

IFT1015 Programmation 1

Fichiers

Marc Feeley

(avec ajouts/modifications de Pascal Vincent et Aaron

Traitement d'information

- Souvent l'information manipulée par les humains est représentée sous forme de **texte**
- Le texte est donc souvent utilisé pour :
 - les **données fournies** à l'ordinateur par un utilisateur humain (**entrée** ou “input”)
 - les **données produites** par l'ordinateur pour un utilisateur humain (**sortie** ou “output”)



Traitement d'information

- Un programme peut avoir **une ou plusieurs entrées** et **une ou plusieurs sorties**



- Exemple :
 - **entrées** = salaire horaire de chaque employé + nombre d'heures travaillées par chaque employé
 - **sorties** = chèques de paie pour chaque employé + rapport pour l'employeur

Fichiers

- Lorsqu'il y a **peu de données** on peut obtenir l'entrée et produire la sortie **interactivement**, par exemple avec les fonctions **prompt** et **alert** de JS
- Lorsqu'il y a **beaucoup de données** ou bien on veut **préserver les données** pour un usage futur, on stocke les informations dans un **fichier**
- **Fichier** : groupe de données stockées sur un **support matériel persistant** (qui préserve le contenu même lorsqu'il est éteint)
 - Exemples : disque dur, disque compact (CD, CD-RW, DVD), clé USB, carte SD, ruban magnétique

Fichiers

- On mesure la capacité de stockage des supports matériel en MB, GB et TB (méga / giga / tera octets)
 - 1 MB = 10^6 octets = 8 000 000 bits (~1 / 5 bible)
 - 1 GB = 10^9 octets = 8 000 000 000 bits (~200 bibles)
 - 1 TB = 10^{12} octets = 8 000 000 000 000 bits (~200 000 bibles)
- Vu la très grande capacité des supports matériel de stockage, on stocke normalement **plusieurs fichiers sur un même support**
- Le **système de fichier** (“file system”) du système d’exploitation de l’ordinateur organise les fichiers dans une structure d’arbre hiérarchique.

Traitement de fichiers

Traitement de fichiers

- codeBoot a des fonctions prédéfinies pour créer des fichiers et lire leur contenu
 - **readFile** (*path*)
 - retourne un texte avec tout le contenu du fichier *path* (les sauts de ligne deviennent des “\n”)
 - **writeFile** (*path*, *texte*)
 - crée le fichier *path* (s’il n’existe pas déjà) et remplace son contenu par *texte*
- codeBoot utilise un système de fichier interne au fureteur et indépendant de celui du système d’expl.

Traitement de fichiers

- node.js utilise le système de fichier réel
- Cependant, des méthodes différentes sont utilisées pour les opérations d'entrée/sortie
- Pour simuler les fonctions de codeBoot on peut faire :

```
var fs = require("fs");

var readFile = function (path) {
    return fs.readFileSync(path).toString();
};

var writeFile = function (path, texte) {
    fs.writeFileSync(path, texte);
};

var print = function (x) {
    console.log(x);
};
```


Exemple : longueur d'un fichier

- **Spécification** : afficher le nombre de caractères et le nombre de lignes contenues dans un fichier



