



INF8808 - Visualisation de données

TP3 : Scatter Plot

Département de génie informatique et génie logiciel
Polytechnique Montréal
Hiver 2017



1. Présentation du TP

a. Objectifs

Le but de ce TP est de réaliser un scatter plot depuis plusieurs bases de données CSV externes. L'utilisateur pourra changer de données à sa guise et une animation permettra de modifier le graphe en temps réel. Il est recommandé d'avoir lu les chapitres 8 et 9 du livre de Scott Murray avant de commencer le TP. Les données utilisées proviennent du portail de données de la banque mondiale : <http://donnees.banquemondiale.org/>.

b. Scatter Plot

Un scatter plot (ou nuage de points) est un type de graphe permettant de visualiser des données possédant plusieurs paramètres. Chaque donnée est représentée par un point. Ils sont par exemple utilisés en physiques pour illustrer des mesures physiques que l'on souhaiterait faire corrélérer à un modèle théorique. Le but de ce type de graphe est soit de trouver un comportement global à une série de points soit de mettre en évidence des clusters en fonction des divers paramètres considérés.

c. Détail du TP

Le travail pratique est constitué d'un fichier HTML et de plusieurs fichiers JavaScript contenant le code et de deux fichiers CSV contenant les données à afficher. Chaque fichier CSV contient des informations sur la zone géographique, le nombre d'habitant, l'espérance de vie moyenne et le revenu moyen de 156 pays pour une année (2000 et 2014). On se propose donc de visualiser ces données et d'observer l'évolution dans leur pays à travers un scatter plot où on indiquera :

- Le revenu moyen du pays en ordonnée
- L'espérance de vie moyenne du pays en abscisse
- La taille de la population du pays avec la taille du rayon du cercle
- La zone géographique du pays avec la couleur du cercle

Dans une première partie, on va prétraiter les données pour les rendre utilisables par D3. Dans une deuxième partie on va dessiner un scatter plot pour l'année 2000. Dans une troisième partie le graphe sera rendu dynamique pour afficher au choix les données de l'année 2000 ou 2014 avec une transition afin de suivre l'évolution. Une quatrième partie rajoutera une tooltip classique et enfin une cinquième partie permettra à l'utilisateur de mettre en évidence des pays de différentes manières.

Les parties 3, 4 et 5 sont indépendantes mais les parties 1 et 2 doivent être réalisées avant et dans l'ordre.



2. Travail à réaliser

a. Prétraitement des données – Données 2000

On souhaite afficher les zones géographiques des pays de différentes couleurs pour visualiser une possible corrélation entre le niveau des pays et leur localisation. Il faut donc dans un premier temps préciser le domaine de notre échelle pour les couleurs pour relier une couleur à une zone géographique. De plus il est nécessaire de convertir les données CSV en objet JavaScript et de les trier dans l'ordre alphabétique. Dans cette partie il faut :

