

# Formation D3

Partie 1 : Les fondamentaux de D3js

# Introduction

# D3 ?

- D3 comme « **Data-Driven Documents** »
- Conçu pour créer des visualisations des données complexes avec les standards du Web
- **Visualisation** des données != des camemberts ou des courbes
  - Diagramme interactif, tableaux de bords, rapports, ...

# Historique

- Mike Bostock a écrit D3 pendant ses études de doctorat à Stanford.
- Il travaille actuellement au New York times qui sponsorise ses travaux open source.
- Son profil Github :  
<https://github.com/mbostock>



# Historique



**flare**

**prefuse**  
INFORMATION VISUALIZATION TOOLKIT



**Protovis**

2005

2007

2009

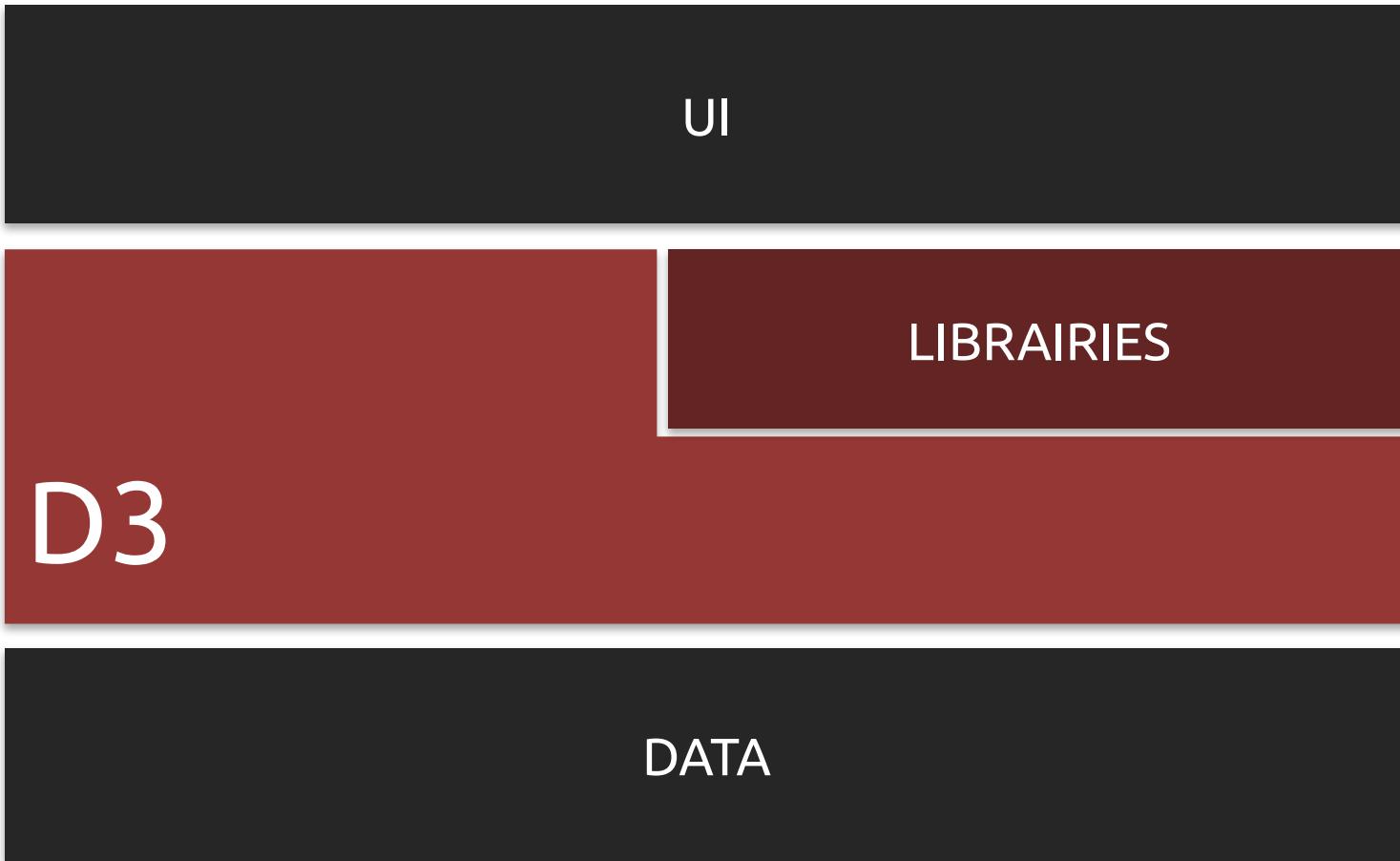
2011



# Principes fondateurs de D3

- Utiliser les standards du Web (HTML 5, CSS, JavaScript) et éviter les technologies propriétaires (Java, Flash, SilverLight, ...)
- Permettre un niveau de personnalisation élevé  
=> D3 est une librairie "bas niveau"

# D3 ?



# <http://c3js.org/>

C3.js | D3-based reusable chart library

[Getting Started](#)

Examples

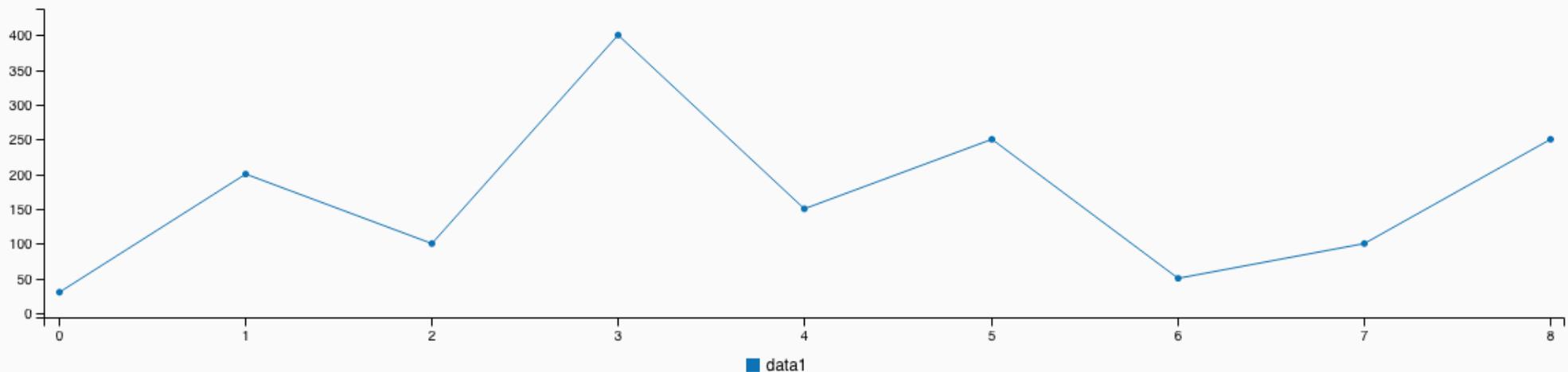
Reference

Forum

Source

## C3.js D3-based reusable chart library

[Start Demo](#)



# <http://nvd3.org/>

NVD3 Re-usable charts for d3.js

This project is an attempt to build re-usable charts and chart components for [d3.js](#) without taking away the power that [d3.js](#) gives you. This is a very young collection of components, with the goal of keeping these components very customizable, staying away from your standard cookie cutter solutions.

[View more examples »](#)

[GitHub Repo](#)