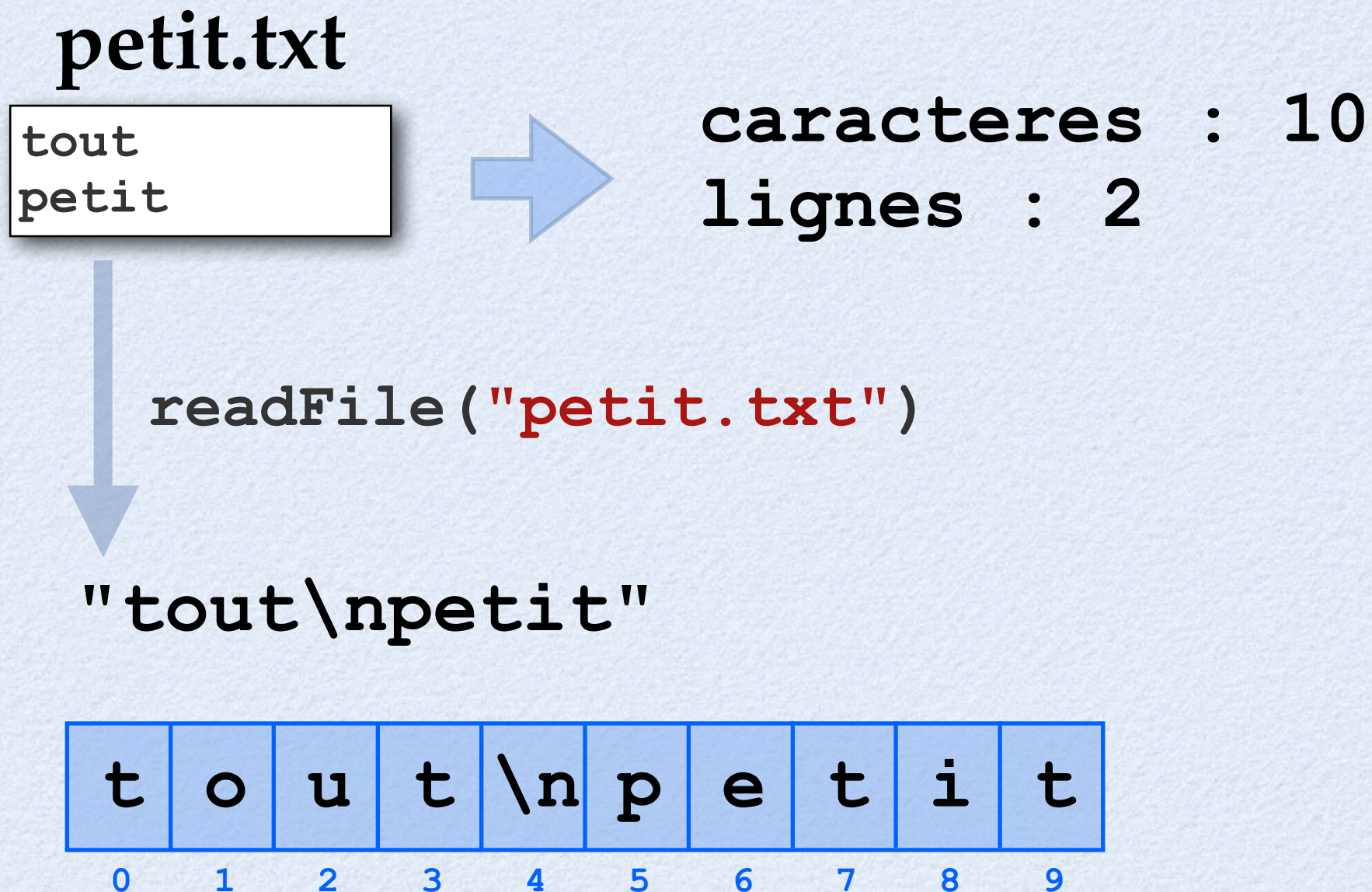
`readFile("petit.txt")`

`"tout\npetit\n"`

t	o	u	t	\n	p	e	t	i	t	\n
0	1	2	3	4	5	6	7	8	9	10

Exemple : longueur d'un fichier

- Il faut noter que la dernière ligne d'un fichier texte pourrait ne pas se terminer par `\n`



Exemple : longueur d'un fichier

- Programme pour codeBoot :

```
var decouperEnLignes = function (contenu) {  
    var lignes = contenu.split("\n");  
    if (lignes[lignes.length-1] == "") {  
        lignes.pop();  
    }  
    return lignes;  
};  
  
var tailleFichier = function (path) {  
    var contenu = readFile(path);  
    var lignes = decouperEnLignes(contenu);  
    print("caracteres = " + contenu.length);  
    print("lignes = " + lignes.length);  
};  
  
tailleFichier("petit.txt");
```


Exemple : longueur d'un fichier

- Avec node.js il faut ajouter au début les fonctions d'entrée / sortie **readFile** et **print**

```
var fs = require("fs");

var readFile = function (path) {
    return fs.readFileSync(path).toString();
};

var print = function (x) {
    console.log(x);
};

var decouperEnLignes = function (contenu) {
    var lignes = contenu.split("\n");
    if (lignes[lignes.length-1] == "") {
        lignes.pop();
    }
    return lignes;
};

var tailleFichier = function (path) {
    var contenu = readFile(path);
    var lignes = decouperEnLignes(contenu);
    print("caracteres = " + contenu.length);
    print("lignes = " + lignes.length);
};

tailleFichier("petit.txt");
```


Exemple : longueur d'un fichier

```
% emacs &                (créer taille-fichier.js et petit.txt)
% nodejs taille-fichier.js
caracteres : 11
lignes : 2
% cat petit.txt
tout
petit
```

- Commandes de shell utiles :
 - **cat** *path* afficher le contenu du fichier *path*
 - **less** *path* affiche *path* une page à la fois

Exemple : longueur d'un fichier

- Malheureusement, les sauts de ligne sont encodés différemment dans divers environnements :

petit.txt

```
tout  
petit
```

`\n` = "linefeed" (code 10)

`\r` = "carriage return" (code 13)

↓ `readFile("petit.txt")`

t	o	u	t	\r	\n	p	e	t	i	t	\r	\n
0	1	2	3	4	5	6	7	8	9	10	11	12

← encodage sous Windows

t	o	u	t	\r	p	e	t	i	t	\r
0	1	2	3	4	5	6	7	8	9	10

← encodage sous Mac (mais dépend du logiciel)

t	o	u	t	\n	p	e	t	i	t	\n
0	1	2	3	4	5	6	7	8	9	10

← encodage sous codeBoot, linux, Mac OS X, Unix

Exemple : longueur d'un fichier

- Pour s'ajuster automatiquement à l'encodage utilisé par l'environnement on peut utiliser cette définition de **decouperEnLignes** :

```
var decouperEnLignes = function (contenu) {  
  
    var lignes = contenu.split("\r\n");  
  
    if (lignes.length == 1) {  
        lignes = contenu.split("\r");  
        if (lignes.length == 1) {  
            lignes = contenu.split("\n");  
        }  
    }  
  
    if (lignes[lignes.length-1] == "") {  
        lignes.pop();  
    }  
  
    return lignes;  
};
```