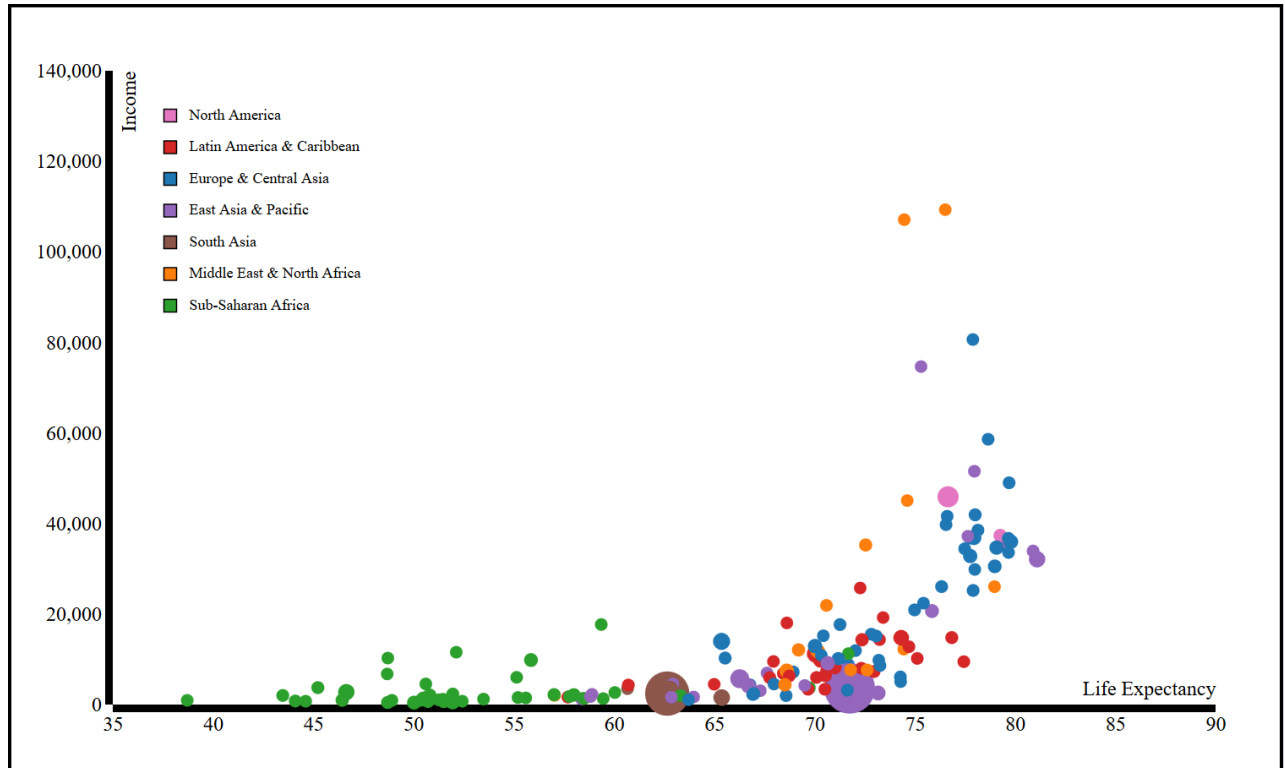
- Obtenir le nom des différentes régions géographiques du CSV (***getRegions***)
- Définir les échelles des axes, du rayon et des couleurs (***setDomains***)
- Transformer les données brutes du CSV (***countriesArray*** et ***sortPays***)

Remarque :

Le scatterPlot va servir à suivre l'évolution des pays pour 2 années. Aussi les échelles que l'on utilise doivent englober l'ensemble des valeurs pour ces deux années (que l'on pourrait facilement remplacer par 15 années ou 100 années). Pour simplifier le problème, on va coder en dur les valeurs à utiliser pour prendre en compte l'ensemble des valeurs des pays entre les deux années. De plus on triera les pays par ordre alphabétique pour qu'un cercle corresponde toujours à un seul pays.

b. Création du scatter plot – Données 2000

La création du scatter plot est très simplifiée suite au prétraitement des données. Il ne reste qu'à construire les axes et à afficher les points sur le graphe en utilisant les données précédentes via d3.js.



Scatter Plot pour l'année 2000

Dans cette partie il faut :

