BUT Science des Données 2 - Semestre 4

COMPETENCE 1 : Traiter des données à des fins décisionnelles

Ressource : **Programmation web (VCOD)**

F.GARNIER

PLAN DE LA RESSOURCE

Chapitre 2 : Le framework javascript D3.js

1. Introduction D3,JS – Data-Driven Documents

Site officiel : https://d3js.org/

D3.js est une librairie javascript (ou framework javascript) très utilisée **pour la visualisation de données sur le web**. Elle est complète avec beaucoup d'exemples à disposition et une personnalisation totale possible. Principalement utilisée pour des visualisations au format `SVG`.
`SVG` (Scalable Vector Graphics) est un format d'images vectorielles basé sur le langage XML. Il permet de décrire des formes graphiques. Pour en savoir plus : https://grafikart.fr/tutoriels/svg-scalable-vector-graphics-468

La librairie est malheureusement assez technique. L'idée principale est de lier les **données** au **DOM** (*Document Object Model*) et **d'appliquer des transformations au document qui sont basées sur les données** afin de mettre à jour du contenu à des temps déterminés, créer des cartes interactives, des animations 2D/3D, des menus vidéo défilants, …

D3.js c'est du code javascript ! Ce chapitre va permettre de décrire les éléments de base pour comprendre cette librairie.

2. Invoquer le framework D3.js dans une page web

Deux possibilités :

- Récupérer l'archive de la dernière version du framework sur le site officiel
- Invoquer l'adresse de la version de référence (cas représenté dans l'exemple ci-dessous) :

```
<!DOCTYPE html>
<html>
	<head>
	  <meta charset="UTF-8">
	</head>
	<body>
		<script type="text/javascript"
		    src="https://cdnjs.cloudflare.com/ajax/libs/d3/4.1.1/d3.js">
		</script>
		<script src="./chapitre1.js"> </script>
	</body>
</html>
```

Lien vers D3.js (accès internet doit fonctionner !)

Lien vers le script javascript à écrire travaillant sur le document html

3. Sélectionner des éléments de la page web

- Pour sélectionner un seul élément : `select()`
  - ⇨ *Retourne le premier élément s'il y en a plusieurs*

- Pour sélectionner plusieurs éléments similaires : `selectAll()`

- Sélection identique à celle du CSS :

Sélecteur possible	Quoi ?	Comment dans le html ?
`"balise"`	Objet(s) de type balise html	`<div>`, `<h1>`, `<body>`, ...
`".maclasse"`	Objet(s) de la classe css indiquée	`<balise class='maclasse'>`
`"#identifiant"`	Objet avec cet identifiant (normalement) unique	`<balise id='identifiant'>`

```
/* Exemple : Sélection de la balise body */
var corps = d3.select("body");
```

4. Récupérer des informations et Modifier des éléments de la page web

Plusieurs fonctions pour :

- avoir des informations sur la sélection :
  - `size()` : taille d'une sélection
  - `empty()` : sélection vide ou non
- modifier les éléments sélectionnés :
  - `style()` : appliquer des règles `CSS`
  - `html()` : modifier le contenu d'une balise

Ex : mettre le texte en rouge pour tout le corps de la page

```
var corps = d3.select("body");
corps.style("color","red");
```

5. Ajouter des éléments à la page web

Deux fonctions :

- Ajouter un élément `HTML` (balise) à la fin de l'élément : `append()`
- Insérer au début : `insert()`

Attention : Stocker le résultat si on veut pouvoir modifier les éléments ajoutés.

Exemple :

```
selection = d3.select("selecteur");
nouveau = selection.append("balise");
selection.insert("balise");
```

Exemple final n°1 :

```
// Sélection de la balise body
var corps = d3.select("body");

// Ajout de deux balises div au corps
var div1 = corps.append("div");
var div2 = corps.append("div");

// Ajout de données (contenu) aux deux balises
div1.html("Je met du texte ici.");
div2.html("Nombre de div : " + d3.selectAll("div").size());

// Modification de la couleur de la police
corps.style("color", "red");
div2.style("color", "steelblue");
```

Résultat :

Je met du texte ici.
Nombre de div : 2

6. Ajout de données au DOM

Permet d'ajouter des **données** à nos éléments du DOM (balises, …). Pour cela, il existe la fonction `data()` sur une sélection avec en paramètre un tableau de données à relier au DOM.

