

Résumé du cours - VisualDon

Cours n°1 : SVG

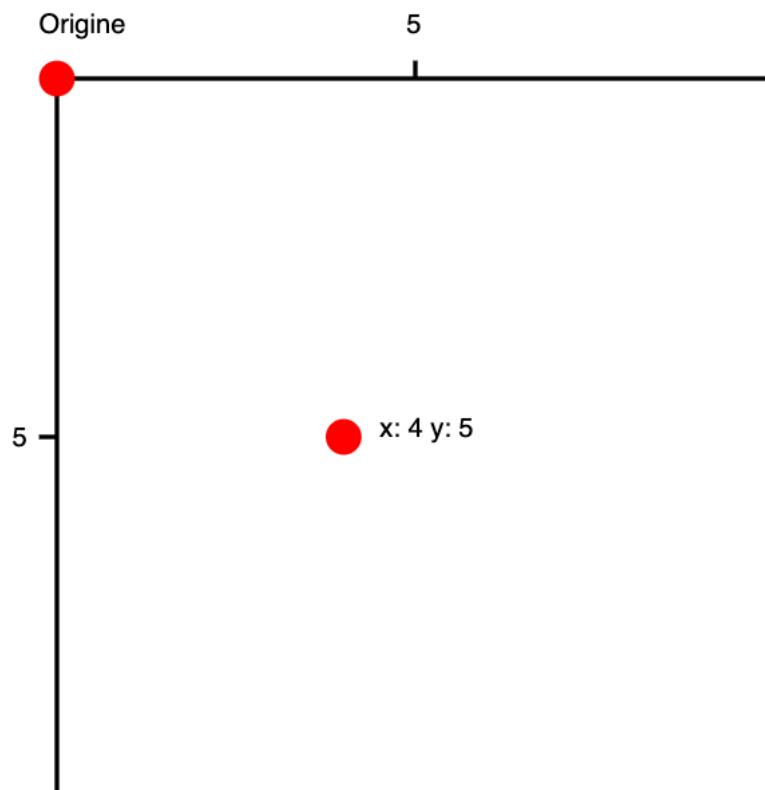
SVG = Scalable Vector Graphics = Graphique Vectoriel Adaptable

S'inscrit dans le HTML pour être affiché sur une page web (dans notre cas)

Système de coordonnées

SVG fonctionne avec un système de coordonnées.

- L'origine se trouve en haut à gauche de l'écran
- X représente la largeur (depuis l'origine)
- Y représente la hauteur (depuis l'origine)



Définir la taille du SVG

Pour définir l'espace que va occuper notre SVG on doit préciser les attributs width et height dans la balise <svg> (nous donne la taille du contenant du SVG) :

```
<svg width="400" height="110"></svg>
```

Créer des formes

Carré et rectangle :

```
<svg width="400" height="110">
  <rect x="0" y="0" width="300" height="100" fill="steelblue" />
</svg>
```

- x et y définissent l'emplacement du coin en haut à gauche
- width et height représentent les dimensions du rectangle
- fill définit la couleur de remplissage

Cercle :

```
<svg height="100" width="100">
  <circle cx="150" cy="50" r="40" stroke="black" stroke-width="3" fill="red" />
</svg>
```

- cx et cy définissent l'emplacement du centre du cercle
- r représente le rayon
- stroke définit la couleur de la bordure
- stroke-width définit la largeur de la bordure
- fill définit la couleur de remplissage

Ligne :

```
<svg height="210" width="500">
  <line x1="0" y1="0" x2="200" y2="200" stroke="red" />
</svg>
```

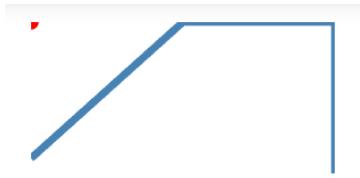
- x1 et y1 donnent l'emplacement du début du trait
- x2 et y2 donnent l'emplacement de la fin du trait
- stroke définit la couleur de la ligne

Texte :

```
<svg height="210" width="500">
  <text x="500" y="600">Le texte à écrire...</text>
</svg>
```

Path :

```
<svg width="200" height="100">
  <path d="M 0 90 L 100 0 H 200 V 100" fill="none" stroke="steelblue" stroke-width="5"></path>
</svg>
```



- M = move to → "M x y" va déplacer le "stylo" au dessus du point (0,90)
- L = line to → "L x y" va dessiner un trait jusqu'au point (100, 0), depuis le point défini avec M (0,90)
- H = horizontal line → "H x" va dessiner un trait horizontal jusqu'à x=200 (depuis le point précédent)
- V = vertical line → "V y" va dessiner un trait vertical jusqu'à y=100 (depuis le point précédent)

Courbe (avec path) :

```
<svg width="200" height="100">
  <path d="M 20 20 L 180 20" fill="none" stroke="steelblue"></path>
  <path d="M 20 20 C 40 50 160 50 180 20" fill="none" stroke="indianred"></path>
  <circle cx="40" cy="50" r="2" fill="red"></circle>
  <circle cx="160" cy="50" r="2" fill="red"></circle>
</svg>
```



- M → move to (comme avant)
- C → courbe (?) → "C cx1 cy1 cx2 cy2 x y"
 - cx1 et cy1 = intensité de la première courbure (?)
 - cx2 et cy2 = intensité de la 2ème courbure (?)
 - x et y = emplacement de l'arrivée