● Grouped   ○ Stacked   Stream0   Stream1   Stream2

10.3  
9.0  
8.0  
7.0  
6.0  
5.0  
4.0  
3.0  
2.0  
1.0  
0.0

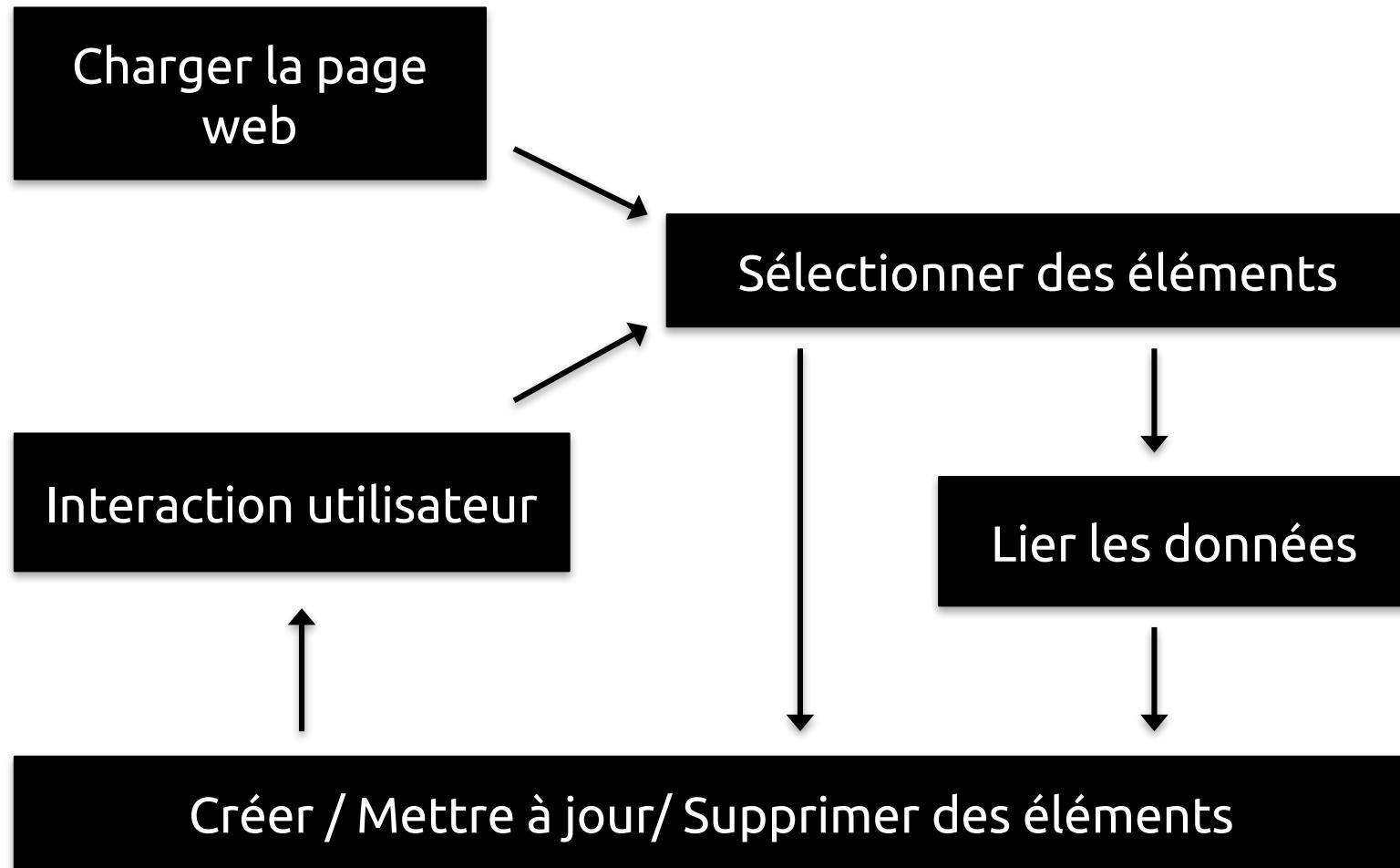
10.3  
8.0  
6.0  
4.0  
2.0  
0.1

11.2  
10.0  
8.0  
6.0  
4.0  
2.0  
0.0

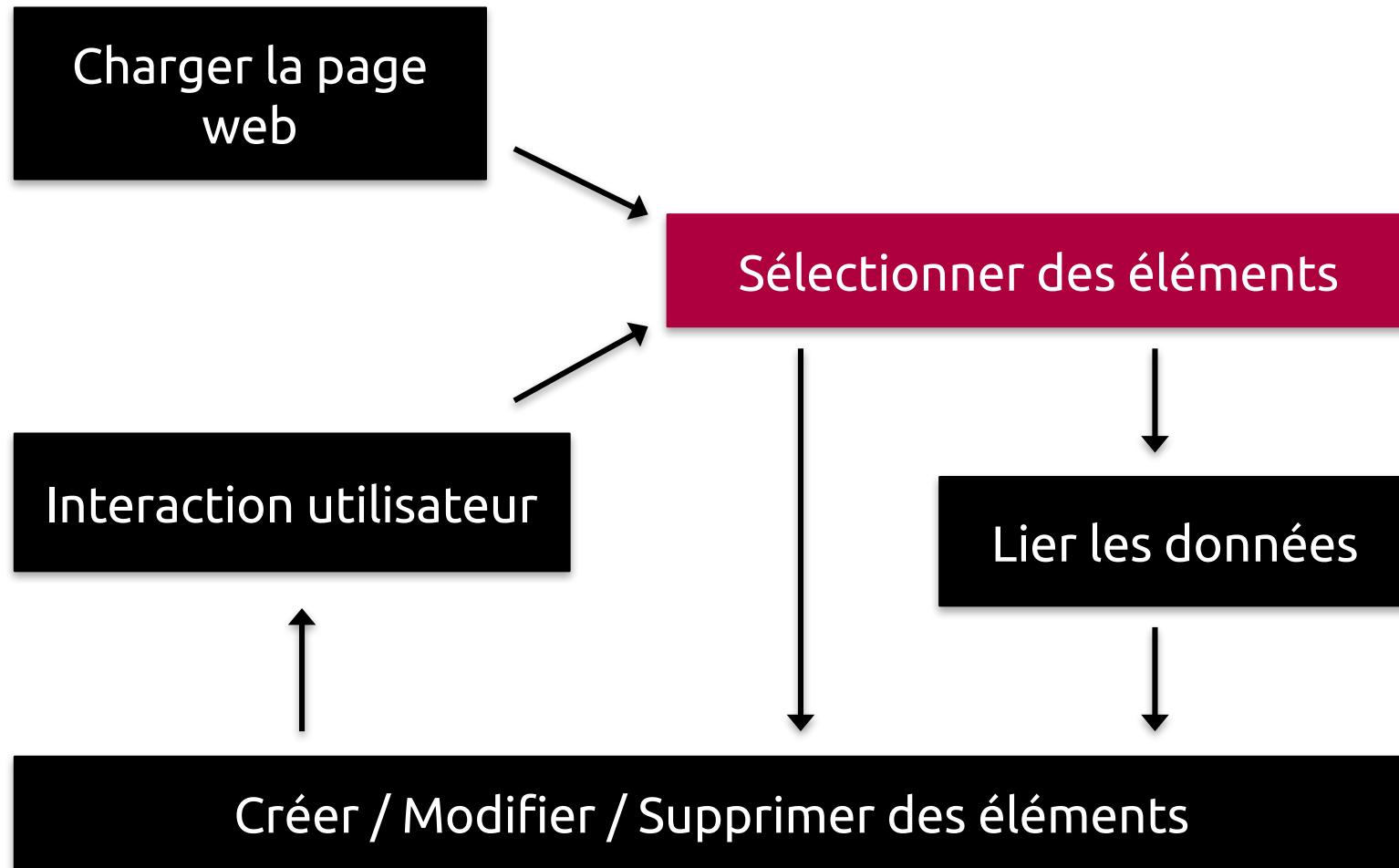
9

# Premiers pas avec D3

# Travailler avec D3js



# Sélectionner des éléments



# select, selectAll

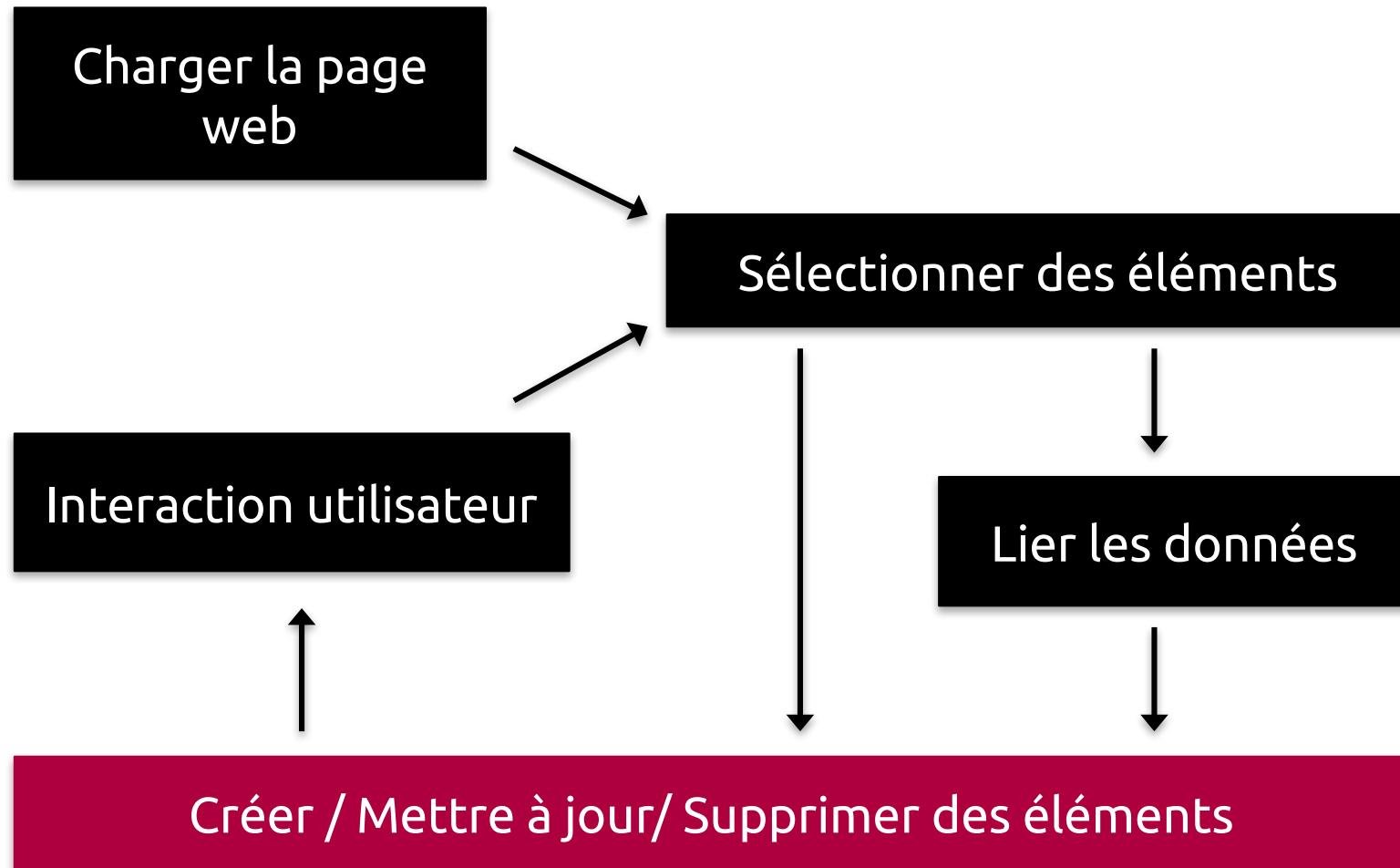
- Utilise le W3C Selectors API
  - `document.querySelector`,  
`document.querySelectorAll`
- **selectAll** : récupère tous les éléments correspondant au pattern fourni.
- **select** : récupère le premier élément correspondant au pattern fourni.

```
d3.select("svg")
```

```
d3.selectAll("div.cities")
```

```
d3.select("#city")
```

# Travailler avec des éléments



# Modifier un attribut

```
d3.select("div").attr("id", "nancy");

d3.selectAll("div").attr("id", "nancy");

d3.select("div").attr("id", function(d,i) {
    return "nancy";
});

d3.selectAll("div").attr("id", function(d,i) {
    return "nancy";
});
```

# Récupérer la valeur d'un attribut

```
var idValue = d3.select("div").attr("id");
```

# Modifier l'attribut style

```
d3.select("svg").style("width", "500px");

d3.select("svg").style("height", function() {
    return Math.random() * 500 + "px";
});

d3.selectAll("svg").style("width", "500px");

d3.selectAll("svg").style("height", function() {
    return Math.random() * 500 + "px";
});
```

# Modifier l'attribut class

```
d3.select("div").classed("foo", true);
```

```
d3.select("div").classed("foo", false);
```

```
d3.select("div").classed("foo", { 'foo' : true, 'bar' : false });
```

```
d3.select("div").classed("foo bar", true);
```

# Récupérer la valeur d'un style

```
var widthValue = d3.select("div").style("width");
```

# append()

```
d3.select("div").append("p"); // ajoute l'élément <p></p>
```

# insert()

```
d3.select("div").insert("p"); // ajoute l'élément <p></p>
```

```
d3.select("div").insert("p", ":first-child"); // ajoute l'élément <p></p> avant le  
// premier élément enfant.
```