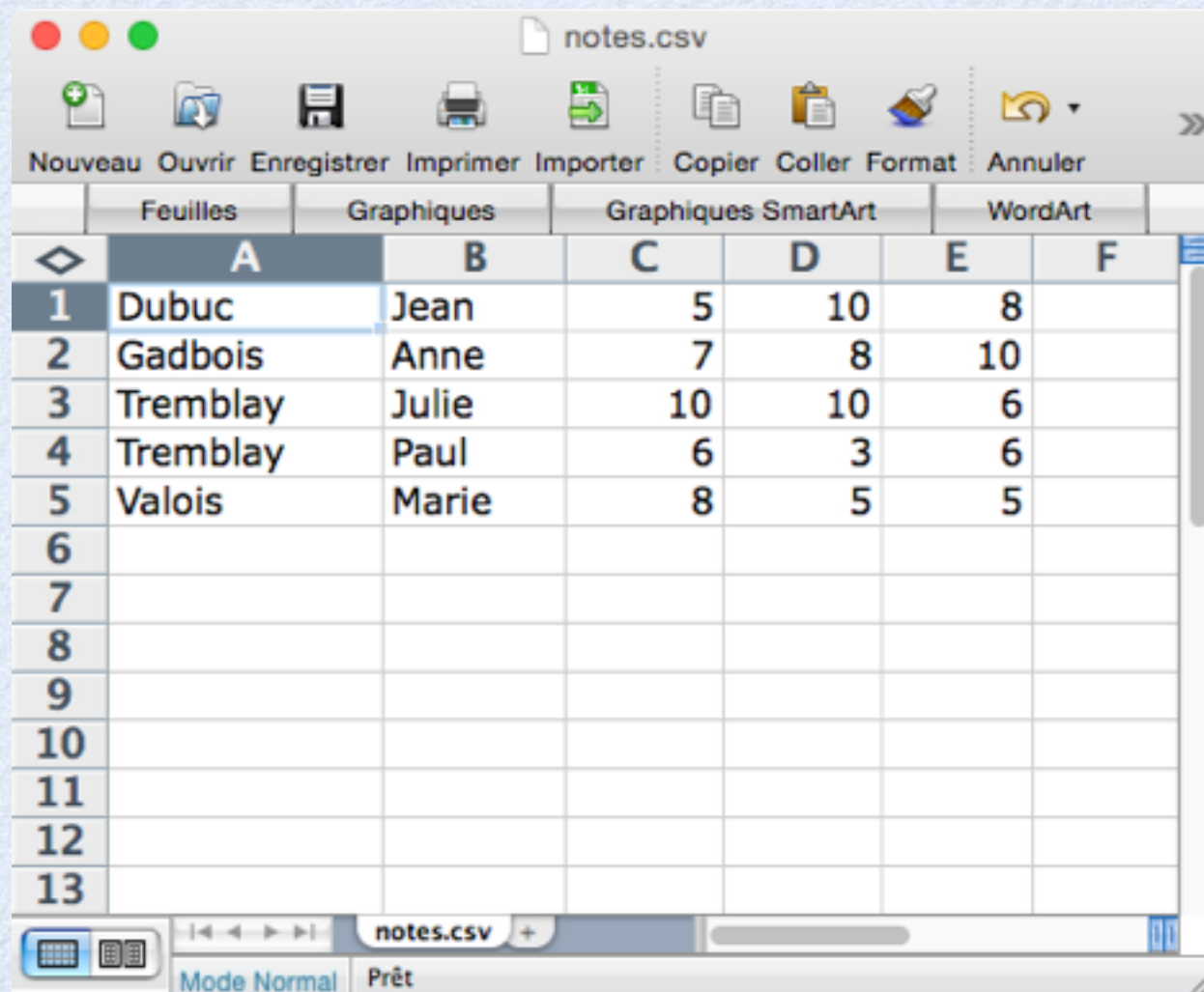

Exemple : fichier CSV

- **CSV** = “Comma Separated Values”
- Fichier CSV : fichier contenant une **matrice 2D** de données où chaque ligne contient une rangée de données et les colonnes sont séparées par des virgules
- Le format de fichier CSV est utilisé pour stocker l'information tabulaire manipulée par les **chiffriers électroniques** (comme Microsoft Excel)
- On peut se servir d'un chiffrier électronique pour **lire** un fichier CSV existant ou pour **créer** un fichier CSV

Exemple : fichier CSV

- Exemple : étudiants d'une classe et leurs notes

Excel



The screenshot shows the Microsoft Excel application window with the file 'notes.csv' open. The spreadsheet contains the following data:

	A	B	C	D	E	F
1	Dubuc	Jean	5	10	8	
2	Gadbois	Anne	7	8	10	
3	Tremblay	Julie	10	10	6	
4	Tremblay	Paul	6	3	6	
5	Valois	Marie	8	5	5	
6						
7						
8						
9						
10						
11						
12						
13						

notes.csv

```
Dubuc , Jean , 5 , 10 , 8
Gadbois , Anne , 7 , 8 , 10
Tremblay , Julie , 10 , 10 , 6
Tremblay , Paul , 6 , 3 , 6
Valois , Marie , 8 , 5 , 5
```