Ex : Affecte chaque élément du tableau à chaque élément renvoyé par le sélecteur précédent

```
var selection = d3.selectAll("selecteur");
selection.data(tableau);
```

S'il y a une différence entre la taille de la sélection et la taille du tableau passé en paramètre :

- `enter()` : pour gérer les éléments du tableau en plus
- `exit()` : pour gérer les éléments de la sélection en plus

6.1 Propriété dynamique : Sur chaque sélection, on peut appliquer des modifications de style ou de contenu (voire autre), en fonction des données qui sont liées au DOM. On passe par l'utilisation d'une fonction **anonyme en paramètre**, dont les paramètres peuvent être, dans cet ordre :

- l'élément du tableau
- l'indice de l'élément dans le tableau

Il est possible de n'utiliser que la valeur, voire aucun paramètre si nécessaire :

```
var selection = d3.selectAll("selecteur");
selection.data(tableau);
selection.html(function(d, i) {
    return "position = " + i + ", valeur = " + d;
})
```

Exemple final 2 :

```
// Fichier html
<div>Pomme</div>
<div>Tomate</div>
<div>Fleur</div>
<script …>
```

```
// Balise <script> ou fichier javascript associé
var div = d3.selectAll("div");
div.data(["green", "red", "blue"])
div.style("color", function (d) { return d; });
```

Résultat :

Pomme
Tomate
Fleur

Dans cet exemple, on affecte les données du tableau (qui contient des couleurs) à chaque `div` du `body`. Et on modifie le style `CSS` (couleur) avec les valeurs du tableau.

Exemple final 3 : **Le tableau de données est plus grand que le nombre de div !**

```
// Même fichier html, Fichier javascript : div : liste des div présents dans le HTML.
// select("body") pour s'assurer que les div soient ajoutées dans le body et non après
var div = d3.select("body").selectAll("div");

// div_data : div avec les données associées
var div_data = div.data(["green", "red", "blue", "orange", "purple"]);

// affectation de la couleur à chaque div
div.style("color", function (d) { return d; });

// div_enter : données en trop
var div_enter = div_data.enter();
// div_nv : nouvelles div ajoutée à partir des données en trop
var div_nv = div_enter.append("div");

// définition du contenu des nouvelles div
div_nv.html("div ajoutée");

// affectation de la couleur à chaque nouvelle div
div_nv.style("color", function (d) { return d; });
```

Résultat :

Pomme
Tomate
Fleur
div ajoutée
div ajoutée

Exemple final 4 : **Le tableau de données est plus petit que le nombre de div (la sélection) !**

```
var div = d3.select("body").selectAll("div");

var div_data = div.data(["green", "red"]);

div.style("color", function (d) { return d; });

// div_exit : div en trop
var div_exit = div_data.exit();

// suppression de ces div en trop
div_exit.remove();
```

Résultat :

Pomme
Tomate

6.2 Chaînage des fonctions

Il faut absolument comprendre le principe généralement appliqué en JS orienté objet : « **Toute fonction d'un objet renvoie cet objet ».** Ceci est vrai sauf si la fonction a pour but de renvoyer un résultat spécifique. Et cela ne concerne donc que les procédures (qui sont aussi des fonctions en JS). Le corollaire de ce principe est intéressant : « Il est possible d'enchaîner un grand nombre de fonctions directement ».

Dans l'exemple ci-dessous, on utilise ce principe pour créer autant de `div` qu'il y a de couleurs dans un tableau (donc recours à `enter()`), en indiquant le contenu `HTML` de celles-ci, et en leur appliquant un style `CSS` spécifique :

```
var couleur = ["green", "red", "blue", "orange", "purple"];
d3.select("body").selectAll("div")
  .data(couleur)
  .enter().append("div")
  .html(function(d,i) { return "Div n°" + (i + 1) + " : " + d; })
  .style("color", function(d, i) { return d; });
```

Résultat :

Div n°1 : green
Div n°2 : red
Div n°3 : blue
Div n°4 : orange
Div n°5 : purple

Remarque : Le fichier HTML initial ne contient plus aucune div !