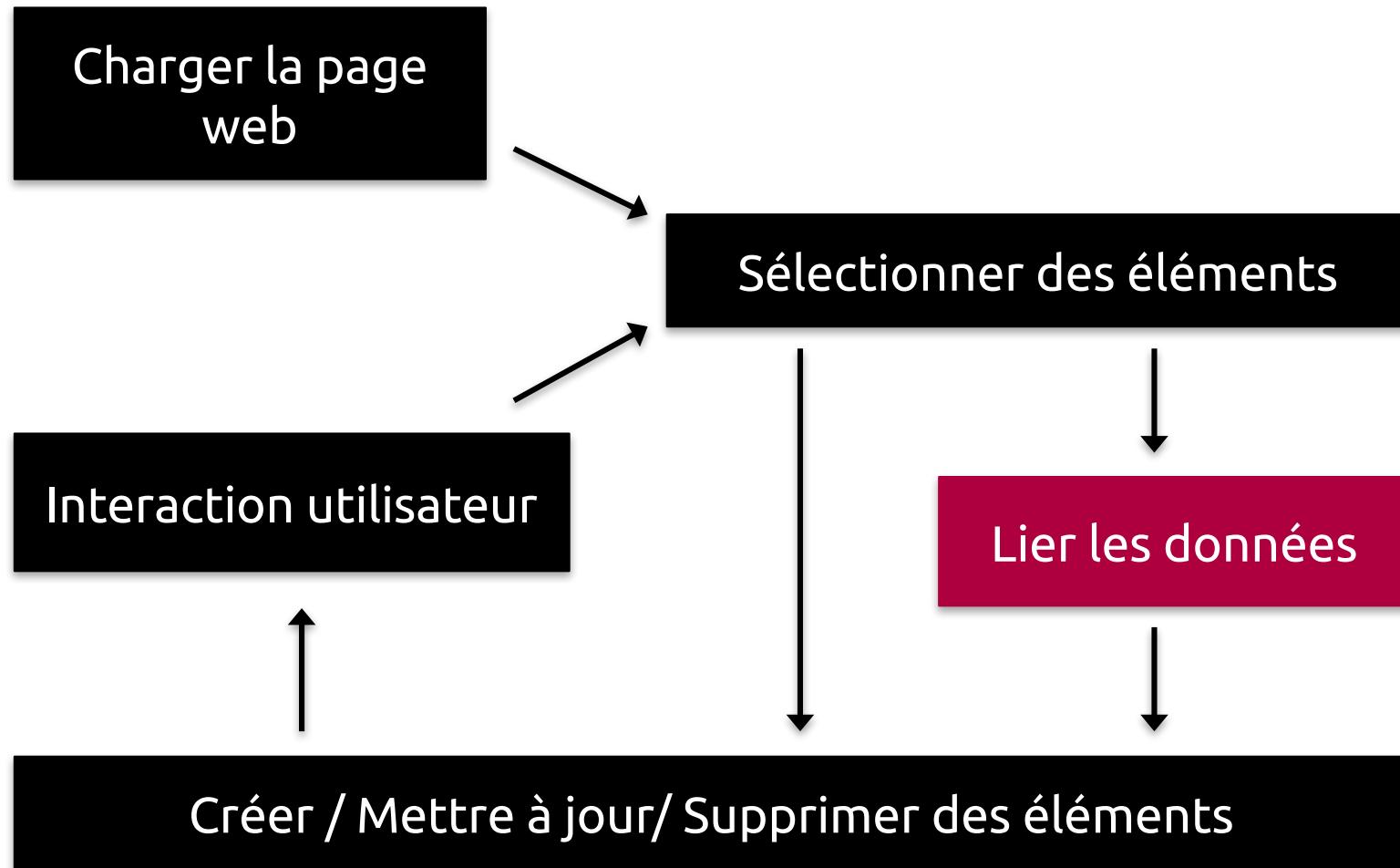
# html()

```
d3.select("div").html("<span>Bob</span>"); // défini du contenu HTML
```

# Chainage des appels

```
d3.select("div").style("width", "500px")
    .style("height", function() {
        return Math.random() * 500 + "px";
    })
    .classed("foo", true)
    .html("<span>Hello</span>");
```

# Lier les données



# datum()

```
var myData = [1, 12, 3, 255, 14, 345];  
  
d3.select("body").selectAll("div")  
    .datum(myData)  
    .append("div").html(function(d) { return d;})
```

d  
vaut myData



# data(), enter(), exit()

```
var myData = [1, 12, 3, 255, 14, 345];
```

d  
une valeur de myData

```
d3.select("body").selectAll("div")
  .data(myData)
  .enter().append("div").html(function(d) { return d;})
  .exit().remove();
```

## data()

Lier une source de données

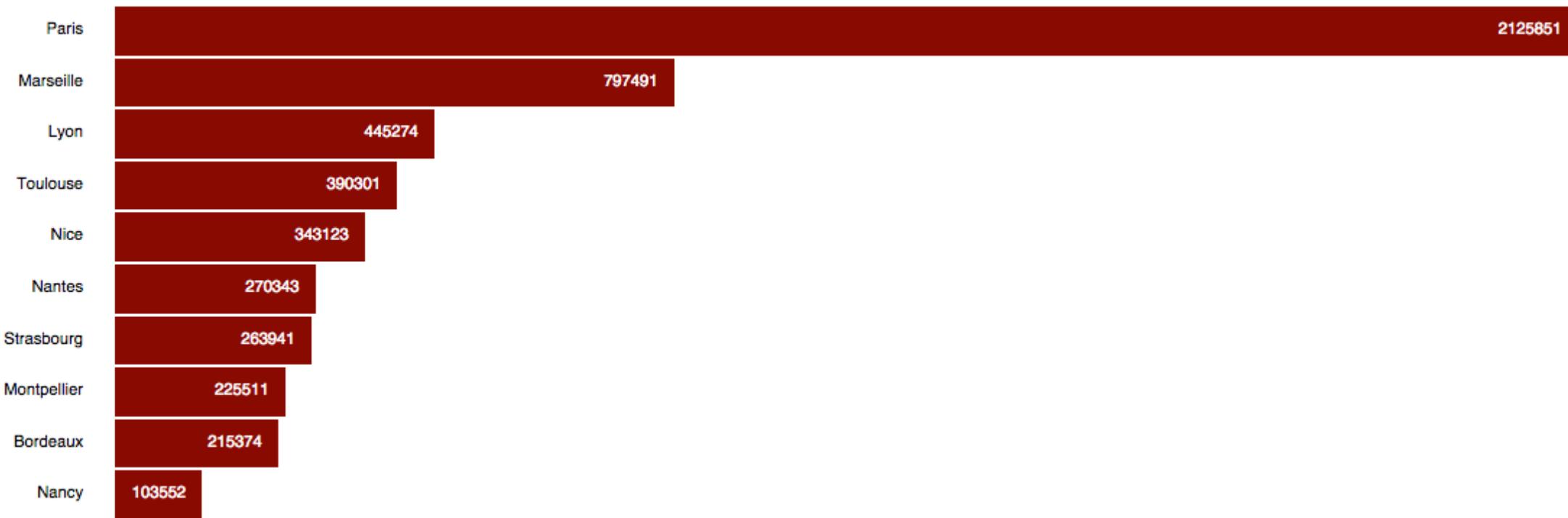
## enter()

Eléments à ajouter pour chaque donnée n'ayant pas de <div> correspondante

## exit()

Permet de gérer le cas où il y a plus d'éléments <div> que de données. Dans l'exemple, les <div> en trop sont supprimées.

# TP 1 – Graphique HTML



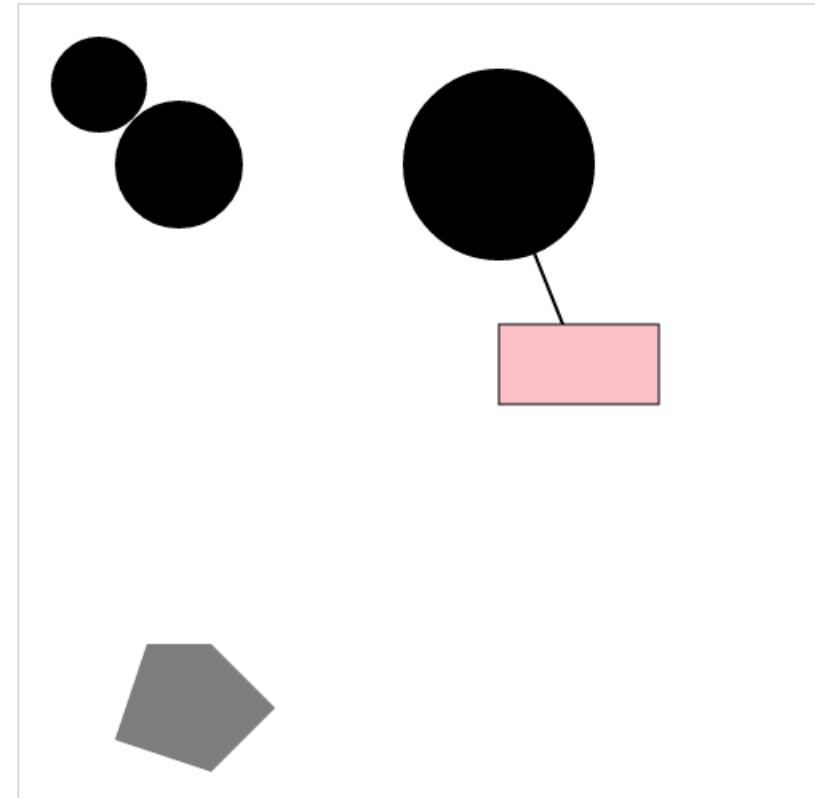
# SVG

# SVG ?

- « **S**calable **V**ector **G**raphics »
- Permet de créer une représentation mathématique d'une image
  - Cela permet par exemple de modifier la taille de l'image sans altérer sa qualité visuelle
  - Technique à l'opposé des images Bitmap qui stockent des points
- Grâce à sa syntaxe dérivée du XML, il peut être inclus dans différents types de documents (HTML, XML, XUL, ...)
- IE 9+

# Exemple de document SVG

```
<svg style="width:500px;height:500px;border:1px  
lightgray solid;">  
  
<circle cy="100" cx="100" r="40"/>  
<circle cy="50" cx="50" r="30"/>  
  
<g>  
  <line x1="300" y1="100" x2="350" y2="225"  
        style="stroke:black;stroke-width:2px;" />  
  <circle cy="100" cx="300" r="60"/>  
  <rect x="300" y="200" width="100" height="50"  
        style="fill:pink;stroke:black;stroke-width:1px;" />  
</g>  
<polygon style="fill:gray;" points="80,400 120,400  
          160,440 120,480 60,460" />  
</svg>
```

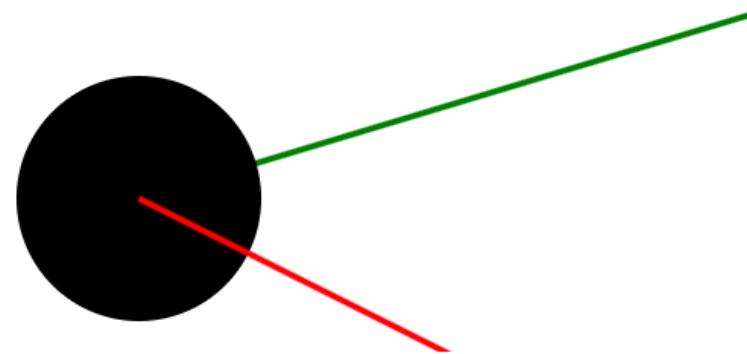


# Balise <svg>

- **<svg>**
  - **xmlns** : définition de l'espace de nommage SVG.
  - **version** : version du standard SVG (1.0, 1.1, 1.2)
  - **basicProfile** : profile (none, full, basic, tiny)
  - **style** : style CSS appliqué à la balise SVG.
  - ...