Enregistrer en format CSV

Exemple : fichier CSV

- Pour faciliter le traitement des données, il est bon de lire le contenu du fichier CSV dans un tableau des rangées
- Chaque rangée sera à son tour un tableau des colonnes

notes.csv

```
Dubuc, Jean, 5, 10, 8
Gadbois, Anne, 7, 8, 10
Tremblay, Julie, 10, 10, 6
Tremblay, Paul, 6, 3, 6
Valois, Marie, 8, 5, 5
```



```
[["Dubuc", "Jean", "5", "10", "8"],
 ["Gadbois", "Anne", "7", "8", "10"],
 ["Tremblay", "Julie", "10", "10", "6"],
 ["Tremblay", "Paul", "6", "3", "6"],
 ["Valois", "Marie", "8", "5", "5"]]
```


Exemple : fichier CSV

- Programme pour codeBoot :

```
var decouperEnLignes = function (contenu) {
    var lignes = contenu.split("\n");
    if (lignes[lignes.length-1] == "") {
        lignes.pop();
    }
    return lignes;
};

var lireCSV = function (path) {
    var lignes = decouperEnLignes(readFile(path));
    var resultat = [];
    for (var i=0; i<lignes.length; i++) {
        resultat.push(lignes[i].split(","));
    }
    return resultat;
};

var notes = lireCSV("notes.csv");

print(notes[2]);
```


Exemple : fichier CSV

- Il serait intéressant d'avoir une fonction **ecrireCSV** pour créer un fichier CSV à partir d'une matrice 2D

```
[["Dubuc", "Jean", "5", "10", "8"],  
 ["Gadbois", "Anne", "7", "8", "10"],  
 ["Tremblay", "Julie", "10", "10", "6"],  
 ["Tremblay", "Paul", "6", "3", "6"],  
 ["Valois", "Marie", "8", "5", "5"]]
```


ecrireCSV

notes.csv

```
Dubuc , Jean , 5 , 10 , 8  
Gadbois , Anne , 7 , 8 , 10  
Tremblay , Julie , 10 , 10 , 6  
Tremblay , Paul , 6 , 3 , 6  
Valois , Marie , 8 , 5 , 5
```


Exemple : fichier CSV

- Programme pour codeBoot :

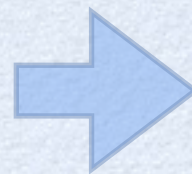
```
var ecrireCSV = function (path, matrice) {  
    var contenu = "";  
    for (var i=0; i<matrice.length; i++) {  
        contenu += matrice[i].join(",") + "\n";  
    }  
    writeFile(path, contenu);  
};  
  
var notes = lireCSV("notes.csv");  
  
ecrireCSV("copie.csv", notes);
```


Exemple : fichier CSV

- **Spécification** : lire un fichier **notes.csv** contenant les notes des étudiants d'une classe et créer le fichier **total.csv** contenant la somme des notes pour chaque étudiant

notes.csv

```
Dubuc , Jean , 5 , 10 , 8  
Gadbois , Anne , 7 , 8 , 10  
Tremblay , Julie , 10 , 10 , 6  
Tremblay , Paul , 6 , 3 , 6  
Valois , Marie , 8 , 5 , 5
```



total.csv

```
Dubuc , Jean , 23  
Gadbois , Anne , 25  
Tremblay , Julie , 26  
Tremblay , Paul , 15  
Valois , Marie , 18
```


Exemple : fichier CSV

- Un problème c'est que la fonction **lireCSV** lit les notes numériques comme des textes
- Il faudra les convertir à des nombres avant de les additionner

notes.csv

```
Dubuc, Jean, 5, 10, 8
Gadbois, Anne, 7, 8, 10
Tremblay, Julie, 10, 10, 6
Tremblay, Paul, 6, 3, 6
Valois, Marie, 8, 5, 5
```



```
[["Dubuc", "Jean", "5", "10", "8"],
 ["Gadbois", "Anne", "7", "8", "10"],
 ["Tremblay", "Julie", "10", "10", "6"],
 ["Tremblay", "Paul", "6", "3", "6"],
 ["Valois", "Marie", "8", "5", "5"]]
```


Exemple : fichier CSV

- Programme pour codeBoot :

```
var calculerTotal = function (matrice) {  
    var resultat = [];  
    for (var i=0; i<matrice.length; i++) {  
        var rangee = matrice[i];  
        var nom = rangee[0];  
        var prenom = rangee[1];  
        var note1 = +rangee[2];  
        var note2 = +rangee[3];  
        var note3 = +rangee[4];  
        resultat.push([nom, prenom, (note1+note2+note3)+""]);  
    }  
    return resultat;  
};  
  
var notes = lireCSV("notes.csv");  
  
var total = calculerTotal(notes);  
  
ecrireCSV("total.csv", total);
```

resultat

```
[["Dubuc", "Jean", "23"],  
 ["Gadbois", "Anne", "25"],  
 ["Tremblay", "Julie", "26"],  
 ["Tremblay", "Paul", "15"],  
 ["Valois", "Marie", "18"]]
```