Groupes :

```
<svg width="200" height="100">
  <g fill="red" stroke="steelblue" stroke-width="2" transform="translate(-50, 20)">
    <circle cx="50" cy="50" r="30"></circle>
    <rect x="40" y="40" width="60" height="30"></rect>
    <text x="100" y="80" font-size="50">VisualDon</text>
  </g>
</svg>
```



→ Le groupe permet de donner les mêmes attributs à plusieurs formes

Transformations

Dans la déclaration d'un groupe ou d'un SVG simple, ajouter un attribut "transform", qui prend comme valeur :

- translation → "translate(x, y)"
- rotation → "rotate(angle, cx, cy)"
- scale(facteur_échelle)

```
<svg width="500" height="150">
  <g fill="indianred" stroke="steelblue" stroke-width="2" transform="rotate(20, 100, 50)">
    <circle cx="150" cy="50" r="30"></circle>
    <rect x="40" y="40" width="60" height="30"></rect>
    <text x="100" y="80" font-size="50">VisualDon</text>
  </g>
</svg>
```

Cours n°2 : D3

Syntaxe

- 1 Chaînage de méthodes sur un objet

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

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

Méthode de sélection → select ou selectAll

- Sélectionne un élément du HTML
- Prend en paramètre un sélecteur CSS
- (Lorsqu'il n'y a pas d'élément au dessus de celui à sélectionner, ou qu'on les veut tous (peu importe leur parent), on sélectionne avec l'objet d3, sinon avec l'élément parent qui aura été préalablement sélectionné et mis dans une variable)

	CSS	Exemple
type	element	d3.select('h1')
class	.class	d3.select('.class')
identifiant	#id	d3.select('#id')

```
const svg = d3.select(".mon-svg");  
const cercle = svg.select("circle");
```

Méthode pour définir les attributs → attr

- S'applique sur l'élément sélectionné
- Ajoute un attribut avec sa valeur
- Prend en paramètre le nom de la propriété puis sa valeur : **.attr("propriete","valeur")**

```
const cercle = svg.select("circle");  
cercle.attr("fill", "#E92528");  
cercle.attr("r", "100");
```

Méthode pour ajouter un élément HTML → append

- S'applique sur l'élément sélectionné
- Prend en paramètre le nom de l'élément à ajouter : **.append("element-html")**

```
const div = d3.select(".mon-svg")  
    .append("svg");
```

Méthode pour écouter un évènement → on

- S'applique sur l'élément sélectionné
- Prend en paramètre le nom de l'évènement et une fonction de callback : **.on("event", fonction)**

```
let r = 15  
const svg = d3.select(".mon-svg");  
const cercle = svg.select("circle")  
    .attr("fill", "#E92528")  
    .attr("r", r);  
cercle.on("click", () => {
```

```

        r = r + 10;
        cercle.attr("r", r);
    });
//agrandir la taille du cercle quand on clique dessus

```

Méthode pour joindre les données d'un tableau à un élément HTML → data

- S'applique sur l'élément sélectionné (ou plusieurs)
- Prend un paramètre un tableau
- Méthode qui compte et analyse les données d'un tableau
- Permet de lier un tableau à un élément HTML sélectionné
- Ne sert pas seule, il faut ajouter une ou plusieurs méthodes → .enter() / .text() / etc.

```

<p>D3 Tutorials</p>

<script>
    var myData = ["Hello World!"];

    var p = d3.select("body")
        .selectAll("p")
        .data(myData)
        .text(function (d) {
            return d;
        });
</script>
//le texte "D3 Tutorials" est remplacé par "Hello World!"

```

Méthode pour parcourir un tableau → enter

- S'applique sur l'élément sélectionné
- Ne prend rien en paramètre : `selection.enter();`
- Utile lorsqu'on veut créer un nouvel élément HTML pour chaque valeur d'un tableau
- Entre dans chaque valeur du tableau l'une après l'autre et va appliquer les méthodes qui suivent à chacunes d'elles

```

const data = [4, 6, 2, 8, 1]

const myDiv = d3.select(".my-div");
const ul = myDiv.append("ul"); //création d'un élément de type liste à puces

ul.selectAll("li") //sélection de chaque élément qui se trouve sous ul (fictif?)
    .data(data) //lier le tableau à ul
    .enter() //entrer dans le tableau pour le parcourir
    .append("li") //ajouter un élément li pour chaque valeur du tableau
    .text(d => d); //l'élément courrant li ajoutera le texte qui correspond à la valeur courante du tableau

```

- 4
- 6
- 2
- 8
- 1

Cours n°3 : Données

Formats de données

CSV :

- Comma-Separated Values
- Exportable depuis Excel
- NomFichier.csv

```

prenom, annee_naissance
Alphonse, 1932
Béatrice, 1964
Charlotte, 1988

```

JSON :

- Javascript Object Notation
- Souvent on télécharge un fichier JSON depuis le web ou on fetch une URL
- NomFichier.json

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

TSV :

- Tab-Separated Values
- NomFichier.tsv

prenom	annee_naissance
Alphonse	1932
Béatrice	1964
Charlotte	1988

XML :

- Extensible Markup Language
- NomFichier.xml

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

Charger les données

CSV :

```
d3.csv('chemin/du/fichier.csv')  
    .then(function(data) {  
  
        //Manipuler les données ici  
        console.log(data);  
    })
```

```

        })
        .catch(function(error){
            // Gérer l'erreur ici
            console.log(error);
        });
    });
}