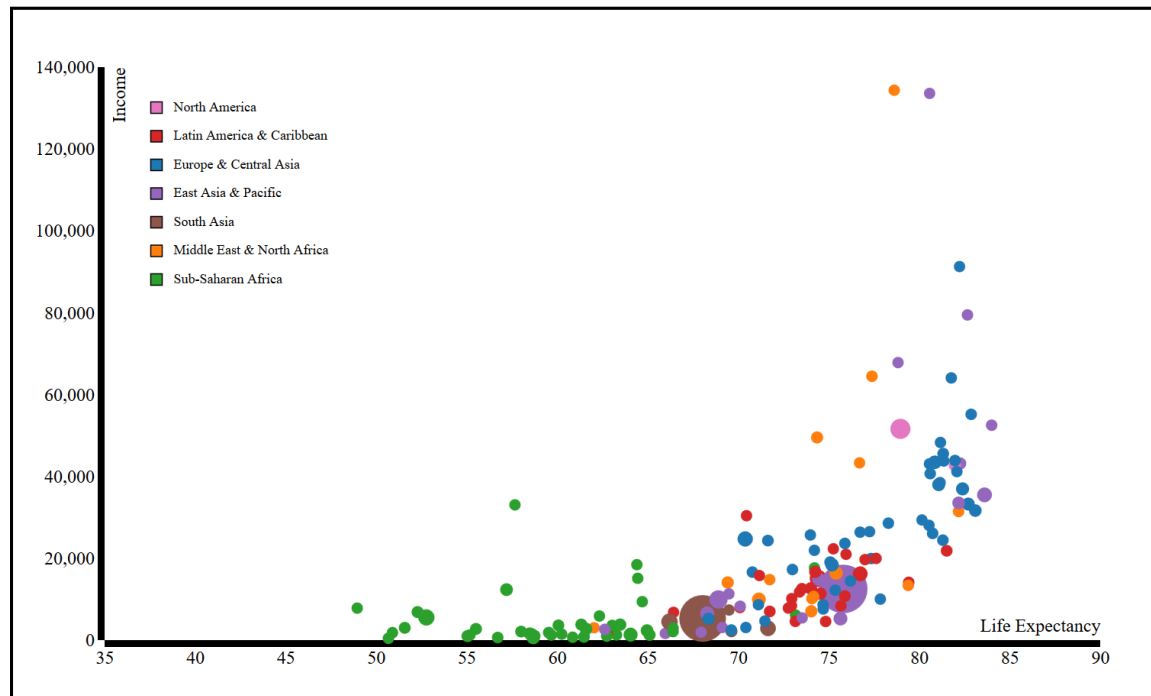
- Afficher les axes (**axes**)
- Afficher le scatter plot pour l'année 2000 (**scatterPlot**)
- Afficher la légende (**legend**)

Remarque :

Il sera utile pour la suite du TP de donner un identifiant à chaque cercle afin de pouvoir les sélectionner facilement par la suite. Une solution simple est d'utiliser le nom des pays. Pour éviter des problèmes de sélection avec des noms contenant des points, ou des espaces, utilisez la fonction **normalizeName** pour nommer et sélectionner les pays. Cette fonction permet aussi d'ignorer la casse.

c. Transition entre les données 2000 et 2014

En changeant le slider de position, une fonction *transition* est appelée et va, en plus d'afficher l'année sélectionnée, modifier le scatter plot en réalisant une transition pour visualiser les nouvelles données.