7. Lecture de données externes

Il existe plusieurs fonctions dans la librairie `D3` pour charger des données de tout type (`JSON`, `CSV`, `TSV`, `XML`, …). Les fonctions pour le faire sont toutes de type `d3.xxx()`
Exemple : `d3.json()` pour traiter un fichier `JSON`

L'exemple ci-dessous (**avec un fichier HTML vierge de données)** charge les données contenues dans le fichier `mpg.csv` disponible sur mon `GitHub`, qui recense un ensemble de voitures avec plusieurs caractéristiques :

```
d3.csv("https://francoisgarnier.github.io/mpg.csv",
       function(err, don) {
  console.log(don);
  if (err)
    d3.select("body").html("Erreur lecture csv : " + err)
  else {
    d3.select("body").selectAll("div")
      .data(don)
      .enter()
      .append("div")
      .html(function(e, i) {
        var r = "<strong>" + e.manufacturer + "</strong>, " +
            e.model +
            " (<em>" + e.year + "</em>)";
        return r;
     });
   }
})   // manufacturer, model et year sont des colonnes du fichier csv
```

Résultat :

audi, a4 (*1999*)
audi, a4 (*1999*)
audi, a4 (*2008*)
audi, a4 (*2008*)
audi, a4 (*1999*)
audi, a4 (*1999*)
audi, a4 (*2008*)
audi, a4 quattro (*1999*)
audi, a4 quattro (*1999*)
audi, a4 quattro (*2008*)
audi, a4 quattro (*2008*)
audi, a4 quattro (*1999*)
audi, a4 quattro (*1999*)

8. Ajout d'interactivité : gestion d'évènements

Il est possible d'ajouter des gestions d'événements sur les objets créés via la fonction `on()`. Celle-ci prend en premier paramètre l'événement (*par ex `mouseover` pour gérer le positionnement de la souris sur l'objet*) et en deuxième paramètre la fonction anonyme à appliquer quand l'évènement survient. Dans cette fonction, **il n'y aucun paramètre possible** mais on accède à l'élément objet via `this`. De plus, si nous avons pris le soin d'ajouter des propriétés à cet objet (via la fonction `property("nom", valeur)`), nous pouvons y accéder via `this.nom`.

Dans l'exemple ci-dessous, au survol de la souris sur chaque `div` (ainsi que sa sortie), nous ajoutons une propriété `couleur` à chacune, qui prendra la valeur de la couleur dans le tableau. Ensuite, on indique que lorsque la souris passe sur la `div` (`on("mouseover", ...)`), on change la couleur de la police par celle spécifique à la `div`.

Si nous ne gérons pas la sortie de la souris, la couleur ne sera jamais modifiée. Nous gérons donc ce cas (via `on("mouseout, ...)`) en indiquant que la couleur redevient noire :

```
d3.selectAll("div")
  .data(["green", "red", "blue"])
  .property("couleur", function(d) { return d;})
  .on("mouseover", function () {
    d3.select(this).style("color", this.couleur);
  })
  .on("mouseout", function () {
    d3.select(this).style("color", "black");
  });
```

Position de la souris :

Objet `d3.event` = informations de la souris (position principalement) :

- `clientX` et `clientY` : position relative à la partie visible du navigateur
- `screenX` et `screenY` : position relative au moniteur
- `pageX` et `pageY` : position relative au document `HTML`
- `offsetX` et `offsetY` : position relative à l'objet sur lequel la souris est positionnée

Implémentation variable entres les navigateurs

Voici un petit exemple de ce qu'on peut récupérer comme informations lorsque l'on survole la forme SVG créée :

```
// Fichier html
<svg width=200 height=100></svg>
<div id = "infos">
  <div id = "client"></div>
  <div id = "screen"></div>
  <div id = "page"></div>
  <div id = "offset"></div>
</div>
```

```
// Fichier css
svg {
  border: solid 1px black;
  margin-left: 50px;
  margin-top: 50px;
}

#infos {
  width: 200px;
  float: right;
}

#client:before {
  content: "client : "
}
#screen:before {
  content: "screen : "
}
#page:before {
  content: "page : "
}
#offset:before {
  content: "offset : "
}
```

```
// Fichier js
d3.select("svg")
  .on("mousemove", function () {
    m = d3.event;
    d3.select("#client").html(m.clientX + "-" +
m.clientY);
    d3.select("#screen").html(m.screenX + "-" +
m.screenY);
    d3.select("#page").html(m.pageX + "-" +
m.pageY);
    d3.select("#offset").html(m.offsetX + "-" +
m.offsetY);
  })
  .on("mouseout", function () {

d3.select("#infos").selectAll("div").html("");
  });
```

Résultat :

9. Graphiques SVG

La librairie d3 permet de créer des graphiques au format `SVG` (*Scalable Vector Graphics*) et c'est régulièrement dans ce cadre qu'on l'utilise.
Ces graphiques sont définis dans un langage de type `XML` (et donc similaire à `HTML`). C'est un langage de définition basé sur des primitives de dessin (rectangle, cercle, ligne, texte, …), qui permet de produire tout type de graphique. L'un des gros avantages est qu'ils sont *zoomables* sans perte de définition. Vous pouvez trouver dans les liens qui suivent un certain nombre d'informations sur ces graphiques :