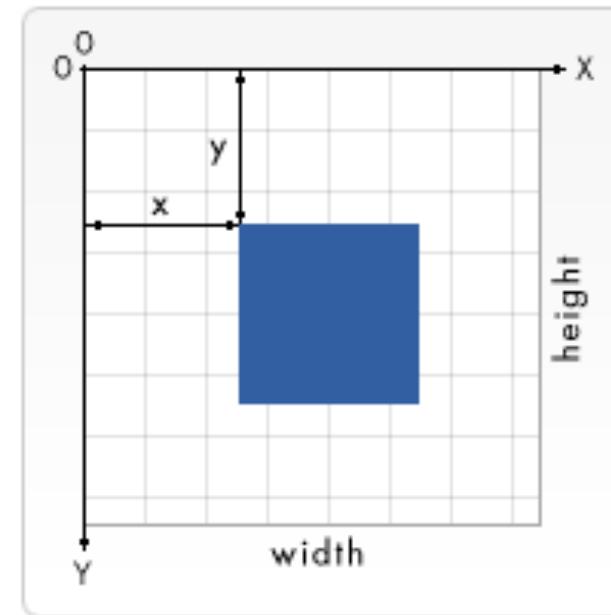
# SVG & Navigateur

- Ordre de rendu des éléments
  - Les éléments qui sont déclarés le plus récemment sont ceux qui seront affichés en avant des autres.  
L'élément défini en bas du document s'affichera au dessus des autres.



# SVG & Positionnement

- La grille
  - Le point haut-gauche est considéré comme le point (0,0).
  - Le positionnement est mesuré en pixel (le périphérique peut décider ce que vaut 1px dans son contexte).
  - Les valeurs positives de x vont vers la droite, les valeurs positives de y vont vers le bas.



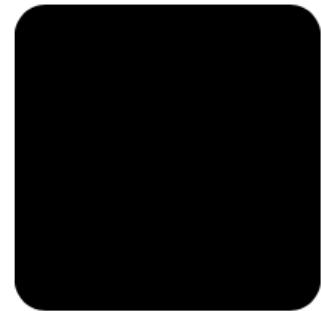
# Figures

- Rectangle : *<rect>*
- Cercle : *<circle>*
- Ellipse : *<ellipse>*
- Ligne : *<line>*
- Ligne brisée : *<polyline>*
- Polygone : *<polygon>*
- Chemin : *<path>*

# Rectangle

- **Elément <rect>**

- **x** : position du rectangle sur l'axe horizontal par rapport au coin supérieur gauche.
- **y** : position du rectangle sur l'axe vertical par rapport au coin supérieur gauche.
- **width** : largeur du rectangle.
- **height** : hauteur du rectangle.
- **rx** : rayon x des coins du rectangle.
- **ry** : rayon y des coins du rectangle.



```
<svg>
```

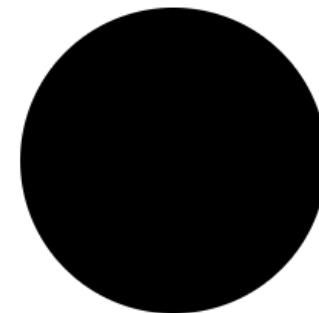
```
  <rect x="10" y="10" width="100" height="100"></rect>
  <rect x="150" y="10" rx="10" ry="10" width="100" height="100"></rect>
```

```
</svg>
```

# Cercle

- **Elément <circle>**

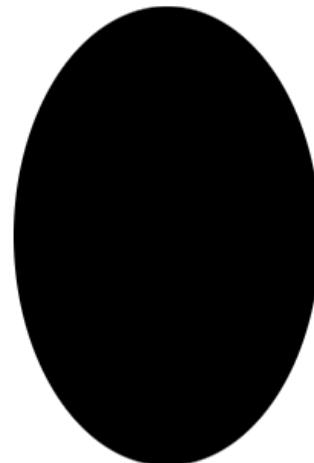
- **r** : rayon du cercle.
- **cx** : position x du centre du cercle.
- **cy** : position y du centre du cercle.



```
<svg>
  <circle cx="50" cy="50" r="50"></circle>
</svg>
```

# Ellipse

- **Elément <ellipse>**
  - **rx** : rayon x de l'ellipse.
  - **ry** : rayon y de l'ellipse.
  - **cx** : position x du centre de l'ellipse.
  - **cy** : position y du centre de l'ellipse.

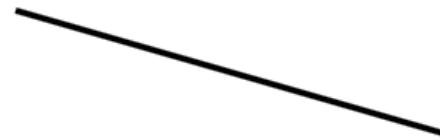


```
<svg>
  <ellipse cx="75" cy="75" rx="50" ry="75"></ellipse>
</svg>
```

# Ligne

- **Elément <line>**

- **x1** : position x du premier point.
- **x2** : position x du deuxième point.
- **y1** : position y du premier point.
- **y2** : position y du deuxième point.



```
<svg>
  <line x1="10" x2="150" y1="10" y2="50"></line>
</svg>
```

# Ligne brisée

- **Elément <polyline>**
  - **points** : liste des points, chaque paire de nombres correspondant aux coordonnées x et y de chaque point. Chaque position x est séparée de la position y par un espace. Chaque ensemble de coordonnées est séparé du suivant par une virgule.

```
<svg>
  <polyline points="100 10, 300 30, 300 100, 20 100"></polyline>
</svg>
```



# Polygone

- **Elément <polygon>**
  - **points** : Idem que l'attribut points de l'élément *polyline*. Une dernière ligne ferme automatiquement la forme en retournant au point de départ.

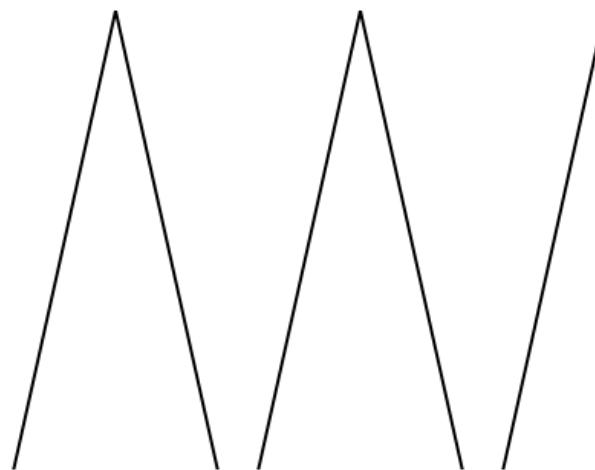


```
<svg>
  <polygon points="10 10, 50 50, 75 100, 10 100"></polygon>
</svg>
```

# Chemin

- **Elément <path>**
  - d: un ensemble d'information définissant le chemin à dessiner.
  - Il s'agit de la forme la plus généraliste qui peut être utilisée avec SVG.

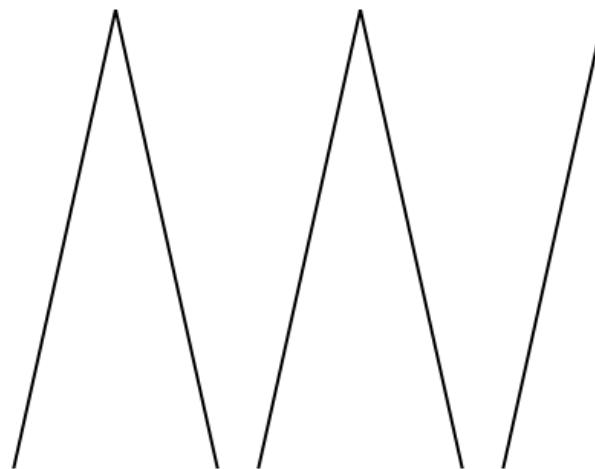
```
<svg>
  <path d="M 100,180 L 140,0 L 180,180 L 220,0 L 260,180 L 300,0 L 330,180"
        style="fill:none; stroke:black"></path>
</svg>
```



# Chemin

- **Quelques commandes**

- **M** (Moveto) : prend comme paramètre les coordonnées du premier point du tracé.
- **L** (Lineto) : trace une ligne entre les dernières coordonnées mentionnés.
- **Z** (ClosePath) : ferme le tracé.
- **H** (Lineto horizontal) : raccourci pour noter des lignes horizontales.
- **V** (Lineto vertical) : raccourci pour noter des lignes verticales.
- ...



```
<svg>
```

```
  <path d="M 100,180 L 140,0 L 180,180 L 220,0 L 260,180 L 300,0 L 330,180"
        style="fill:none; stroke:black"></path>
```

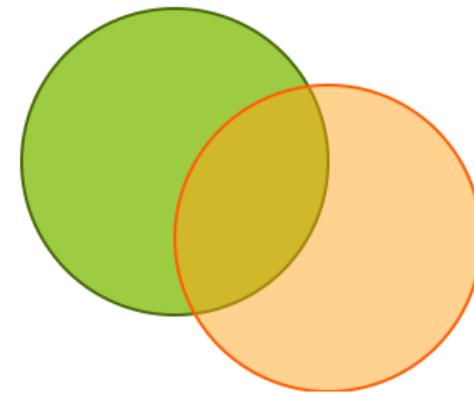
```
</svg>
```

# Remplissages

- **Attributs**

- **stroke** : couleur du contour.
- **stroke-width** : épaisseur du contour.
- **stroke-opacity** : opacité du contour (de 0 à 1).
- **stroke-dasharray** : pointillés.
- **fill** : couleur de remplissage.
- **fill-opacity** : opacité du remplissage (de 0 à 1).

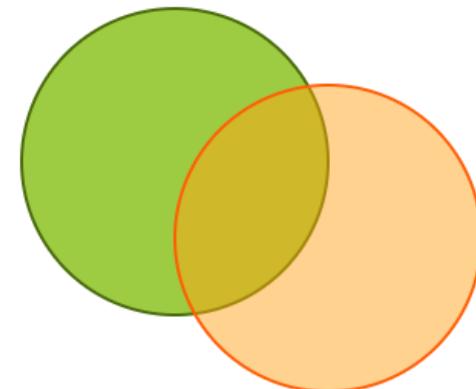
```
<svg>
  <circle cx="100" cy="75" r="50" stroke="#4b6c0b" fill="#9dcc41" ></circle>
  <circle cx="150" cy="100" r="50" stroke="#ff6000" fill="orange" fill-opacity="0.5" ></circle>
</svg>
```



# Grouper plusieurs formes

- **Elément <g>**

- Permet de regrouper plusieurs formes dans un seul élément afin de pouvoir les réutiliser ensemble par la suite.
- L'élément *g* (*group*) est souvent accompagné d'un identifiant *id* pour le désigner de manière unique.