scatter plot pour l'année 2014

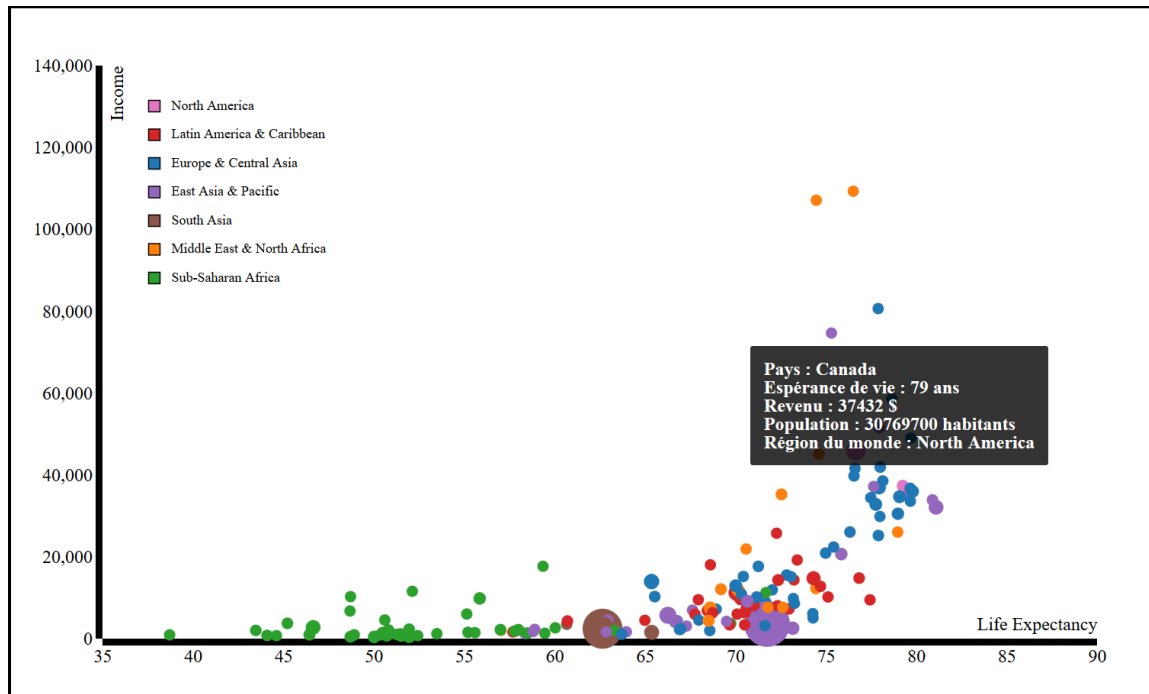
Évidemment, la transition doit se faire dans les deux sens et une double transition 2000->2014->2000 doit amener au même graphe. Dans cette partie il faut :

- Rajouter un EventListener au slider pour réagir au changement d'année
- Charger les données correspondantes à l'année sélectionnée (**loadCSV**)
- Réaliser la transition du scatter plot (**transitionPays**)

Ne pas oublier de mettre à jour le texte indiquant l'année sélectionnée !

d. Tooltip

Une tooltip permet d'afficher les informations des pays en passant la souris sur leur cercle. Cette tooltip doit contenir le nom, la population, l'espérance de vie, le revenu moyen et la zone géographique du pays.



Tool-tip pour la Guinée Equatorial en 2014

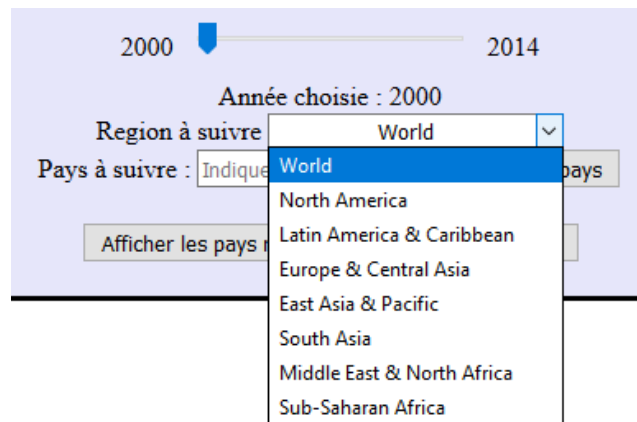
Dans cette partie il faut :