Exemple : fichier CSV

- Ce programme est assez spécifique à un fichier de notes contenant 3 notes par étudiant
- Un programme plus général devrait permettre un **nombre quelconque de notes par étudiant**
- Pour cela il serait bon de définir ces fonctions :
 - `valNum (["5" , "10" , "8"]) => [5 , 10 , 8]`
 - `somme ([5 , 10 , 8]) => 23`

Exemple : fichier CSV

- Programme pour codeBoot :

```
var valNum = function (tab) {  
    var resultat = [];  
    for (var i=0; i<tab.length; i++) {  
        resultat.push(+tab[i]);  
    }  
    return resultat;  
};  
  
var somme = function (tab) {  
    var s = 0;  
    for (var i=0; i<tab.length; i++) {  
        s += tab[i];  
    }  
    return s;  
};
```


Exemple : fichier CSV

- Programme pour codeBoot :

```
var calculerTotal = function (matrice) {  
    var resultat = [];  
    for (var i=0; i<matrice.length; i++) {  
        var rangee = matrice[i];  
        var nom = rangee[0];  
        var prenom = rangee[1];  
        var notes = rangee.slice(2, rangee.length);  
        var total = somme(valNum(notes));  
        resultat.push([nom, prenom, total+""]);  
    }  
    return resultat;  
};  
  
var notes = lireCSV("notes.csv");  
  
var total = calculerTotal(notes);  
  
ecrireCSV("total.csv", total);
```


Exemple : fichier CSV

- Pour que le traitement des différents champs soit plus facile à comprendre, on peut représenter chaque rangée avec un **enregistrement** (“record”)

notes.csv

```
Dubuc, Jean, 5, 10, 8
Gadbois, Anne, 7, 8, 10
Tremblay, Julie, 10, 9, 6
Tremblay, Paul, 6, 3, 6
Valois, Marie, 8, 5, 5
```

→
lireCSV

```
[["Dubuc", "Jean", "5", "10", "8"],  
 ["Gadbois", "Anne", "7", "8", "10"],  
 ["Tremblay", "Julie", "10", "9", "6"],  
 ["Tremblay", "Paul", "6", "3", "6"],  
 ["Valois", "Marie", "8", "5", "5"]]
```

mat2RecArray

recArray2Mat

```
[{id: {nom: "Dubuc", prenom: "Jean"}, notes: [5, 10, 8]},  
 {id: {nom: "Gadbois", prenom: "Anne"}, notes: [7, 8, 10]},  
 {id: {nom: "Tremblay", prenom: "Julie"}, notes: [10, 9, 6]},  
 {id: {nom: "Tremblay", prenom: "Paul"}, notes: [6, 3, 6]},  
 {id: {nom: "Valois", prenom: "Marie"}, notes: [8, 5, 5]}]
```


Exemple : fichier CSV

```
var valNum = function (tab) {  
  var resultat = [];  
  for (var i=0; i<tab.length; i++) {  
    resultat.push(+tab[i]);  
  }  
  return resultat;  
};  
  
var cols2Rec = function (cols) {  
  return {id: {nom: cols[0], prenom: cols[1]},  
    notes: valNum(cols.slice(2, cols.length))};  
};  
  
var mat2RecArray = function (mat) {  
  var resultat = [];  
  for (var i=0; i<mat.length; i++) {  
    resultat.push(cols2Rec(mat[i]));  
  }  
  return resultat;  
};
```


Exemple : fichier CSV

```
var texteNum = function (tab) {  
    var resultat = [];  
    for (var i=0; i<tab.length; i++) {  
        resultat.push(tab[i]+"");  
    }  
    return resultat;  
};  
  
var rec2Cols = function (rec) {  
    return [rec.id.nom, rec.id.prenom].concat(texteNum(rec.notes));  
};  
  
var recArray2Mat = function (tab) {  
    var resultat = [];  
    for (var i=0; i<tab.length; i++) {  
        resultat.push(rec2Cols(tab[i]));  
    }  
    return resultat;  
};
```


Exemple : fichier CSV

- Version plus propre de `calculerTotal` :

```
var totalRec = function (rec) {  
    return {id: rec.id, notes: [somme(rec.notes)]};  
};  
  
var calculerTotal = function (tab) {  
    var resultat = [];  
    for (var i=0; i<tab.length; i++) {  
        resultat.push(totalRec(tab[i]));  
    }  
    return resultat;  
};  
  
var notes = mat2RecArray(lireCSV("notes.csv"));  
  
var total = calculerTotal(notes);  
  
ecrireCSV("total.csv", recArray2Mat(total));
```


