_projection()
```

d3-geo-projection

La façon dont on va aplatis la carte. On obtient des coordonnées projetées sur une sphère puis on va appliquer des projections avec des angles données pour passer en **2D**. Pour passer en projection il suffit d'appeler le nom qu'on veut.

On a besoin de mapper une donnée en pixel. Il faut penser au projection comme une échelle.

Projections

Transformations

```
projection.fonction(arguments)
```

Manuelles

```
.scale(facteur_échelle)
```

Changer l'échelle de la carte

```
.translate(X, Y)
```

Appliquer une translation de X et Y

Selon les données

```
.fitSize([width, height], geojson)
```

Adapter selon largeur/hauteur du SVG

```
.fitExtent([x0, y0, x1, y1], geojson)
```

Adapter selon une certaine étendue

On peut faire des changements sur cette projection. (la façon automatique adapte aux données qu'on lui donne, ce qui est bien pratique ? (ex. selon la taille de notre svg))

Geopath

Syntaxe

```
.geojson → <path></path>
```

```
// Définition projection (p. ex. geomerclator, geoOrthographic etc.)  
let projection = geomerclator().fitSize([width, height], geojson);  
  
// Générateur de <path></path>  
const pathGenerator = geoPath()  
  .projection(projection)  
  
// <path></path> selon les données  
groupe.selectAll("path")  
  .data(geojson.features)  
  .join(enter => enter.append("path")  
    .attr("d", pathGenerator))
```

Code Pen

<https://codepen.io/romanoe/pen/mdGZqBx>

La notion de path : la fonction lit les coordonnées et dessine automatiquement des lignes. Geopath est un générateur de ligne qui prend comme argument une

projection. Car il a besoin de mapper une coordonnées à qqch qui se trouve dans les pixels.

Leaflet

Installation

```
npm install leaflet
```

Librairie la plus utiliser pour faire de la visualisation de données sur D3.

Leaflet

Syntaxe

1. Création division

```
// index.html
// 1. Définition du conteneur
<div id="mapid" style="width: 600px; height: 400px;"></div>
```

2. Générer un élément *map*

```
// index.js
// 2. Rajouter la carte au conteneur
```

3. Rajouter un **fond de plans**

```
let map = L.map('mapid').setView([cx, cy], zoom);
```

4. Ajouter des **éléments** à la carte

```
// 3. Fond de carte
L.tileLayer(fondDeCarte).addTo(map);
```

5. Écouter des **événements**

```
// 4. Rajouter des éléments
L.Marker([cx,cy]).addTo(map);
L.geoJSON(objetGeojson).addTo(map);
L.bindPopup([cx, cy], "<h1>My popup</h1>").openPopup();
```

[↳ Tutoriels](#)