```

JSON :

```

d3.json('chemin/du/fichier.json')
    .then(function(data) {
        //Manipuler les données ici
        console.log(data);

    })
    .catch(function(error){
        // Gérer l'erreur ici
        console.log(error);
    })
}

```

```

d3.json('URL')
    .then( function(data) {
        //Manipuler les données ici
        console.log(data);

    })
    .catch(function(error){
        // Gérer l'erreur ici
        console.log(error);
    })
}

```

Manipuler les données

- Les données récupérées (data) sont présentées sous leur format (json, csv, etc)
- Déconstruire ce format et mettre les données sous forme de tableau

Map :

- `array.map(function(currentValue, index, arr), thisValue)`
- Parcours un tableau (ou autre format de données) pour en faire une copie avec un traitement sur les valeurs

```

d3.csv('chemin/du/fichier.csv')
    .then(function(data) {
        const prenoms = data.map((d, i) => {
            return d.prenom;
        });
    })
    .catch(function(error){
        console.log(error);
    })
// Création d'un nouveau tableau "prenoms", qui contient les valeurs du "champ" prenom du doc CSV, soit :
// ["Alphonse","Béatrice","Charlotte"]

```

Filter :

- `array.filter(condition)`
- Parcours les données et permet de créer un nouveau tableau qui contient des valeurs triées selon des conditions données

```

d3.csv('chemin/du/fichier.csv')
    .then(function(data) {
        const now = 2022;
        const retraite = data.filter((d, i) => {
            retraite = now - d.anne_naissance;
            return retraite < 65 ;
        });
    })
    .catch(function(error){
        console.log(error);
    })
// Création d'un nouveau tableau "retraite" qui contient seulement les objets où l'âge est en dessous de 65
// ["Charlotte", 1988]

```

Reduce :

- `array.reduce(function(total, currentValue, currentIndex, arr), initialValue)`
- Réduit un tableau à une seule valeur en réduisant deux éléments à la fois (de gauche à droite) par une fonction donnée.

```
d3.csv('chemin/du/fichier.csv')
  .then(function(data) {
    const start = 0;
    const somme_annees = data.reduce((r, d) => {
      r + d.annee_naissance, start //c quoi r ?
    });
  })
  .catch(function(error){
    console.log(error);
  })

// nouvelle valeur "somme_annees" qui correspond à l'addition de chaque année
// 5884
```

```
d3.json('chemin/du/fichier.json')
  .then( function(data) {
    const notesParEleve = data.reduce((r, d) => {
      const notes = r[d.nom] || []
      return { ...r, [d.nom]: [...notes, d.note] }
    }, {})
  })
  .catch(function(error){
    console.log(error);
  })

// { Alice: [5, 6], Bob: [4, 3] }
```

Cours n°4 : Échelles et axes

Scale functions

Scale functions are JavaScript functions that: take an input (usually a number, date or category) and return a value (such as a coordinate, a colour, a length or a radius). They're typically used to transform (or 'map') data values into visual variables (such as position, length and colour).

<https://www.d3indepth.com/scales/>

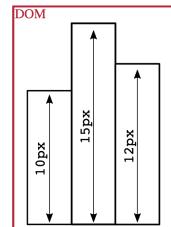


D 3

Échelles

- Données = Echelle * Pixels (avant : données = pixels)
- On ajoute une échelle aux données pour réduire ou agrandir leur visualisation de manière proportionnelle

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



Données continues

- Séquence continue de données
- Points connectés, valeurs qui se suivent
- Exemple : la variation des prix d'un produit, âge d'une personne, température, temps, argent

CONTINU -> CONTINU

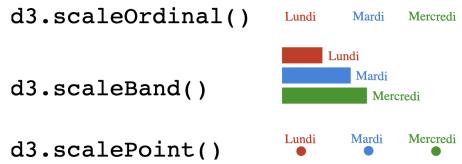


Données discrètes

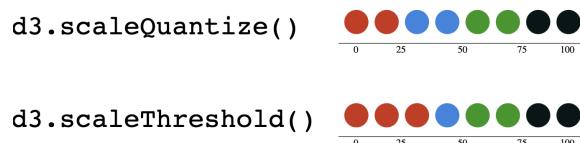
- Comptables
- Que des nombres entiers (ex: nombre d'enfants dans une famille, âges des personnes,...)

- Points isolés, espaces libres entre les valeurs
- Exemple : les jours de la semaine, nombre d'enfant dans une classe

DISCRET -> DISCRET



CONTINU -> DISCRET



Domaine

- `.domain([v1, v2])`
- Unité = l'unité de la donnée
- Entre quel et quel chiffre de notre tableau (ou autre ensemble de données) on veut afficher → portée des données

Gamme (plage)

- `.range([v1, v2])`
- Unité = pixels
- Espace pris sur la page → souvent on met height et width pour que ça prenne tout l'écran

```
var x = d3.time.scale().range([0, width]);
var y = d3.scale.linear().range([height, 0]);
```

Axes

Axe x :

```
d3.axisLeft(ecelle)
```



```
d3.axisRight(ecelle)
```

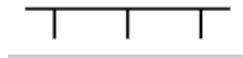


Axe y :

```
d3.axisTop(echelle)
```



```
d3.axisBottom(echelle)
```



Création :

```
const axe = d3.axisBottom(echelle)
```

Dessin :