- Recommandation W3C traduite
- Section SVG sur Mozilla

Dans l'exemple ci-dessous, nous créons :

- un graphique de largeur 200 pixels et de hauteur 100 pixels
- une transformation (via la balise g ajoutée). Celle-ci est une translation de 10 pixels en *x* et de 10 pixels en *y*. C'est le résultat de la translation qui est renvoyé, ce qui veut dire que tout ce que l'on ajoutera au sein de la balise g intégrera donc cette première translation.
- un rectangle dont le point haut gauche est situé en *(0,0)*. Notez donc que le point origine est donc situé **en haut à gauche** sur un écran. Ce rectangle est un carré de 50 pixels, rempli en rouge.
- un texte :

Fichier html vide :

```
// Fichier css
svg {
  border: solid 1px black;
}
```

```
// Fichier js
var graph = d3.select("body").append("svg")
    .attr("width", 200)
    .attr("height", 100)
    .append("g")
    .attr("transform", "translate(10, 10)");

graph.append("rect")
  .attr("x", 0).attr("y", 0)
  .attr("width", 50).attr("height", 50)
  .style("fill", "red");

graph.append("text")
  .attr("x", 75).attr("y", 50)
  .text("Voici un graphique")
  .style("font-size", ".75em");
```

Résultat :

10. Echelles

Dans un graphique, nous devons faire un passage d'échelle entre les données et la zone graphique. Par exemple, si l'on doit afficher des valeurs entre -1000 et 1000 sur l'axe *x*, il nous faut une fonction pour les transformer dans l'intervalle *[0,largeur]* (où *largeur* représente la largeur du graphique `SVG` produit).
Pour réaliser cela, les fonctions dans `d3` ont toutes comme nom `d3.scaleXxx()`, où `Xxx` est à remplacer par le type de changement d'échelle que l'on souhaite. Il faut noter que ces fonctions renvoient elle-même une fonction de changement d'échelle. Il faut de plus déterminer deux éléments importants :

- Le **domaine** (ou *domain*) : la plage des données d'origine
- L'**étendu** (ou *range*) : la plage de ce qu'on doit obtenir au final

Quantitatif

L'exemple proposé ci-dessus est typiquement un problème de changement d'échelle **linéaire**. Il existe pour cela la fonction `d3.scaleLinear()` :

```
var echelle = d3.scaleLinear().domain([-1000, 1000]).range([0, 100]);
console.log(echelle(-1000)) // renvoie 0
console.log(echelle(0))     // renvoie 50
console.log(echelle(1000))  // renvoie 100
```

L'intérêt de ces échelles réside aussi dans la possibilité de passer de valeurs numériques à des couleurs par exemple. On doit juste définir dans l'étendu les couleurs de début et de fin (et éventuellement certaines intermédiaires) :

```
var echelle = d3.scaleLinear().domain([-1000, 1000]).range(["red", "green"]);
console.log(echelle(-1000)) // renvoie "#ff0000"
console.log(echelle(0))     // renvoie "#804000"
console.log(echelle(1000))  // renvoie "#008000"
```

Qualitatif

Un autre changement d'échelle classique est le passage d'un ensemble de modalités à une plage de valeurs. Pour cela, on utilise la fonction `d3.scaleBand`(). Ici, nous définissons l'étendu par bandes ("A" sera sur la bande ainsi *[0, 33.33…]*).

```
var echelle = d3.scaleBand()
        .domain(["A", "B", "Z"])
        .range([0, 90]);
console.log(echelle("A")) // renvoie 0
console.log(echelle("B")) // renvoie 30
console.log(echelle("Z")) // renvoie 60
console.log(echelle.bandwidth()) // renvoie 30
```

On peut aussi affecter une couleur à chaque modalité, toujours en définissant des couleurs dans l'étendu. Il existe de plus des fonctions spécifiques pour cela dans `d3`, comme `d3.scaleOrdinal()`, dans lesquelles il n'y a pas d'étendu à définir :

```
var echelle = d3.scaleOrdinal(d3["schemeSet1"])
    .domain(["A", "B", "Z"]);
console.log(echelle("A")) // renvoie "#1f77b4"
console.log(echelle("B")) // renvoie "#ff7f0e"
console.log(echelle("Z")) // renvoie "#2ca02c"
```

Il existe un certain nombre de couleurs prédéfinies : site **`Color Brewer`**