- Réaliser la tooltip répondant aux spécifications demandées

e. Interaction utilisateur

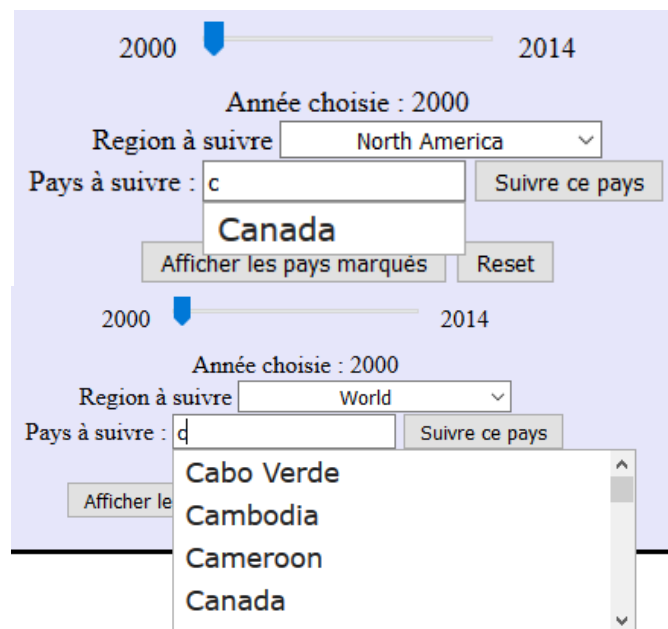
Dans cette partie nous allons implémenter une interface utilisateur afin de rendre le graphe plus dynamique pour l'utilisateur.

On souhaite d'abord ajouter une liste contenant l'ensemble des régions géographiques plus l'ensemble des régions (*World*). Nous utiliserons par la suite cette sélection pour afficher uniquement les pays de la région sélectionnée (fonction ***addListRegions***)



Les différentes régions sélectionnables. World correspond à tous les pays

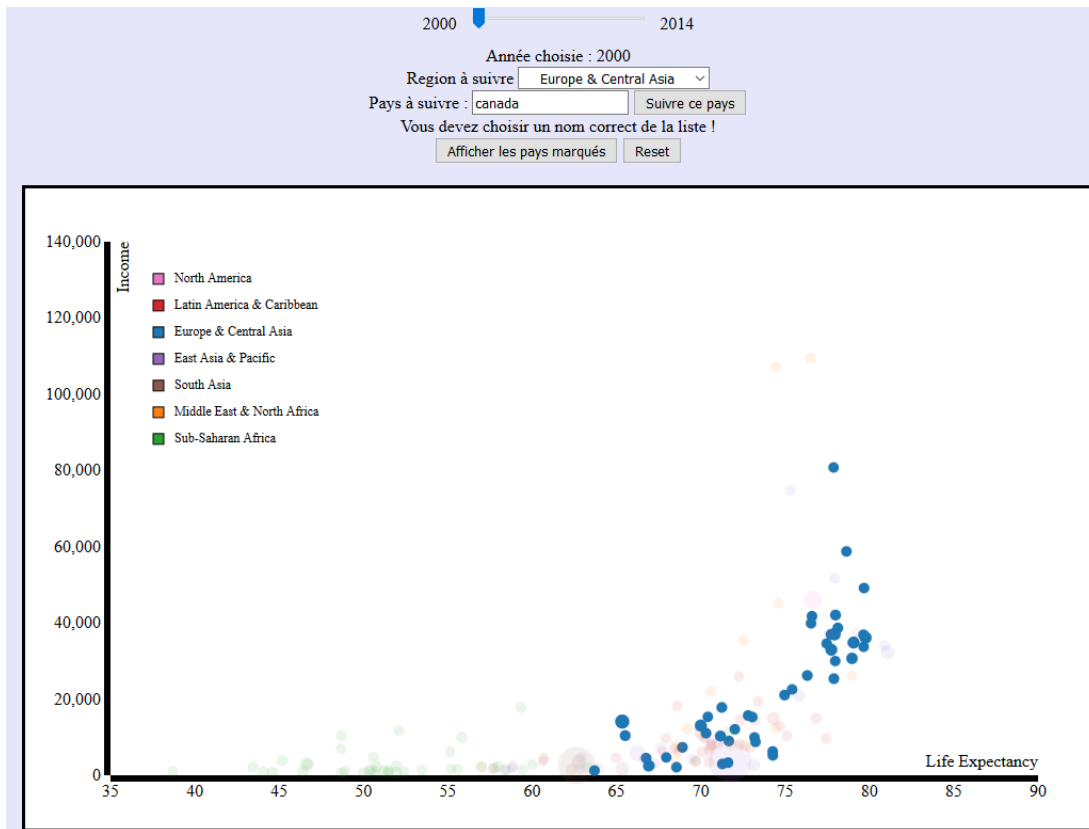
On souhaite aussi rajouter une auto complétion pour permettre à l'utilisateur d'obtenir des résultats préliminaires en fonction de ce qu'il a déjà écrit (fonction ***autoFillerCountries***). Ces résultats doivent contenir l'ensemble des pays **dans la région sélectionnée** (fonction ***countriesInRegions***)