```
// 5. Event
map.on('click', fonction);
```

SCROLLYTELLING

Permet la simplification de la visualisation de storytelling en horizontal.



- Creative coding
- HTML Canvas
- Liens entre DOM et Canvas pas direct

Creative coding : code qui a une veine artistique. Très utiliser pour des expositions. C'est souvent les media interactive designer qui utilise ce genre de chose. On voit vraiment du dessins et on ne rajoute pas des éléments svg dans l'utilisateurs.

On peut traduire des lois mathématiques assez complexe qui peuvent générer du contenu modelable et ou on dirait vraiment du dessin via JS.

Souvent utilisé pour des effets de particules, abstraites.

Librairie qui dessine directement les canvas (pas comme D3) Ca va le dessiner. Ça sera plus efficace mais il y aura moins d'interactivité car cela ne sera plus des éléments du dessins.



- 3D
- WebGL
- Haute performance

Permet de créer des effets 3D. Basé sur webGL. Il utilise la carte graphique de notre pc, de notre performance. Très utilisé dans les jeux vidéo 3D. Créer l'espace 3D à partir des données.

<https://2050.earth/>



Chart.js

- Simple et rapide
- HTML Canvas
- Visualisation simples
- Peu de flexibilité (6 graphiques)

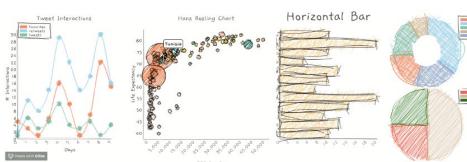
Peut créer du canvas et aussi du vectoriel. Beaucoup plus simple à utiliser que D3. Permet de faire des graphiques assez simples, interactifs de manière très très très rapide. Se base sur un fichier d'intégrations. Si on veut faire des graphiques simples, il peut suffire.



- JSON-declarative
- Visualisation simples
- Bonne documentation
- Difficilement personnalisable

Produit des graphiques similaires à D3, la différence est qu'on va configurer par un JSON ce que l'on va faire, créer une visualisation (pas comme D3 qui est basé sur JS) Permet de créer des visualisations assez simples et pas trop modifiables.

roughviz.js



- Basée sur rough.js, d3.js et handy
- Visualisation simples
- Visuelle mais rend résultats approximatifs
- Difficilement personnalisable

Librairie qui se base sur D3. Cette librairie permet de faire un style de dessin sur les différents éléments SVG. Effet sympathique, mais attention au type de public que l'on a. Cela dépend de l'ambiance que l'on veut mettre, de ce que l'on veut montrer. C'est très visuel mais cela rend les résultats assez approximatifs. Difficilement personnalisable si on ne connaît pas D3.

Cela peut vraiment être pris comme un thème spécial.

Mais alors... pourquoi d3.js?

- Flexibilité
- Manipulation du DOM facile
- Maîtrise de **d3.js**, facile d'apprendre les autres outils!

Flexible, permet de manipuler facilement le DOM. Si on maîtrise D3, cela sera beaucoup plus simple d'apprendre les autres types de visualisations.

D3 pour échelles, fonction mathématiques.

Mais alors... pourquoi **pas** d3.js?

- Avec beaucoup de données, peu performant
- Documentation difficile à trouver
- Longue courbe d'apprentissage
- Simples visualisations: overkill

De manière général, il y a peu de personnes qui ont beaucoup de données. Si on a beaucoup de données, ce n'est pas D3 que l'on devrait prendre.

Il n'y a pas beaucoup de documentation, et quand on en trouve, elle est souvent obsolète. (sauf sur git)

D3 n'est qu'une petite partie d'une application et pas une application entière (préférer SVELT, il est plus rapide, génère du double dynamique très efficace et la syntaxe est très simple à utiliser. On écrit directement des itérations et des git)

Scrollytelling

Wiki death

covid-19

Mapbox

Outil de base pour faire du storytelling dans un contexte assez simple.

scrollama

Installation

```
npm install scrollama intersection-observer
```

↳ Doc

↳ Exemples

Librairie très utilisées par les journaux. Permet de détecter où l'on se trouve sur notre page web. Détermine à quelle étape on est dans le scroll.

Sticky scroller

Codepen

<https://codepen.io/romanoe/pen/qBJjVaR>

The screenshot shows the CodePen interface with three panels: HTML, CSS, and JS. The HTML panel contains a section with four steps. The CSS panel contains styles for the scrollama library. The JS panel contains the scrollama initialization code. Below the panels is a preview area showing a yellow header and a grey body with a step labeled 'STEP 2' and a large number '1' indicating the current scroll position.

```
HTML
<section id="scrollly">
  <article>
    <div class="step" data-step="1">
      <p>STEP 1</p>
    </div>
    <div class="step" data-step="2">
      <p>STEP 2</p>
    </div>
    <div class="step" data-step="3">
      <p>STEP 3</p>
    </div>
    <div class="step" data-step="4">
      <p>STEP 4</p>
    </div>
  </article>
```

```
CSS
#scrollly {
  position: relative;
  display: -webkit-box;
  display: -ms-flexbox;
  display: flex;
  background-color: #f3f3f3;
  padding: 1rem;
}

#scrollly>* {
  -webkit-box-flex: 1;
  -ms-flex: 1;
  flex: 1;
```

```
JS
const scrollly = d3.select("#scrollly");
const figure = scrollly.select("figure");
const article = scrollly.select("article");
const step = article.selectAll(".step");

// initialize the scrollama
const scroller = scrollama();