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

Exemple :

```
const x = d3.scaleLinear()
    .domain([0, 100])
    .range([0, 400]);

svg.call(d3.axisBottom(x));

svg.append("circle")
    .attr("cx", x(10))
    .attr("cy", x(100))
    .attr("r", 40)
    .style("fill", "blue");
```

1 10 20 30 40 50 60 70 80 90 100



Marges

```
const margin = {top : 10, right: 40, bottom: 10, left: 40},
width = 450 - margin.left - margin.right,
height = 400 - margin.top - margin.bottom;

//définir le svg :
svg.attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
    .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");
```

```

//créer l'axe x :
const x = d3.scaleLinear()
    .domain([0,100])
    .range([0,width])

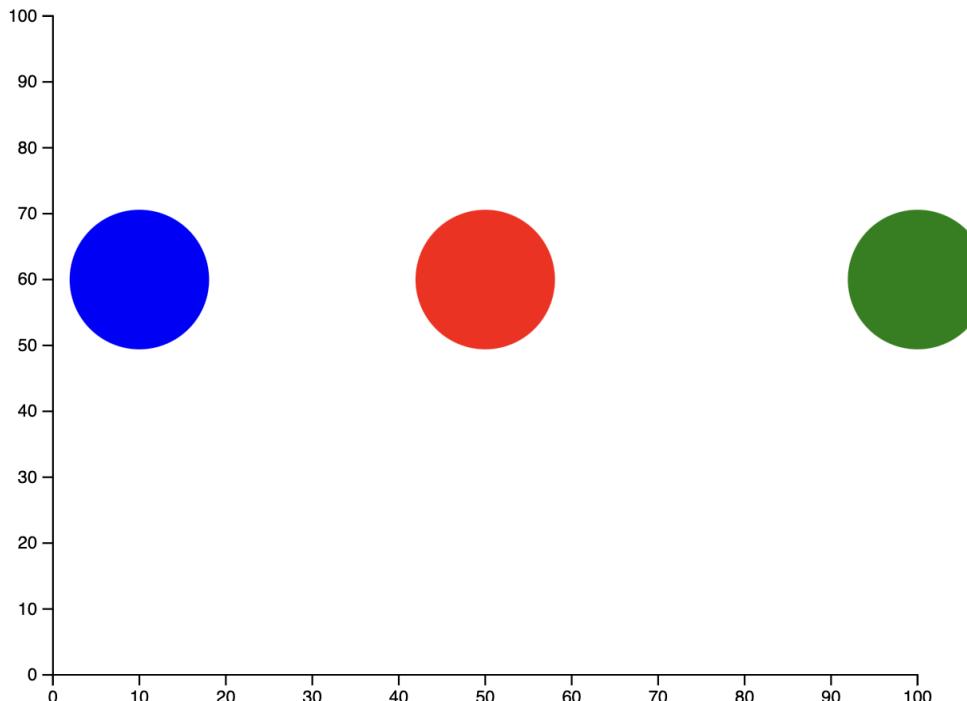
//dessiner l'axe x :
svg.append('g')
    .attr("transform", "translate(0," + height + ")")
    .call(d3.axisBottom(x));

//créer l'axe y :
const y = d3.scaleLinear()
    .domain([0,100])
    .range([height,0])

//dessiner l'axe y :
svg.append('g')
    .call(d3.axisLeft(y));

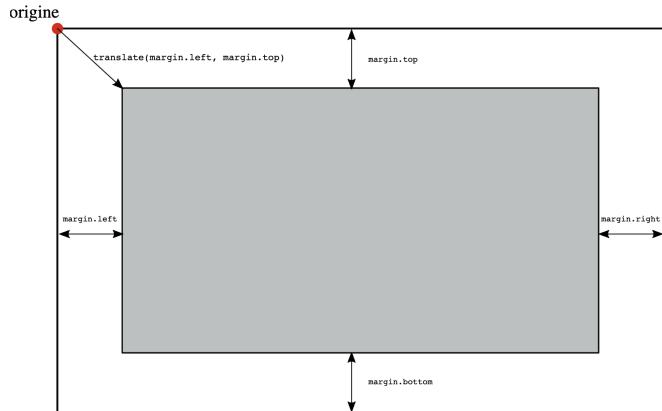
//ajout de cerles :
svg.append("circle")
    .attr("cx", x(10)).attr("cy", y(60)).attr("r", 40).style("fill", "blue");
svg.append("circle")
    .attr("cx", x(50)).attr("cy", y(60)).attr("r", 40).style("fill", "red");
svg.append("circle")
    .attr("cx", x(100)).attr("cy", y(60)).attr("r", 40).style("fill", "green");

```



Translation

```
translate(margin.left, margin.top)
```



Cours n°5 & 6 : Interaction et animation

Transitions

- Passer d'un état 1 à un état 2

```
let t = d3.transition()
    .duration(temps_milisecondes)
    .ease(fonction_easing) // "assouplissement" du mouvement

svg.append('element')
    .attr('attribute', Etat_1)
    .transition(t)
    .attr('attribute', Etat_2)
```

Fonctions d'easing :

- **easeLinear(*t*)**

The identity function; no easing.

- **easeQuadIn(*t*)**

Quadratic easing.

- **easeQuadOut(*t*)**

Reversed quadratic easing

→ fonctions prennent un temps en paramètre (valeur entre 0 et 1)

Easing Animations

Eased transitions exhibit more plausible motion. d3-ease implements various easing methods, demonstrated below, which take a normalized time and return an "eased" time. Click to restart the animation. See also easing graphs. Linear easeLinear(*t*) The identity function; no easing. Quadratic

<https://observablehq.com/@d3/easing-animations>

Linear

easeLinear(*t*)

The identity function; no easing.