Exemple : fichier CSV

- Cinq des fonctions sont très semblables...

```
var valNum = function (tab) {  
  var resultat = [];  
  for (var i=0; i<tab.length; i++) {  
    resultat.push(+tab[i]);  
  }  
  return resultat;  
};  
  
var texteNum = function (tab) {  
  var resultat = [];  
  for (var i=0; i<tab.length; i++) {  
    resultat.push(tab[i]+"");  
  }  
  return resultat;  
};  
  
var calculerTotal = function (tab) {  
  var resultat = [];  
  for (var i=0; i<tab.length; i++) {  
    resultat.push(totalRec(tab[i]));  
  }  
  return resultat;  
};
```

seules différences

même chose pour mat2RecArray et recArray2Mat

Abstraction : map

- C'est l'occasion de créer une abstraction... "map"

```
var map = function (tab, fn) {  
  var resultat = [];  
  for (var i=0; i<tab.length; i++) {  
    resultat.push(fn(tab[i]));  
  }  
  return resultat;  
};  
  
var valNum = function (tab) {  
  return map(tab, function (x) { return +x; });  
};  
  
var texteNum = function (tab) {  
  return map(tab, function (x) { return x+""; });  
};  
  
var calculerTotal = function (tab) {  
  return map(tab, totalRec);  
};
```

fonctions
passées en
paramètre
à **map**

même chose pour **mat2RecArray** et **recArray2Mat**

Méthode map prédéfinie

- $\text{map}(\text{tableau}, fn) \equiv \text{tableau.map}(fn)$
- La méthode **map** prédéfinie sur les tableaux retourne un **nouveau tableau** de même longueur que *tableau* où chaque élément a été transformé en appelant la fonction *fn*

```
var valNum = function (tab) {  
    return tab.map(function (x) { return +x; });  
};  
  
var texteNum = function (tab) {  
    return tab.map(function (x) { return x+""; });  
};  
  
var calculerTotal = function (tab) {  
    return tab.map(totalRec);  
};
```

même chose pour **mat2RecArray** et **recArray2Mat**

Méthode map prédéfinie

- Exemple avec des nombres :

```
var tab1 = [3, 1, 2, 2];  
var tab2 = tab1.map(function (x) { return x+1; }); // [4,2,3,3]  
var tab3 = tab1.map(function (x) { return x*x; }); // [9,1,4,4]
```

- La méthode **map** passe l'index de l'élément comme deuxième paramètre de la fonction :

```
var tab1 = [3, 1, 2, 2];  
var tab2 = [4, 0, 4, 1];  
var tab3 = tab1.map(function (x,i) { return i; }); // [0,1,2,3]  
var tab4 = tab1.map(function (x,i) { return x+tab2[i]; }); // [7,1,6,3]
```


Méthode fill prédéfinie

- La méthode **fill** prédéfinie sur les tableaux remplace tous les éléments par la valeur en paramètre et retourne le tableau
- C'est utile pour créer des tableaux **initialisés à une valeur spécifique** et pour combiner avec **map** :

```
var tab1 = Array(5); // [undefined,undefined,undefined,undefined,undefined]
var tab2 = Array(5).fill(0); // [0,0,0,0,0]
var tab3 = Array(5).fill(0).map(function (x,i) { return i; }); // [0,1,2,3,4]
```



Nécessaire car **map** saute les éléments **undefined**

Méthode `forEach` prédéfinie

- La méthode **`forEach`** prédéfinie sur les tableaux est similaire à la méthode **`map`**, mais ne retourne pas un tableau des résultats
- C'est utile comme substitut des boucles :

```
var tab = [3, 7, 0];

tab.forEach(function (x) { print(x); }); // imprime: 3
                                           //          7
                                           //          0

tab.forEach(function (x,i) { print(i + ":" + x); }); // imprime: 0:3
                                                         //          1:7
                                                         //          2:0

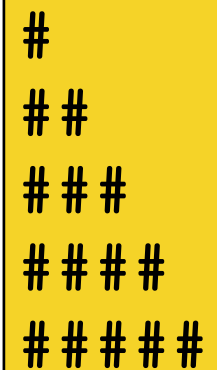
for (var i=0; i<tab.length; i++) { // code équivalent avec boucle for
  print(i + ":" + tab[i]);
}
```


Boucles vs. map et forEach

- Souvent les boucles peuvent être remplacées par des utilisations de **map** et **forEach** :

```
for (var i=1; i<=5; i++) {  
    var barre = "";  
    for (var j=1; j<=i; j++) {  
        barre += "#";  
    }  
    print(barre);  
}
```

sortie :



```
#  
##  
###  
####  
#####
```

```
for (var i=1; i<=5; i++) {  
    print(Array(i).fill("#").join(""));  
}
```

```
print(Array(5).fill(0)  
    .map(function (x,i) {  
        return Array(i+1).fill("#").join("") + "\n";  
    })  
    .join(""));
```


Exemple : fichier CSV

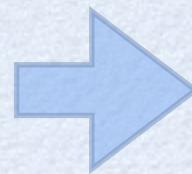
- **Spécification** : lire un fichier **notes1.csv** contenant les notes des étudiants d'une classe et un fichier **notes2.csv** contenant les notes à un travail et modifier le fichier **notes3.csv** pour y ajouter une nouvelle colonne pour le travail

notes1.csv

```
Dubuc,Jean,5,10,8  
Gadbois,Anne,7,8,10  
Tremblay,Julie,10,9,6  
Tremblay,Paul,6,3,6  
Valois,Marie,8,5,5
```

notes2.csv

```
Dubuc,Jean,7  
Gadbois,Anne,10  
Tremblay,Julie,9  
Tremblay,Paul,3  
Valois,Marie,9
```



notes3.csv

```
Dubuc,Jean,5,10,8,7  
Gadbois,Anne,7,8,10,10  
Tremblay,Julie,10,9,6,9  
Tremblay,Paul,6,3,6,3  
Valois,Marie,8,5,5,9
```