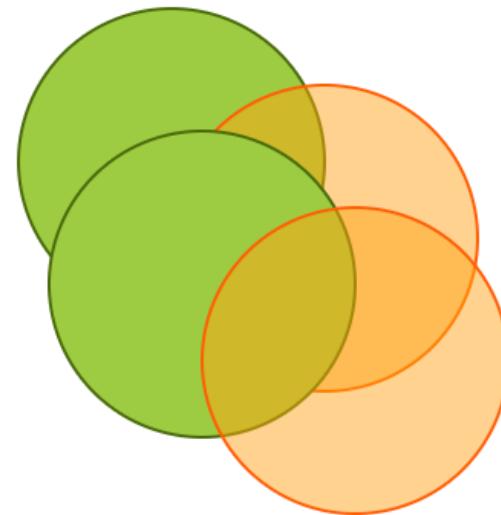
```
<svg>
  <g id="cercles">
    <circle cx="100" cy="75" r="50" stroke="#4b6c0b" fill="#9dcc41" ></circle>
    <circle cx="150" cy="100" r="50" stroke="#ff6000" fill="orange" fill-opacity="0.5" ></circle>
  </g>
</svg>
```

# Réutiliser des formes

- **Réutilisation**

- Les définitions de formes sont regroupées entre les balises **<defs>** et **</defs>**. Par défaut, les formes sont cachées.
- La balise **<use>** permet de réutiliser ces ensembles.

```
<svg>
<defs>
  <g id="cercles">
    <circle cx="100" cy="75" r="50" stroke="#4b6c0b" fill="#9dcc41" ></circle>
    <circle cx="150" cy="100" r="50" stroke="#ff6000" fill="orange" fill-opacity="0.5" ></circle>
  </g>
</defs>
<use xlink:href="#cercles" x="10" y="10"></use>
<use xlink:href="#cercles" x="20" y="50"></use>
</svg>
```



# Texte

- **Elément <text>**

- **x** : position du texte sur l'axe horizontal par rapport au coin supérieur gauche.
- **y** : position du texte sur l'axe vertical par rapport au coin supérieur gauche.
- **style** : (font-weight, font-style, text-decoration, font-size, ...)

Ceci est un texte SVG

```
<svg>
    <text x="10", y="30">Ceci est un texte SVG</text>
</svg>
```

# Chemin du texte

- **Elément <textpath>**
  - **xlink:** référence vers la définition d'une courbe

```
<svg>
<defs>
    <path id="courbe" d="M 30 40 C 50 10, 70 10, 120 40 S 150 0, 200 40"></path>
</defs>
<text>
    <textpath xlink:href="#courbe"> Le texte qui suit la courbe</textpath>
</text>
</svg>
```

*Le texte qui suit la courbe*

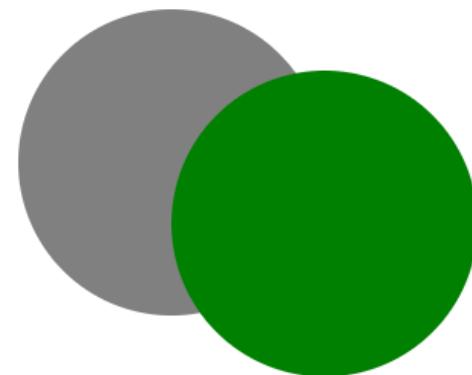
# Transformation SVG

- L'attribut ***transform*** permet de définir une transformation SVG.
- Plusieurs types de transformations :
  - translate
  - rotate
  - scale
  - skewX
  - skewY

# Translation

- **translate(x,y)**

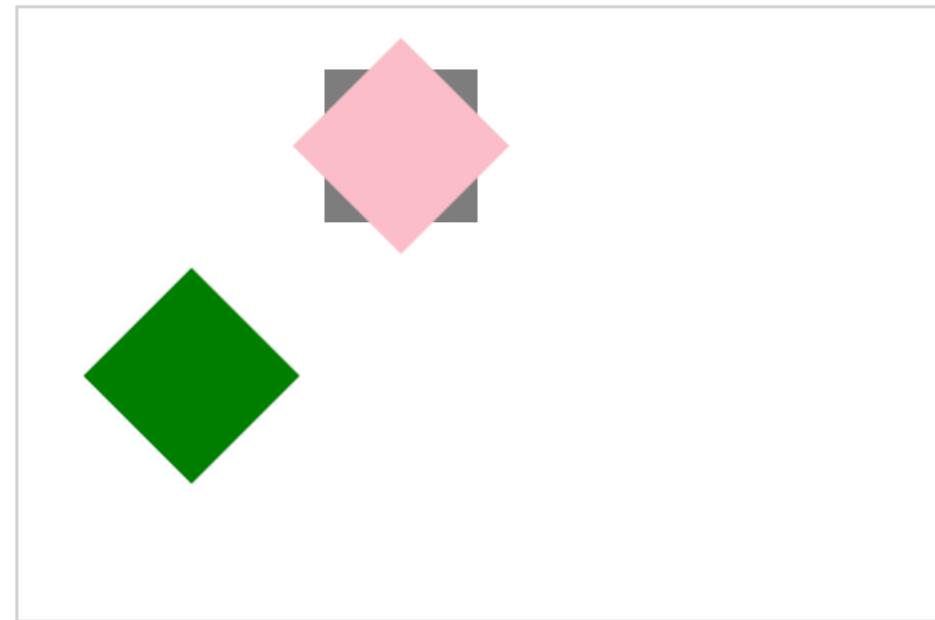
- **x**: déplacement suivant l'axe horizontal.
- **y**: déplacement suivant l'axe vertical.



```
<svg>
  <circle cx="100" cy="75" r="50" fill="gray"></circle>
  <circle cx="100" cy="75" r="50" fill="green" transform="translate(50,20)"></circle>
</svg>
```

# Rotation

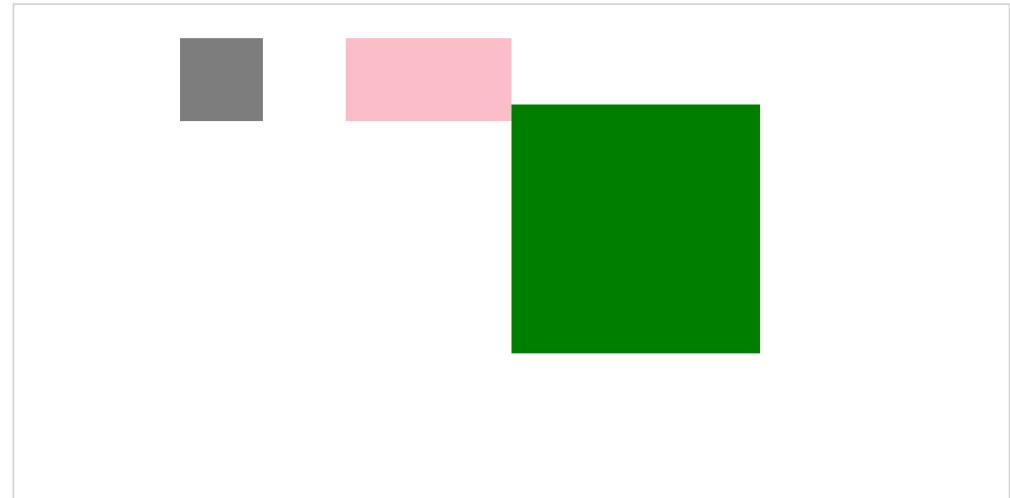
- **rotate(angle,x,y)**
  - **angle** : valeur en degré de la rotation.
  - **x (optionnel)** : abscisse du centre de la rotation
  - **y (optionnel)**: ordonnée du centre de la rotation



```
<svg>
  <rect x="100" y="20" width="50" height="50" fill="gray"></rect>
  <rect x="100" y="20" width="50" height="50" fill="green" transform="rotate(45)"></rect>
  <rect x="100" y="20" width="50" height="50" fill="pink" transform="rotate(45,125,45)"></rect>
</svg>
```

# Changement d'échelle

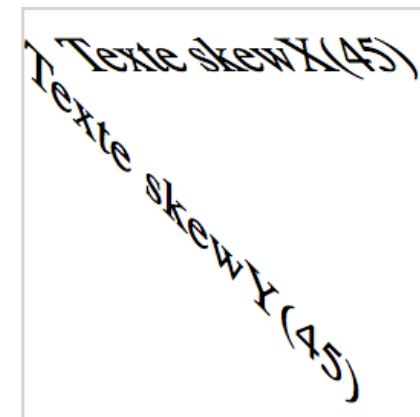
- **scale(x,y)**
  - **x** : échelle des abscisses.
  - **y** : échelle des ordonnées. **y** prends la même valeur que **x** si sa valeur n'est pas définie.



```
<svg>
  <rect x="100" y="20" width="50" height="50" fill="gray" ></rect>
  <rect x="100" y="20" width="50" height="50" fill="green" transform="scale(3)"></rect>
  <rect x="100" y="20" width="50" height="50" fill="pink" transform="scale(2,1)"></rect>
</svg>
```

# Déformation

- **skewX(angle)**
  - **angle**: valeur en degré de la rotation suivant l'axe des abscisses.
- **skewY(angle)**
  - **angle**: valeur en degré de la rotation suivant l'axe des ordonnées.

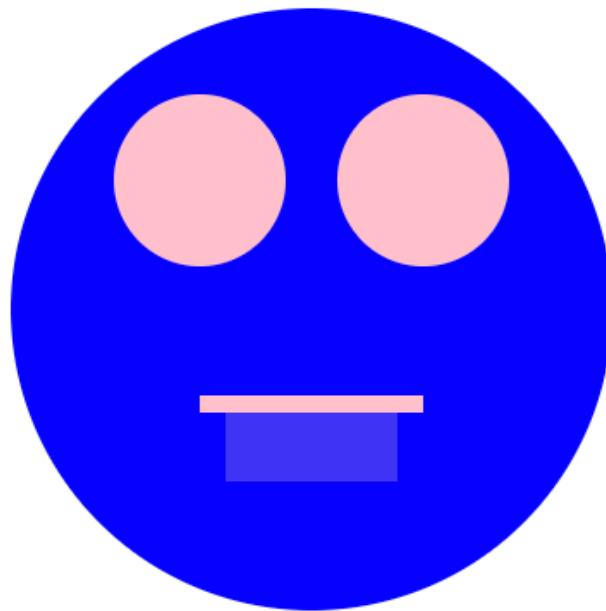


```
<svg>
  <text x="0" y="20" transform="skewX(45)">Texte skewX(45)</text>
  <text x="0" y="20" transform="skewY(45)">Texte skewY(45)</text>
</svg>
```

# Démo

- Quelques animations utiles pour comprendre les transformations :
  - <http://svground.fr/transformations.php>

# TP 2 – Premier graphique SVG



Bonjour !

# Transitions D3

# Transitions D3

- Les transitions D3 permettent d'animer des modifications du DOM effectuées avec D3.
- Depuis la version 4, les fonctionnalités sont regroupées au sein du projet d3-transition (<https://github.com/d3/d3-transition>).
- Appliquer une transition revient à appeler la fonction **transition()** sur une sélection.

```
d3.select("circle")
      .transition()
```

# Paramétrer le temps

```
d3.select("circle")
    .transition()
    .delay(1000)
    .style("opacity", 1);
```

```
d3.selectAll("circle")
    .transition()
    .duration(2000)
    .attr("cy", 200)
    .ease("elastic"); //v3
    .ease(d3.easeLinear); //v4
```

## **transition()**

Appliquer une transition sur des modifications d'éléments

## **delay()**

Délai avant le démarrage de la transition

## **duration()**

Durée de la transition

## **ease()**

Fonction de transformation appliquée au temps à appliquer à la transition. Liste de paramètres : « d3-ease »

(<https://github.com/d3/d3-ease>).