Quadratic

easeQuadIn(*t*)

Quadratic easing.

Entrée, mise à jour et sortie

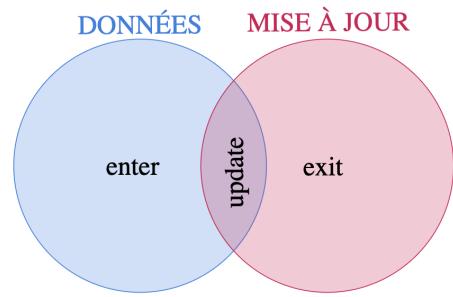
- Décrit le comportement de l'objet à l'arrivée, pendant sa présence et à sa sortie

```
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')
.attr('attribute', d => d.valeur)

```



Zoom

- Choisir sur quel élément le zoom s'applique (tout le svg, autre)

```

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

let zoom = d3.zoom()
  .on('zoom', handleZoom);

svg.call(zoom);

```

Contraintes :

- d3.zoom().contrainte(props)
 - scaleExtent([min, max])
 - translateExtent([minX,minY],[maxX, maxY])

Méthodes :

- svg.call(zoom.methode, props)
- `(zoom.scaleBy, facteur_échelle)` → Multiplie l'échelle actuelle par le facteur d'échelle
- `(zoom.scaleTo, échelle)` → Change l'échelle à échelle définie
- `(zoom.translateBy, x, y)` → Translation de x, y
- `(zoom.translateTo, x, y)` → Translation jusqu'à x, y

Brush

- Sélection rectangulaire avec la souris qui change les attributs des objets placés dans la sélection

```

let brush = d3.brush()
  .on('brush', handleBrush);

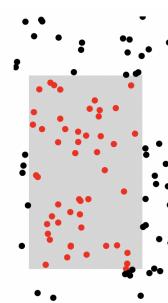
d3.select('svg')
  .call(brush);

let brushExtent;

function handleBrush(e) {
  brushExtent = e.selection();

  d3.select('svg')
    .selectAll('circle')
    .data(data)
    .join('circle')
    .attr('cx', d=>d.x)
    .attr('cy', d=>d.y)
    .attr('r', 4)
    .style('fill',
      d=> isInBrushExtent(d) ? 'red' : null);
}

```



```

}

function isInBrushExtent(d) {
    return brushExtent &&
        d.x >= brushExtent[0][0] &&
        d.x <= brushExtent[1][0] &&
        d.y >= brushExtent[0][1] &&
        d.y <= brushExtent[1][1];
}

```

Temps

Intervalle :

```
setInterval(function, intervalle) //démarrer l'intervalle de temps
```

```
clearInterval(function, intervalle) //annuler l'intervalle
```

- faire faire quelque chose (fonction) tout les x temps (intervalle)
- (l'intervalle prend des valeurs en millisecondes genre 500 ou 1000)