Exemple : fichier CSV

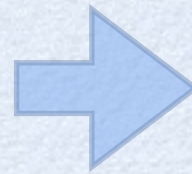
- Note : ce genre de traitement, qui combine deux listes en une, se nomme **fusion** (“merge”)
- Ce serait **plus général** de permettre plusieurs colonnes dans le fichier **notes2.csv**

notes1.csv

```
Dubuc,Jean,5,10,8  
Gadbois,Anne,7,8,10  
Tremblay,Julie,10,9,6  
Tremblay,Paul,6,3,6  
Valois,Marie,8,5,5
```

notes2.csv

```
Dubuc,Jean,7,4  
Gadbois,Anne,10,3  
Tremblay,Julie,9,5  
Tremblay,Paul,3,2  
Valois,Marie,9,9
```



notes3.csv

```
Dubuc,Jean,5,10,8,7,4  
Gadbois,Anne,7,8,10,10,3  
Tremblay,Julie,10,9,6,9,5  
Tremblay,Paul,6,3,6,3,2  
Valois,Marie,8,5,5,9,9
```


Exemple : fichier CSV

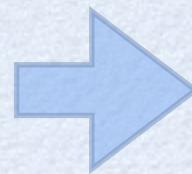
- Ce serait encore plus général de permettre des rangées manquantes dans les fichiers **notes1.csv** et **notes2.csv** (qui seront traitées comme des 0)

notes1.csv

```
Dubuc,Jean,5,10,8  
Gadbois,Anne,7,8,10  
Tremblay,Julie,10,9,6  
Valois,Marie,8,5,5
```