// generic window resize listener event
function handleResize() {
  // 1. update height of step elements
  const stepH = Math.floor(window.innerHeight * 0.75);
  step.style("height", stepH + "px");
}
```

Le mécanisme est pareil pour tous les types de scrolling. Ce qui change de ce qui passe de l'un à l'autre est juste du CSS .

scrollama

Noemi Romano [+Follow](#)

HTML

```

15 </article>
16
17 <!--C'est là que vous allez mettre votre svg qui
évolue-->
18 <figure>
19 <p></p>
20 </figure>
21
22
23 </section>
24

```

CSS

```

15
16 article {
17   position: relative;
18   padding: 0 1rem;
19   max-width: 20rem;
20 }
21
22 figure {
23   position: -webkit-sticky;
24   position: sticky;
25   width: 100%;
26   margin: 0;
27   -webkit-transform: translate3d(0, 0, 0);
28   -moz-transform: translate3d(0, 0, 0);
29   transform: translate3d(0, 0, 0);

```

JS

```

14
15 const figureHeight = window.innerHeight / 2;
16 const figureMarginTop = (window.innerHeight -
figureHeight) / 2;
17
18 figure.style("height", figureHeight + "px")
.style("top", figureMarginTop + "px");
19
20 // 3. tell scrollama to update new element dimensions
21 scroller.resize();
22
23 }
24
25
26 // scrollama event handlers
27 function handleStepEnter(response) {

```

STEP 2

1

scrollama

Noemi Romano [+Follow](#)

HTML

```

15 </article>
16
17 <!--C'est là que vous allez mettre votre svg qui
évolue-->
18 <figure>
19 <p></p>
20 </figure>
21
22
23 </section>
24

```

CSS

```

29
30 transform: translate3d(0, 0, 0);
31 background-color: #8a8a8a;
32 z-index: 0;
33
34 figure p {
35   text-align: center;
36   padding: 1rem;
37   position: absolute;
38   top: 0%;
39   left: 50%;
40   -moz-transform: translate(-50%, -50%);
41   -webkit-transform: translate(-50%, -50%);
42   transform: translate(-50%, -50%);
43   font-size: 8rem;

```

JS

```

27 function handleStepEnter(response) {
28   console.log(response);
29   // response = { element, direction, index }
30
31   // add color to current step only
32   step.classList("is-active", function (d, i) {
33     return i === response.index;
34   });
35
36   // update graphic based on step
37   figure.select("p").text(response.index + 1);
38
39   function init() {
40
41

```

STEP 2

1

scrollama

Noemi Romano [+Follow](#)

HTML

```

15 </article>
16
17 <!--C'est là que vous allez mettre votre svg qui
évolue-->
18 <figure>
19 <p></p>
20 </figure>
21
22
23 </section>
24

```

CSS

```

44 font-weight: 900;
45 color: #fff;
46
47
48 .step {
49   margin: 0 auto 2rem auto;
50   background-color: #b3b3b3;
51   color: #fff;
52 }
53
54 .step:last-child {
55   margin-bottom: 0;
56 }
57
58 &::not(.is-active) {

```

JS

```

42 // 1. force a resize on load to ensure proper
dimensions are sent to scrollama
43 handleResize();
44
45 // 2. setup the scroller passing options
46 // this will also initialize trigger
47 observations
48 // 3. bind scrollama event handlers (this can be
chained like below)
49 scroller
50 .setup({
51   step: "#scrolly article .step",
52   offset: 0.33,
53   debug: false
54 })

```

STEP 2

1

scrollama
Noemi Romano [Follow](#)

HTML

```
15      </article>
16
17 + <!--C'est là que vous allez mettre votre svg qui
18 +     évolue-->
18 +     <figure>
19 +       <p>@</p>
20     </figure>
21
22   </section>
23
24
```

CSS

```
58 +
59   .step.is-active {
60     background-color: goldenrod;
61     color: #3b3b3b;
62   }
63
64   .step p {
65     text-align: center;
66     padding: 1rem;
67     font-size: 1.5rem;
68 }
```

JS

```
53
54   .onStepEnter(handleStepEnter);
55
56
57 // kick things off
58 init();
```

STEP 2

1