Cours n°7 & 8 : Cartographie web

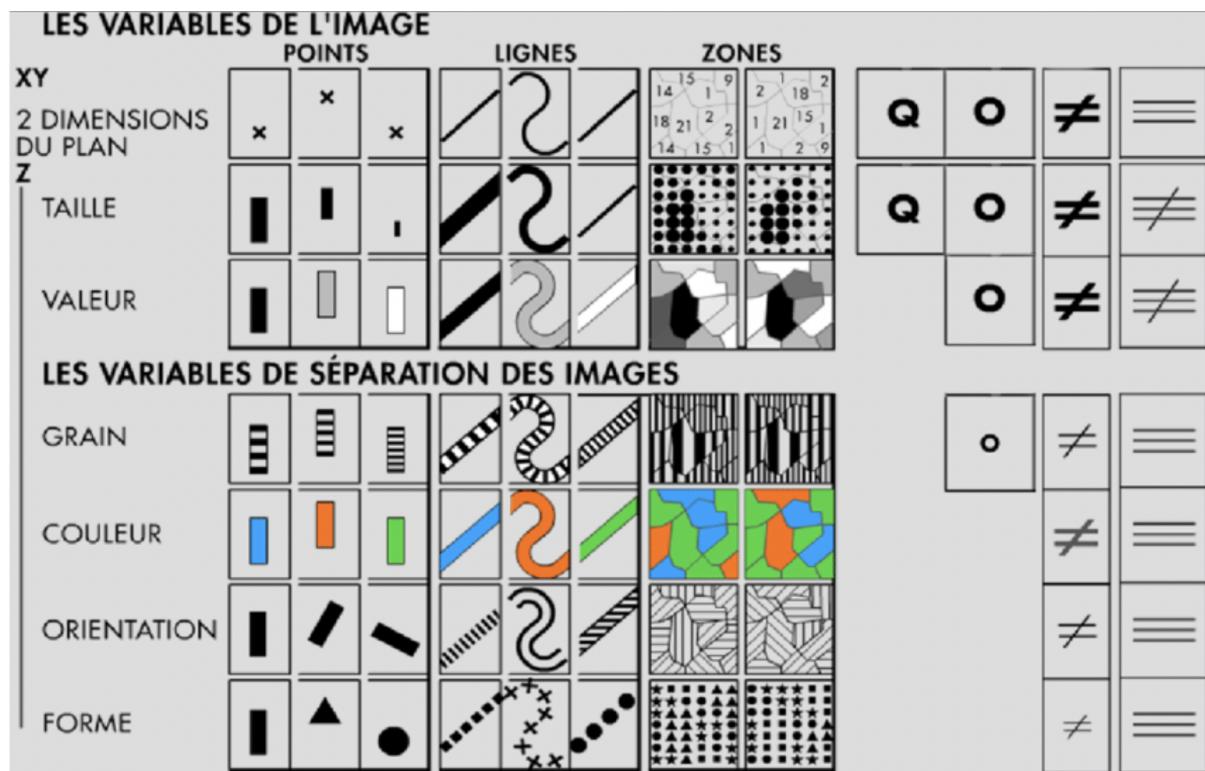
Types de carte

- Cartes de référence (Google Maps, Open Street Map, Maps.me)
- Cartes thématiques → qualitatives ou chloroplètes (carte de la suisse qui définit les cantons en couleurs)
- Cartogrammes
- Flowmap (représenté avec des points et des liens entre ces points)
- Cartes de chaleur
- Cartes topographiques (relief)
- Cartes topologiques (schéma d'utilisation pour transports public)

Projection

- Ensemble de techniques pour représenter une surface non plane sur une carte

Symbologie



Formats de données

- geojson (Json pour données géographiques)
- OSM → XML (Open Street Map)
- shp (shapefile)
- WKT (Well Known Text)

Sources de données

- Mondiales: [Natural Earth](#)
- Suisse: [opendata.swiss](#)
- OpenStreetMap (OSM)
- Ne pas utiliser Google Maps car pas open source

Récupérer les données

Overpass API

- Créer une requête
- Exécuter
- Exporter (geojson)

```
node
[amenity=school]
({{bbox}});
out;
```

overpass turbo

A web based data mining tool for OpenStreetMap which runs any kind of Overpass API query and shows the results on an interactive map.

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

Key:amenity



For describing useful and important facilities for visitors and residents. Facilities include for example toilets, telephones, banks, pharmacies, prisons and schools. Mapping varies widely depending on feature: As an area, e. g. . If the amenity shares a boundary with a building, both features are mapped on a single area

<https://wiki.openstreetmap.org/wiki/Key:amenity>

D3 (librairie js)

- Utilisation du path (vu avec SVG)

1. Définir la projection et le centre de la carte
2. Générer le path
3. Dessiner le path en fonction des données

```
//projection
let projection = d3.geoMercator()
    .fitSize([width,height], data)

//path
let path = d3.geoPath()
    .projection(projection)

//dessin en fonction de data (les données du geojson)
svg.selectAll("path")
    .data(data.features)
    .join(enter => enter.append('path')
        .attr("d",path)
        .attr("fill", "none")
        .attr("stroke-width",1))

//dessin d'un cercle avec des coordonnées définies (Yverdon)
svg.append('circle')
    .attr("cx", projection([6.6412, 46.7785])[0])
    .attr("cy", projection([6.6412, 46.7785])[1])
    .attr("fill", "red")
    .attr("r", 3)
```



Leaflet (librairie js)

- Récupérer des données OSM (par ex)

1. Créer un container
2. Générer un élément map centré en cx-cy, avec un zoom (scale)
3. Ajouter des éléments à l'élément map
4. Définir le fond de plan (base de la carte)

→ **HTML**

```
//importer Leaflet
<head>
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" integrity="sha512-xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9
    <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js" integrity="sha512-XQoYMqMTK8LvdXYG3nZ448h0EQiglfqkJs1NOQV44cWhUrBc8
</head>
//définir le container
<div id="mapid" style="width: 600px; height: 400px;"></div>
```

→ **JS**

```
let map = L.map('mapid').setView([cx, cy], zoom);
//le 'L' provient de Leaflet, faire attention à ce qu'il soit importé depuis le HTML
L.tileLayer(fondDeCarte).addTo(map);
L.Marker([cx, cy]).addTo(map);
L.geoJSON(objectGeojson).addTo(map);
```

Fond de carte :

```
Leaflet Provider Demo  
https://leaflet-extras.github.io/leaflet-providers/preview/
```

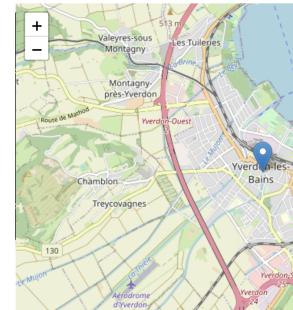
(tuto : <https://www.youtube.com/watch?v=wzsQj0ssC5M>)

Exemple :

```
let map = L.map('container').setView([46.7809, 6.6376], 14);

L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',
}).addTo(map);

L.geoJSON(
    {
        type: 'Feature',
        properties: { name: 'Yverdon' },
        geometry: { type: 'Point', coordinates: [6.6412, 46.7785] }
    },
    {
        onEachFeature: (feature, layer) => layer.bindPopup(feature.properties.name),
    }
).addTo(map)
```



Cours n°9 : Web scraping

Concept de base

- Extraire des données d'un site web

Méthodes

- Code asynchrone → async & await
- Isomorphic fetch
- jsdom
- puppeteer → pour les sites dynamiques