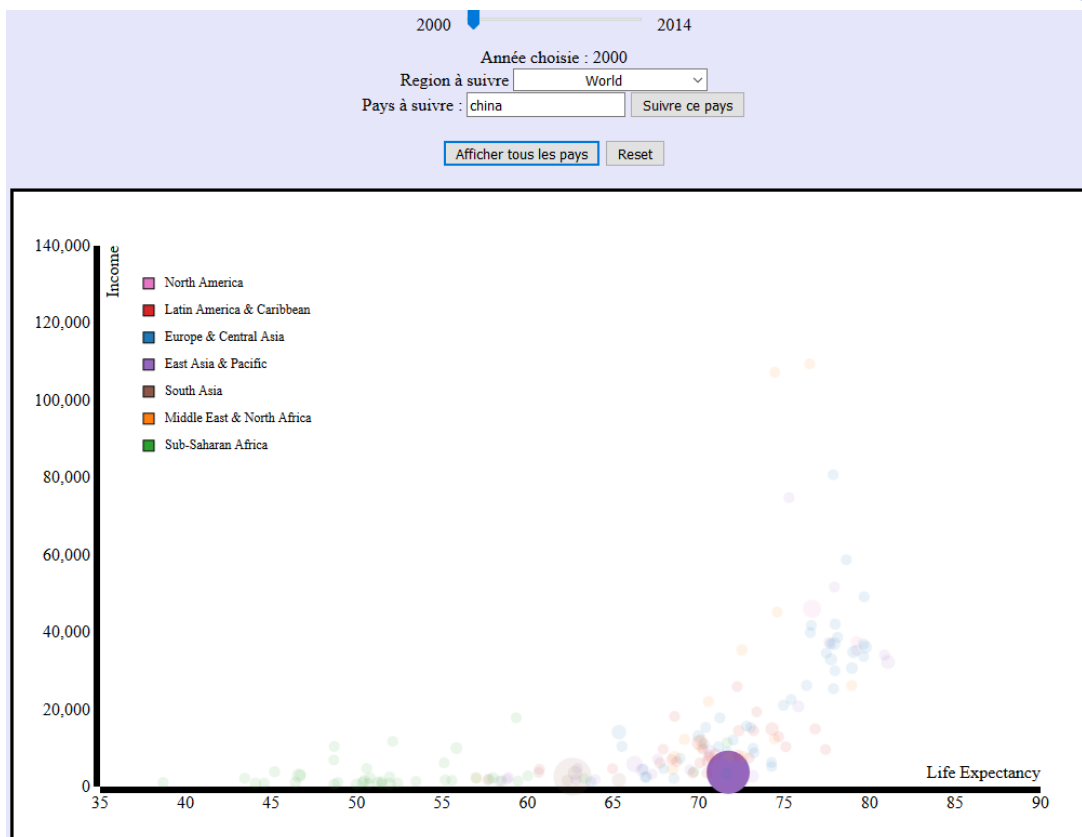
Différents résultats pour la lettre c selon la région à suivre sélectionnée