# d3.transition()

```
var t = d3.transition()  
    .delay(1000)  
    .duration(2000)  
    .ease(d3.easeLinear);  
  
d3.selectAll("circle")  
    .transition(t)  
    .attr("cy", 200)
```

## d3.transition()

Créer une transition et l'utiliser plus tard.

# on()

```
d3.selectAll("circle")
  .transition()
  .duration(2000)
  .attr("cy", 200)
  .on("end", afterTransition);
```

```
function afterTransition(){
  ....
}
```

## on()

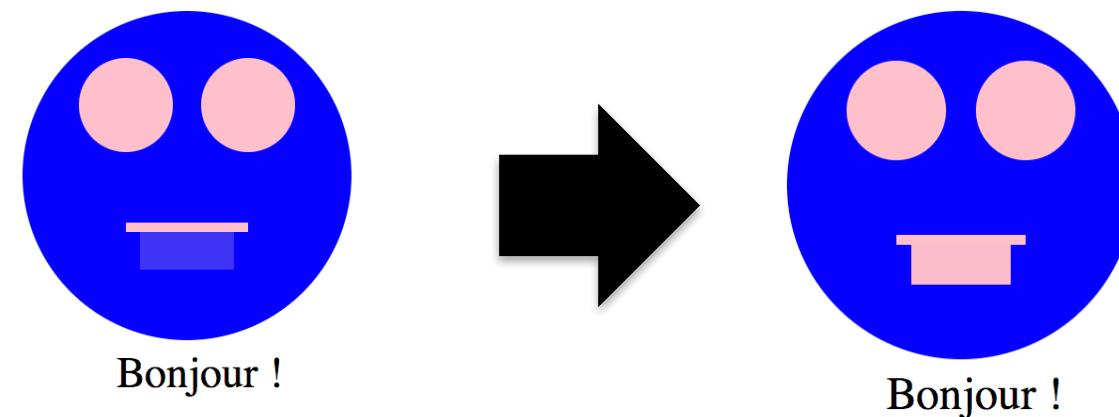
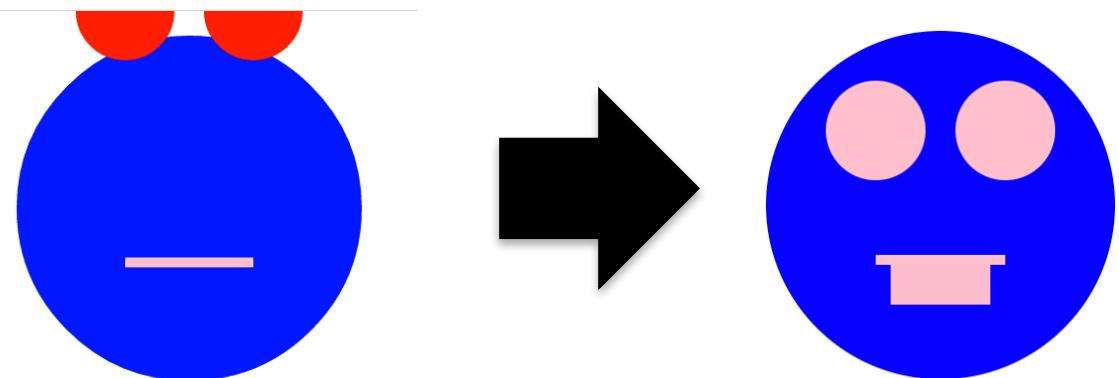
Ajoute un écouteur sur un événement de transition. Les événements existants : ***start, end, interrupt.***

## v3 vs v4

La fonction `on()` en v4 remplace la fonction `each()` existant dans les versions précédentes.

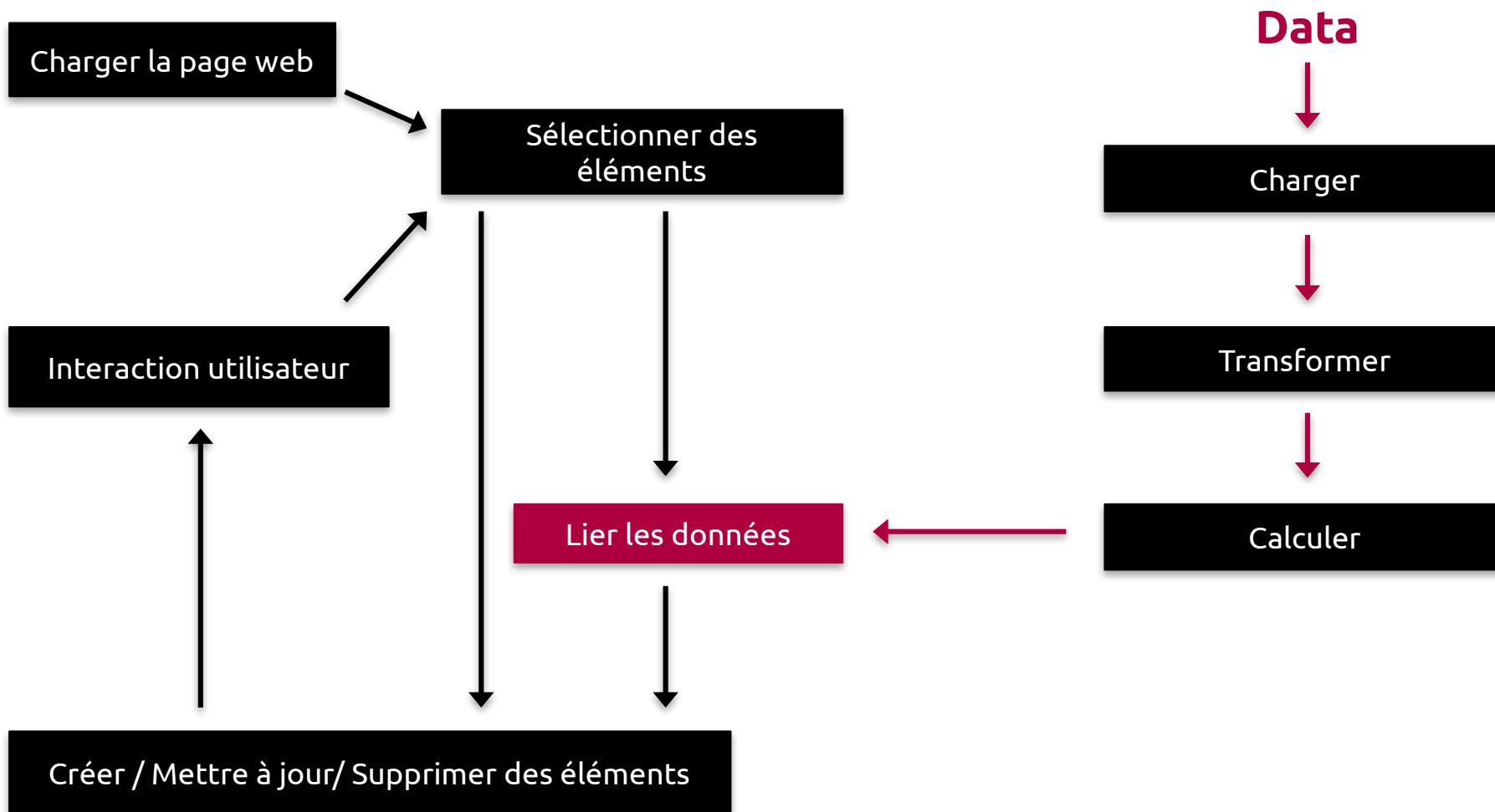
La méthode `each()` v4 permet d'appliquer un traitement à chaque itération de données.

# TP 3 - Transition



# Gestion des données

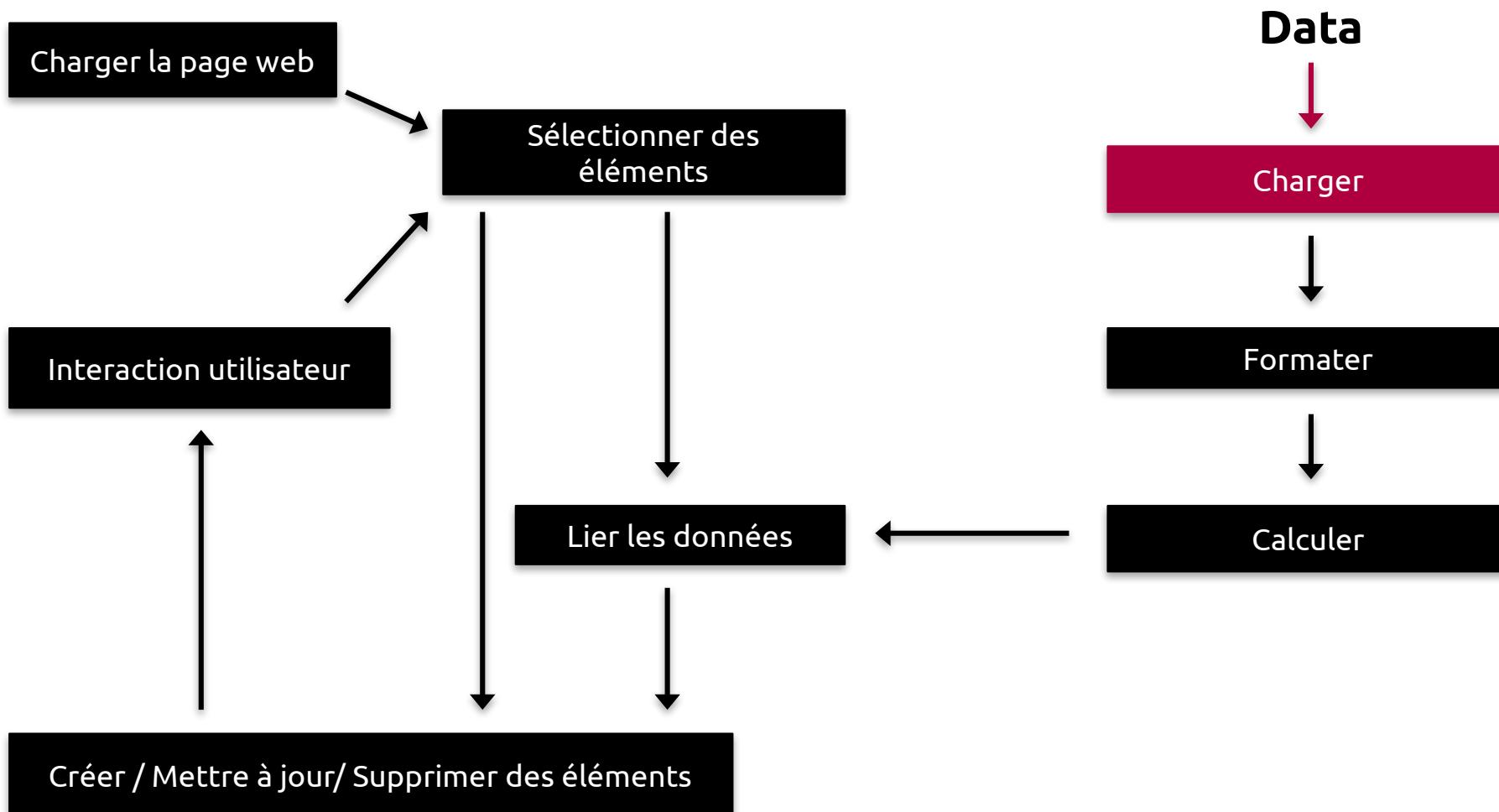
# Travailler avec les données



# Classification des données

- Données tabulaire (tabular data)
- Données imbriquées (Nested data)
- Données en réseau (Network data)
- Données géographiques (Geographic data)
- Données brutes (Raw data)
- Données objets (Objects)