Puppeteer

- Librairie node.js
 - Connecte le code à un browser avec une API
 - Possibilité de faire des clicks, des screenshots, appliquer des fonctions aux données, scroller, ...
1. Importer puppeteer
 2. Créer une fonction async
 3. Créer des variables pour l'url, le browser et la page
 4. Lier la variable page à url
 5. Sélectionner les éléments qui nous intéressent avec des sélecteurs css (utile d'aller inspecter la page en question pour voir sa structure)
 6. Pour appliquer des fonctions aux sélecteurs, utiliser page.\$\$eval
 7. Pour faire un click, utiliser page.click
 8. Pour faire un screenshot, utiliser page.screenshot
 9. Ne pas oublier de fermer le browser après tous les traitements

```

import puppeteer from 'puppeteer';

(async () => {
  const url = 'urlexemple.com';
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto(url);

  await page.$(sélecteur); // document.querySelector
  await page.$$(sélecteur); // document.querySelectorAll
  // EXEMPLE : const div = await page.$$(div);

  //Appliquer une fonction aux sélecteurs
  const resultatFonction = await page.$$eval(sélecteur, pageFunction[...args]);
  // EXEMPLE : const divCount = await page.$$eval('div', (divs) => divs.length);

  //Cliquer sur un selecteur
  await page.click(sélecteur);

  //Faire un screenshot
  await page.screenshot({ path: 'example.png' });

  // Fermer les browser
  await browser.close();
})();

```

Autre

```

// Fonction permettant de séparer chaque 1000 par un ' (et chaque décimale par un .)
function numberWithApos(n) {
    var parts=n.toString().split(".");
    return parts[0].replace(/\B(?=(\d{3})+(?!\d))/g, "'") + (parts[1] ? "." + parts[1] : "");
}

```

Correction LH

```

// 1. Colorier la carte selon la population des pays
// Marges du graphique
const margin = {top: 0, right: 0, bottom: 0, left: 0},
      width = 1000 - margin.left - margin.right,
      height = 1000 - margin.top - margin.bottom;

// Nuances de couleurs selon la grandeur de la population
const color = d3.scaleLinear()
  .domain([5000000, 80000000]) // Population
  .range(["#e4d7f3", "#4a168b"]); // Degrés de couleurs selon la population

// Projection de carte (Mercator)
var projection = d3.geoMercator()
  .scale(120)
  .translate([width / 2, height / 2.8]);

// Générateur de chemin
var geoPath = d3.geoPath()
  .projection(projection);

// Création de SVG
var carte = d3.select("#carte")
  .append("svg")
  .attr("width", "100vw")
  .attr("height", "100vh");

var tooltipDiv = d3.select("#carte")
  .append("div")
  .attr("class", "tooltip")
  .style("opacity", 0);

carte.append("g").selectAll("path")
  .data(data.features)
  .enter()
  .append("path")

  .attr("data-population", function(d) {
    return d.properties.POP2005;
  })

```

```
.attr("data-country", function(d) {
    return d.properties.NAME;
})

.attr("d", geoPath) // Dessine chaque pays
.attr("fill", function(d) {
    return color(d.properties.POP2005); // Color (défini plus haut) repris avec le domaine et les nuances min et max
})
.style("stroke", "white")
.style('stroke-width', 0.3)

// 2. Rajouter une info-bulle avec la population du pays quand on y survole avec la souris
.on("mouseover", function(d) {
    d3.select(this)
        .style("opacity", 1)
        .style("stroke", "white")
        .style("stroke-width", "3")

    tooltipDiv.transition()
        .duration(200)
        .style("opacity", 1);
    tooltipDiv.html(
        "Pays : " + this.dataset.country + "<br>" + "Population : " + numberWithApos(this.dataset.population))
        .style("left", (d.clientX) + "px")
        .style("top", (d.clientY + 100) + "px");
})

.on("mouseleave", function(d) {
    d3.select(this)
        .style("opacity", 0.8)
        .style("stroke", "white")
        .style("stroke-width", "0.3");

    tooltipDiv.transition()
        .duration(500)
        .style("opacity", 0);
});

});
```