Désormais, nous pouvons utiliser l'interface précédente pour afficher les données souhaitées par l'utilisateur. Nous allons utiliser différents EventListeners afin de permettre cet affichage personnalisé. Les différentes fonctions sont décrites dans le code de **5-display.js** aussi seules le résultat pour l'utilisateur est précisé ci-dessous (vous pouvez l'interpréter comme un cahier des charges) :

- La sélection des pays par l'écriture ne doit pas être sensible à la casse
- En cas de mauvais nom sélectionné, un texte indiquant que l'utilisateur doit afficher un nom de pays correct s'affiche
- Les pays sélectionnés par l'utilisateur se colorent en noir
- L'espace réservé au texte d'erreur doit toujours être présent pour ne pas altérer verticalement la page.
- En cliquant sur le bouton **Afficher les pays marqués** les pays sélectionnés par l'utilisateur reprennent leur couleur initiale et les pays non sélectionnés deviennent transparents et n'affiche plus leur tooltip. On ne peut plus sélectionner ou désélectionner de nouveaux pays quand on affiche les pays sélectionnés.
- En cliquant sur le bouton **Afficher tous les pays** on revient à la situation initiale (pays opaques, pays sélectionnés en noir)
- Pour simplifier le code, on considère qu'on ne peut mettre en valeur qu'une seule région à la fois (*World* est considérée comme une région) et changer de région annule les sélections de pays de l'utilisateur.
- Les pays ne faisant pas partie de la région sélectionnée et/ou des pays sélectionnés par l'utilisateur deviennent transparents et n'affichent plus leur tooltip.



Si on sélectionne le Canada dans la région de l'Europe, il y a une erreur



Affichage de la Chine uniquement

3. Fichiers Fournis

TP3.html	Squelette HTML du TP
style.css	Code CSS du graphe
données2000.csv et données2014.csv	Données à utiliser
d3.js et d3-tip.js	Librairie de d3.js et du tool-tip
preproc.js	Code de la partie a
scatterplot.js	Code de la partie b
transition.js	Code de la partie c
toolTip.js	Code de la partie d
userInterface.js	Code de la partie e

Remarque :

Si vous utilisez des classes pour la mise en forme des données, mettez votre css dans la balise <style> du fichier HTML



4. Travail demandé

Compléter le code JavaScript des différentes parties pour réaliser le scatter plot. A savoir :

- Le code de *preproc.js*
- Le code de *scatterplot.js*
- Le code de *transition.js*
- Le code de *toolTip.js*
- Le code de *userInterface.js*

De plus, vous devez répondre à la question suivante :

Sans indication particulière, le navigateur charge et exécute les scripts JavaScripts dans le même ordre que celui indiqué dans le fichier HTML. C'est pour cela que dans tous les TP, le script de *d3.js* est chargé en premier. Dans ce TP, vous remarquerez que, hormis les bibliothèques *d3* et *jQuery*, les scripts JavaScripts du TP sont chargés et exécutés dans l'ordre suivant :

1. *preproc.js*
2. *toolTip.js*
3. *transition.js*
4. *userInterface.js*
5. ***main code***
6. ***scatterPlot.js***

Charger *preproc.js*, *toolTip.js* et *userInterface.js* en premier est légitime puisque le code principal utilise des fonctions ou des variables de ces scripts. Le script *transition.js*, hormis ses fonctions, ne fait qu'ajouter un *eventListener* au slider HTML qui est de toute façon créé avant le chargement des scripts et aurait donc pu être chargé à n'importe quel endroit après la balise *div*. Par contre, le code principal utilise des fonctions de *scatterplot.js* qui ne sont normalement pas encore chargées : *axes*, *scatterPlot* et *legend*. Vous pouvez d'ailleurs vérifier en chargeant par exemple *toolTip.js* après le code principal : il y aura une erreur (***toolTip is not defined***).

Question :

Pourquoi alors n'a-t-on pas d'erreur en chargeant le script de *scatterPlot* APRÈS le main code ?