# Charger



# Charger des données

```
d3.csv("villes.csv", function(error,data) { console.log(error,data);});
```

```
d3.tsv("villes.tsv", function(error,data) { console.log(error,data);});
```

```
var myDsv = d3.dsv("|", "text/plain");
myDsv("villes.psv", function(error,data) { console.log(error,data);});
```

```
d3.text("villes.txt", function(data) { console.log(data);}); // error optionnel
```

```
d3.json("villes.json", function(data) { console.log(data);});
```

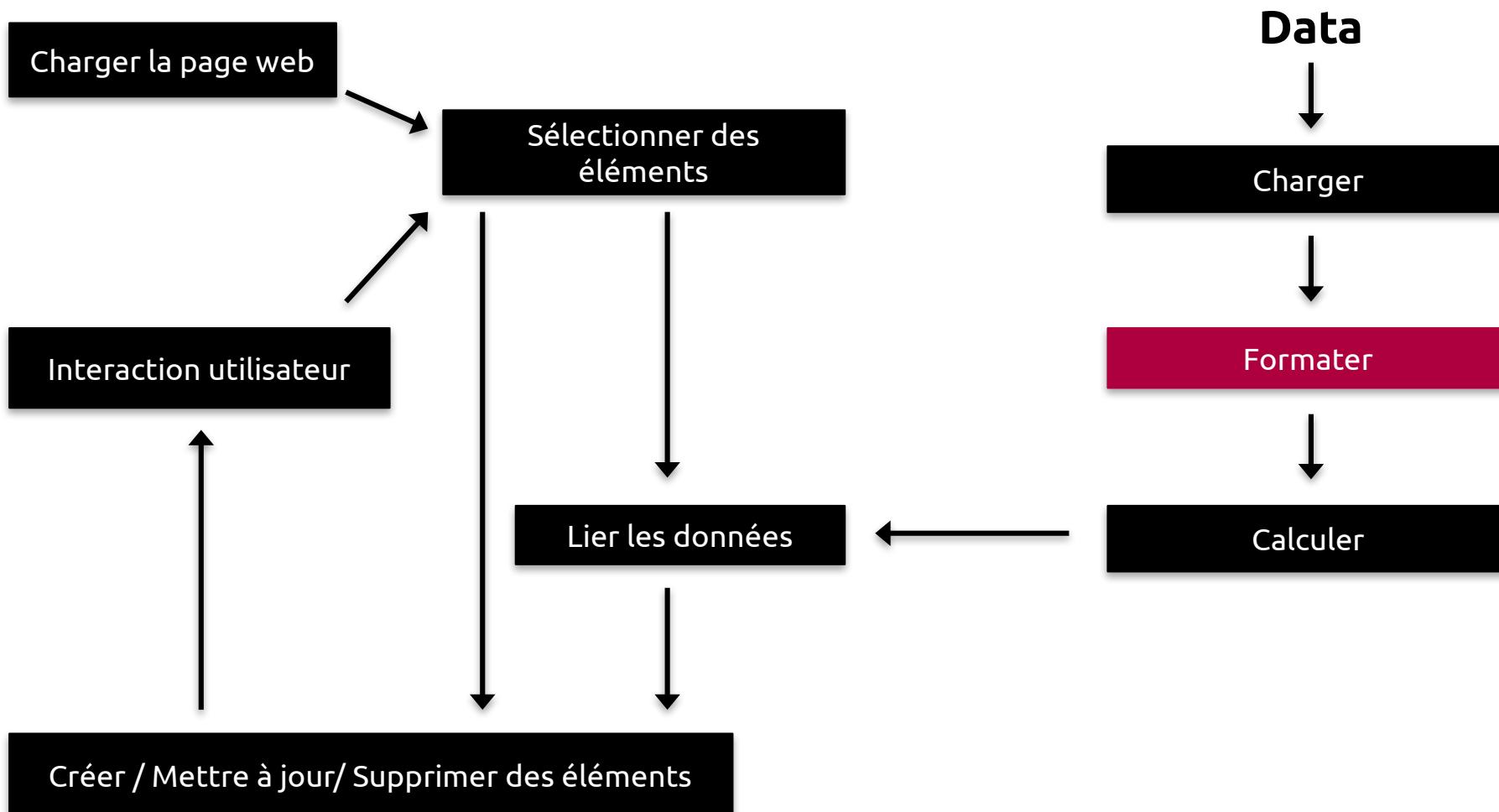
```
d3.xml("villes.xml", function(data) { console.log(data);});
```

```
d3.html("villes.html", function(data) { console.log(data);});
```

v4

Project d3-request :  
<https://github.com/d3/d3-request/>

# Formater



# Formater

- L'objectif de l'étape de transformation est de modifier le format des données pour qu'elles soient en phase avec les éléments graphiques à manipuler.
- Quelques formats usuels :
  - quantitatif (nombre d'habitants d'une ville)
  - catégorisé (nationalité, genre)
  - topologique (arbre généalogique)
  - géométrique (carte des régions d'un pays)
  - temporelle (historique d'une ville)
  - brut (images, textes, ...)

# Formater avec Javascript (1)

```
parseInt("12");
```

```
parseFloat("3.14");
```

```
Date.parse("Sun, 26 Mar 2015 14:00:00 GMT");
```

```
var text = "blue,red,yellow";
var blue = text.split(",")[0];
```

```
JSON.parse("{ 'name' : 'Nancy'}");
```

# Formater avec Javascript (2)

```
var nombres = [1, 3, 43, 4, 121, 9, 314, 31];  
  
var multiplierPar2 = function(element) {  
    return element*2;  
};  
  
var additionner = function(previous, current) {  
    return previous + current;  
};  
  
nombres.map(multiplierPar2).reduce(additionner);
```

map  
forEach  
filter

...

IE9+

# Scale, Domain, Range

## scaleXXX

Définit la mécanique de transformation

## Domain

Définit la plage des données en entrée

## Range

Définit la plage des données en sortie

```
var ramp = d3.scaleLinear().domain([500000, 13000000]).range([0, 500]);  
  
ramp(100000); // = 20  
ramp(900000); // = 340  
  
ramp.invert(313); // = 8325000
```

# Echelles (Scale)

- Une échelle transforme les valeurs des données en entrée en valeurs utilisables dans un graphique.
- Trois catégories d'échelle dans D3 :
  - Echelle quantitative
  - Echelle ordinaire
  - Echelle temporelle

# Echelle quantitative

```
d3.scaleLinear(); // (d3.scale.linear() en v3)
```

```
d3.scaleIdentity();
```

```
d3.scaleSqrt();
```

```
d3.scalePow();
```

```
d3.scaleLog();
```

```
d3.scaleQuantize();
```

```
d3.scale.Quantile();
```

```
d3.scaleThreshold();
```

**v3 vs v4**

*En v3,  
d3.scale.xxxx().*

*d3.scale.linear();  
d3.scale.identity();  
d3.scale.sqrt();  
...*

**v4**

Project d3-scale :  
[https://github.com/  
d3/d3-scale/](https://github.com/d3/d3-scale/)

# Echelle ordinaire

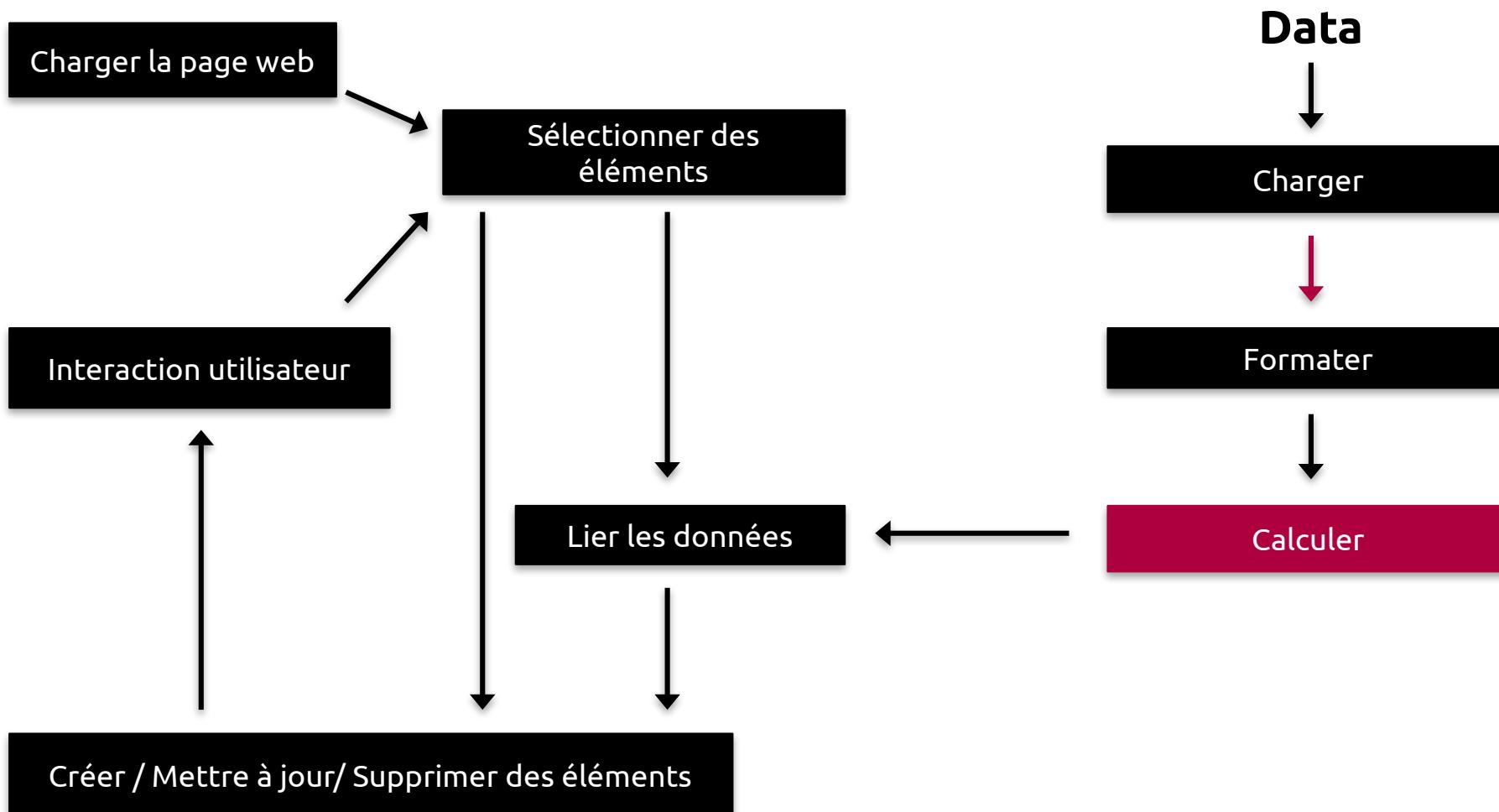
```
var a = d3.scaleOrdinal()  
    .domain(["A", "B", "C", "D"])  
    .range([0, 10, 20, 30]);  
  
a("B"); // = 10
```