Correction vincent

```
import * as d3 from 'd3'

// Import des données
import data from '../data/countries.geojson'

// EXERCICE 1
const ex1 = d3.select("body").append("div").attr("id", "ex1");

ex1.append("h1").text("Exercice 1");

const ex1svg = ex1.append("svg").attr("width", "200px").attr("height", "200px");

const ex1grid = ex1svg.append("path").attr("d", "M0 0 L0 100 L10 100 L10 0 L20 0 L20 100 L30 100 L30 0 L40 0 L40 100 L50 100 L");

const ex1drawing = ex1svg.append("path").attr("d", "M20 10 L20 70 L70 70 L70 10 L30 10 L30 60 L60 60 L60 30 L40 30 L40 50 L");

const ex1drawingCircle = ex1svg.append("circle").attr("cx", "50").attr("cy", "40").attr("r", "5").attr("fill", "red");

// EXERCICE 2

const continentsNames = {
  142: "Asie",
  150: "Europe",
  2: "Afrique",
  9: "Océanie",
  19: "Amériques"
}

const ex2 = d3.select("body").append("div").attr("id", "ex2");

ex2.append("h1").text("Exercice 2");

const populations = data.features.filter(feature => feature.properties.POP2005 > 1000000);

let continents = Array.from(d3.group(populations, d => d.properties.REGION))
  .map(continent => {
    return {
      name: continent[0].properties.REGION,
      count: continent.length
    }
  })
```

```

        'code': continentsNames[continent[0]],
        'average': Math.round(continent[1]
            .reduce((acc, curr) => acc + curr.properties.POP2005, 0) / continent[1].length)
    );
}
);
console.log("populations", populations);
console.log("continents", continents);

ex2.append("p").text("Pays > 1 million de population " + populations.length);
ex2.append("p").text("Continent " + continents.length);
ex2.append("h5").text("Voir la console pour les détails");



// EXERCICE 3

const ex3 = d3.select("body").append("div").attr("id", "ex3");

ex3.append("h1").text("Exercice 3");

const ex3svg = ex3.append("svg").attr("width", "100vw").attr("height", "300px");

var projection = d3.geoMercator();

var path = d3.geoPath()
    .projection(projection);

console.log([
    d3.min(data.features.map(feature => feature.properties.POP2005)),
    d3.max(data.features.map(feature => feature.properties.POP2005))
]);
var myColor = d3.scaleSqrt()
    .domain([
        d3.min(data.features.map(feature => feature.properties.POP2005)),
        d3.max(data.features.map(feature => feature.properties.POP2005))
    ])
    .range(["white", "blue"])


var tooltipDiv = d3.select("body").append("div")
    .attr("class", "tooltip")
    .style("opacity", 0);

ex3svg.selectAll("path")
    .data(data.features)
    .enter()
    .append("path")
    .attr("data-population", function(d) {
        return d.properties.POP2005;
    })
    .attr("data-country", function(d) {
        return d.properties.NAME;
    })
    .attr("d", path)
    .attr("fill", function(d) {
        return myColor(d.properties.POP2005);
    })
    .attr("stroke", "black")
    .attr("stroke-width", "1px")
    .on("mouseover", function(d) {
        tooltipDiv.transition()
            .duration(200)
            .style("opacity", 1);
        tooltipDiv.html(this.dataset.country + " " + this.dataset.population)
            .style("left", (d.clientX) + "px")
            .style("top", (d.clientY + 100) + "px");
    })
    .on("mouseout", function(d) {
        tooltipDiv.transition()
            .duration(500)
            .style("opacity", 0);
    });
});


// CHART BAR

continents = continents.sort((b, a) => a.average - b.average);

const ex3ChartWidth = 500;
const ex3ChartHeight = 500;

const ex3Chart = d3.select("body").append("div").attr("id", "ex3Chart");

ex3Chart.append("h1").text("Exercice 3 Chart");

const ex3ChartsVG = ex3Chart.append("svg").attr("width", "550px").attr("height", "600px").attr("transform", "rotate(90)");

```

```

console.log(continents.map(c => c.code));

var x = d3.scaleBand()
  .range([0, ex3ChartWidth])
  .domain(continents.map(c => c.code))
  .padding(0.2);

ex3ChartSVG.append("g")
  .attr("transform", "translate(0," + ex3ChartHeight + ")")
  .call(d3.axisBottom(x))
  .selectAll("text")
  .attr("transform", "translate(-13,10)rotate(-90)")
  .style("text-anchor", "end");

// Add Y axis
var y = d3.scaleLinear()
  .domain([0, d3.max(continents.map(c => c.average))])
  .range([ex3ChartHeight, 0]);

ex3ChartSVG.append("g")
  .call(d3.axisLeft(y));

// Bars
ex3ChartSVG.selectAll("mybar")
  .data(continents)
  .enter()
  .append("rect")
  .attr('data-name', d => d.code)
  .attr('data-pop', d => d.average)
  .attr("x", function(d) { return x(d.code); })
  .attr("y", function(d) { return y(d.average); })
  .attr("width", x.bandwidth())
  .attr("height", function(d) { return ex3ChartHeight - y(d.average); })
  .attr("fill", "#69b3a2")
  .on("mouseover", function(d) {
    tooltipDiv.transition()
      .duration(200)
      .style("opacity", 1);
    tooltipDiv.html(this.dataset.name + " " + this.dataset.pop)
      .style("left", (d.clientX) + "px")
      .style("top", (d.clientY + 500) + "px");
  })
  .on("mouseout", function(d) {
    tooltipDiv.transition()
      .duration(500)
      .style("opacity", 0);
  });
});

//EXERCICE 4

/* const url = 'https://heig-vd.ch/formations/bachelor/filieres';

//screenshot
(async() => {
  const browser = await puppeteer.launch()
  const page = await browser.newPage()
  await page.setViewport({ width: 1280, height: 800 })
  await page.goto(url)
  await page.screenshot({ path: 'screenshot.png' })
  await browser.close()
})();

// filieres
(async() => {
  const response = await axios.get(url)
  const dom = new JSDOM(response.data);
  let fillieresTd = dom.window.document.querySelectorAll(".liste-formations td");
  //console.log(p[0].querySelectorAll("div.ratings p")[1].getAttribute("data-rating"));
  const fillieresName = [];

  const orientations = [];

  for (const td of fillieresTd) {
    if (td.classList.contains("prog")) {
      fillieresName.push(td.innerText);
    }
  }
  console.log(fillieresName);

  for (const td of fillieresTd) {

```

```
        if (td.classList.contains("ori")) {
            orientations.push(td.innerText);
        }
    }

    console.log(orientations);

})(); */

```

Structure examen

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Examen type</title>
</head>
<body>
    <div class="ex-1">
        </div>
        <div class="ex-2">
            </div>
            <div class="ex-3">
                <div class="map">
                    </div>
                    <div class="graph">
                        </div>
                </div>
            </div>
        </div>
    </body>
</html>
```