# Echelle temporelle

- Une extension de `d3.scaleLinear` qui utilise des dates.

```
var timeRamp = d3.scaleTime()  
    .domain([new Date('2015-01-01T12:00:00.000Z'),  
            new Date('2015-12-31T00:00:00.000Z')])  
    .range([0, 100]);  
  
timeRamp(new Date('2015-01-01T12:00:00.000Z')) // = 0  
timeRamp(new Date('2015-12-31T00:00:00.000Z')) // = 100  
timeRamp(new Date('2015-03-26T00:00:00.000Z')) // = 22,97
```

# Calculer



# Calculer

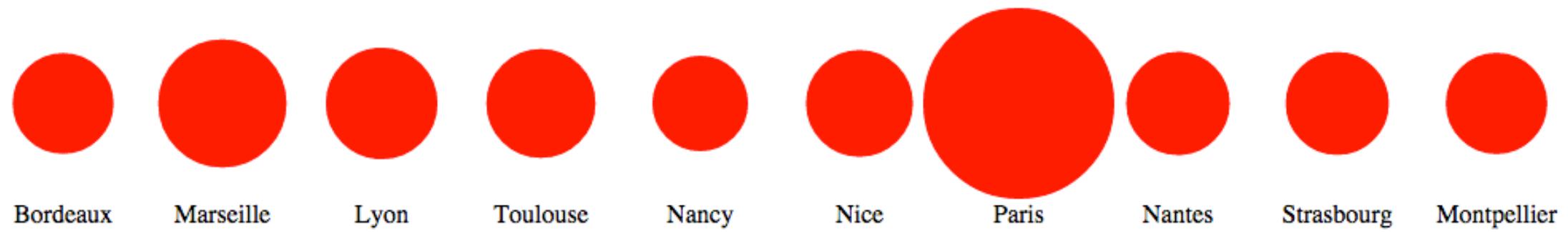
v4

Project d3-array :

[https://github.com/  
d3/d3-array/](https://github.com/d3/d3-array/)

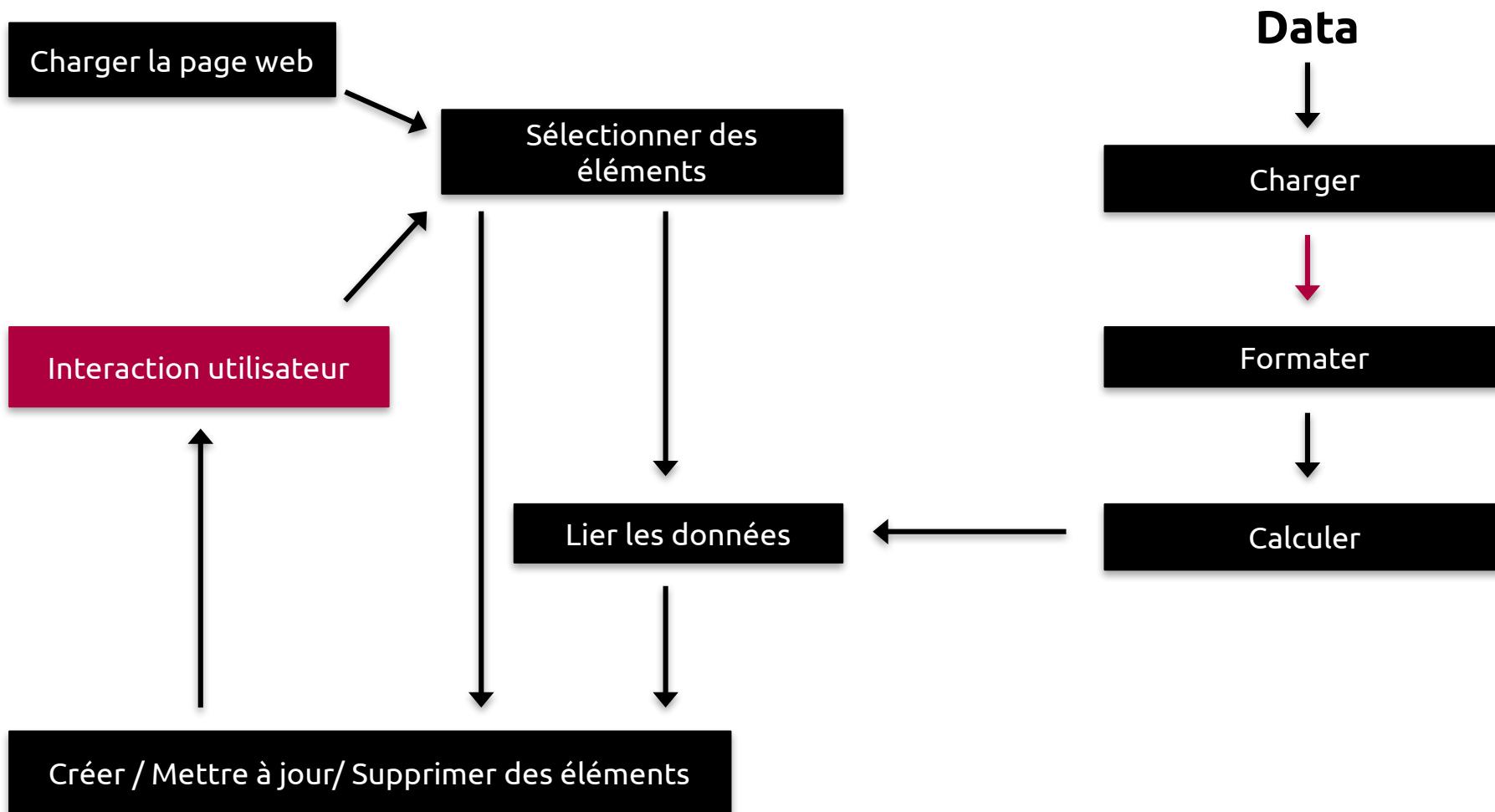
```
d3.json("villes.json", function(data) {  
  
    var min = d3.min(data, function(ville) { return ville.population;});  
    var max = d3.max(data, function(ville) { return ville.population;});  
    var mean = d3.mean(data, function(ville) { return ville.population;});  
    var minMax = d3.extent(data, function(ville) { return ville.population;});  
  
    ... sum, median, quantile, variance, deviation, ...  
});
```

# TP 4 – Manipuler des données



# Les événements

# Evénements

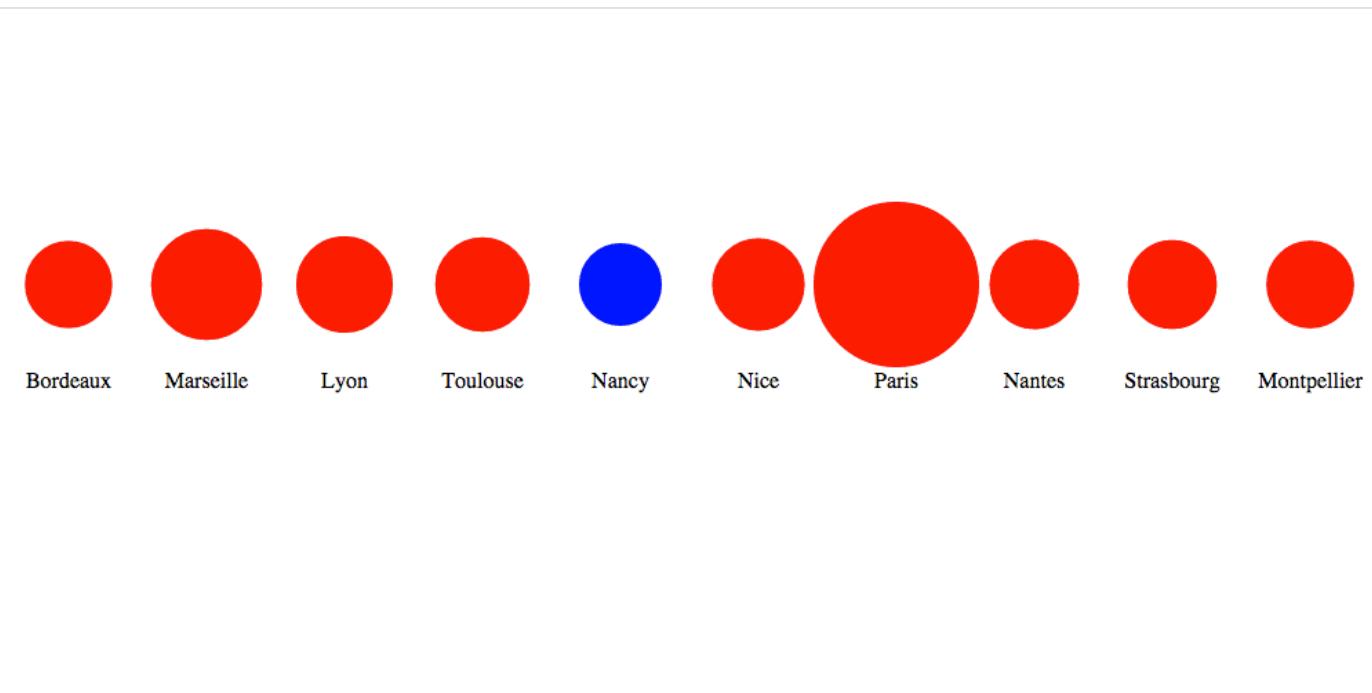


# .on()

```
d3.selectAll("#mybuttons")
  .append("button")
    .on("click", function() { console.log("click");});
```

```
d3.selectAll("#mybuttons")
  .data(datas)
  .enter()
    .append("button")
      .on("click", function(d) { console.log("click " + d);});
```

# TP 5 – Les événements



## Détail

Nom Lyon  
Population 445274  
Superficie 47.9  
Rang 3

[population](#) [superficie](#) [rang](#)