notes2.csv

```
Dubuc,Jean,7,4  
Tremblay,Paul,3,2  
Valois,Marie,9,9
```

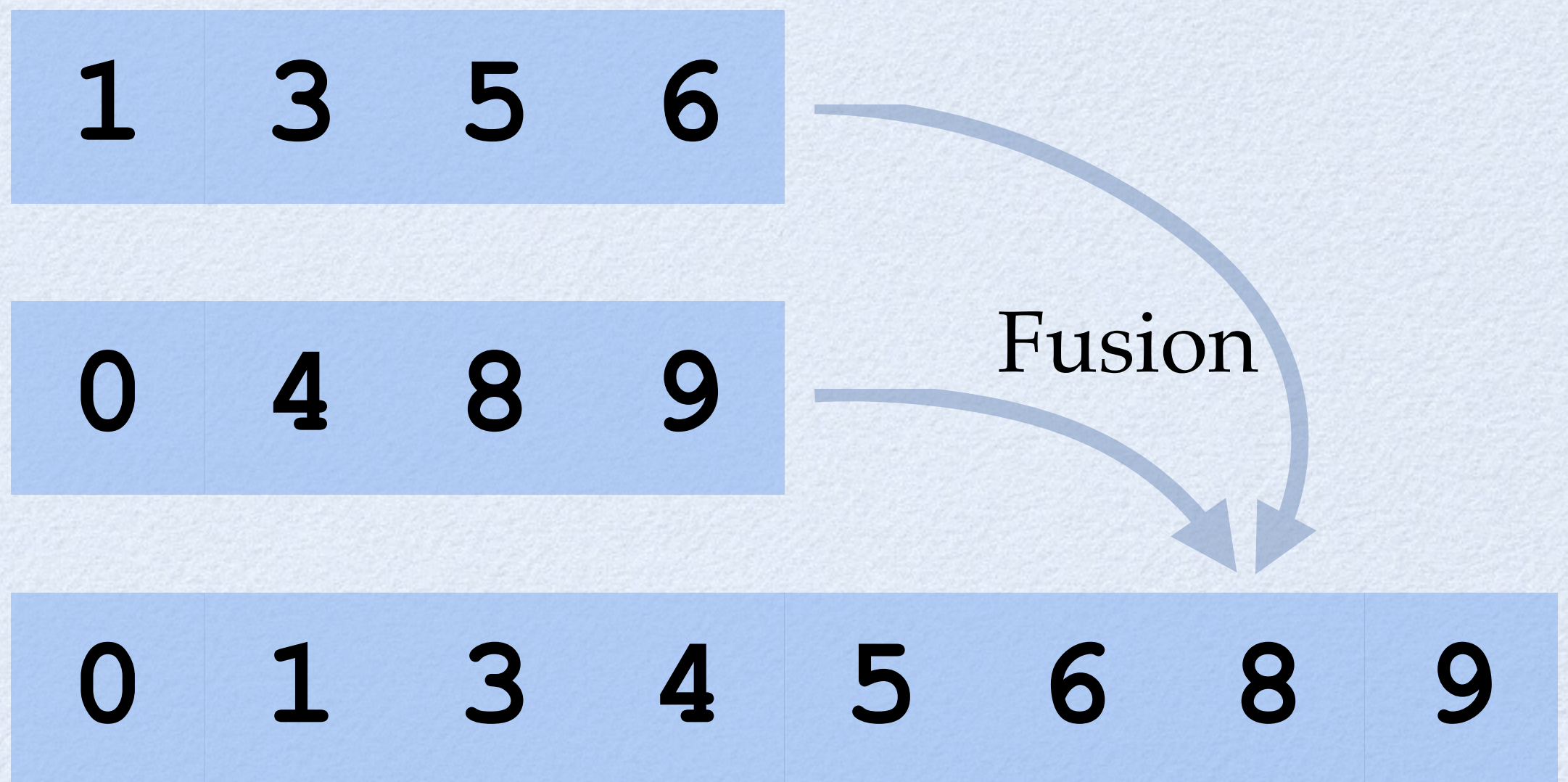


notes3.csv

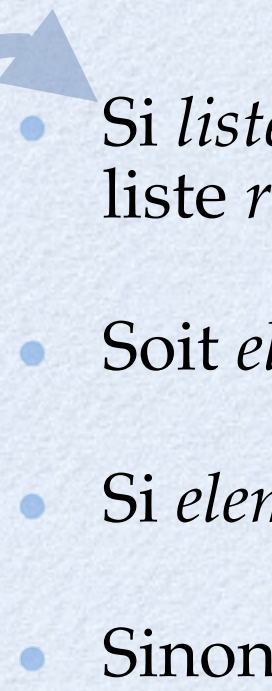
```
Dubuc,Jean,5,10,8,7,4  
Gadbois,Anne,7,8,10,0,0  
Tremblay,Julie,10,9,6,0,0  
Tremblay,Paul,0,0,0,3,2  
Valois,Marie,8,5,5,9,9
```


Exemple : fichier CSV

- Exemple de fusion de listes de nombres :



Exemple : fichier CSV

- **Algorithme de fusion général :**
 - Étant donné deux listes d'éléments triées en ordre croissant *liste1* et *liste2*
 - Créer une liste *résultante* vide
 - Si *liste1* ou *liste2* est vide, ajouter l'autre liste à la liste *résultante* et retourner la liste *résultante*
 - Soit *elem1* le premier élément de la *liste1* et *elem2* le premier élément de la *liste2*
 - Si $elem1 < elem2$, retirer *elem1* de la *liste1* et l'ajouter à la fin de la liste *résultante*
 - Sinon, retirer *elem2* de la *liste2* et l'ajouter à la fin de la liste *résultante*
- 

la liste *résultante* sera donc aussi triée en ordre croissant

Exemple : fichier CSV

- Exemple de fusion de listes de nombres :

liste1

1	3	5	6
--------------	--------------	--------------	--------------

liste2

0	4	8	9
--------------	--------------	--------------	--------------

résultante

0	1	3	4	5	6	8	9
---	---	---	---	---	---	---	---

Exemple : fichier CSV

- Fusion des listes de nombres :

```
var fusion = function (liste1, liste2) {  
  
    var resultat = [];  
    var i = 0;    // index du prochain élément de liste1  
    var j = 0;    // index du prochain élément de liste2  
  
    while (i < liste1.length && j < liste2.length) {  
        if (liste1[i] < liste2[j])  
            resultat.push(liste1[i++]);  
        else  
            resultat.push(liste2[j++]);  
    }  
  
    while (i < liste1.length) resultat.push(liste1[i++]);  
    while (j < liste2.length) resultat.push(liste2[j++]);  
  
    return resultat;  
};  
  
print( fusion([1,3,5,6],[0,4,8,9]) );
```


Exemple : fichier CSV

- Pour l'algorithme de fusion des notes il faut faire la **comparaison des identifiants** à la tête des listes

```
// La fonction comparerId(id1,id2) retourne 0 si les  
// deux identifiants sont égaux, -1 si id1 est  
// plus petit que id2, et 1 si id1 est plus grand  
// que id2.
```

```
var comparerId = function (id1, id2) {  
    if (id1.nom < id2.nom) return -1;  
    if (id1.nom > id2.nom) return 1;  
    if (id1.prenom < id2.prenom) return -1;  
    if (id1.prenom > id2.prenom) return 1;  
    return 0;  
};
```

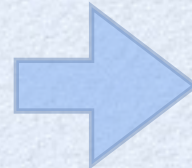
```
var comparerRec = function (rec1, rec2) {  
    return comparerId(rec1.id, rec2.id);  
};
```


Exemple : fichier CSV

- L'enregistrement avec le **plus petit identifiant** est transféré à la liste résultante
- Cependant, si deux enregistrements ont le même identifiant il faut **combiner** les enregistrements (concaténation des notes)

```
Dubuc, Jean, 5, 10, 8  
Gadbois, Anne, 7, 8, 10  
Tremblay, Julie, 10, 9, 6  
Valois, Marie, 8, 5, 5
```

```
Dubuc, Jean, 7, 4  
Tremblay, Paul, 3, 2  
Valois, Marie, 9, 9
```



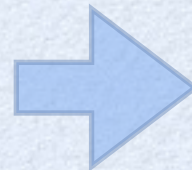
```
Dubuc, Jean, 5, 10, 8, 7, 4  
Gadbois, Anne, 7, 8, 10, 0, 0  
Tremblay, Julie, 10, 9, 6, 0, 0  
Tremblay, Paul, 0, 0, 0, 3, 2  
Valois, Marie, 8, 5, 5, 9, 9
```


Exemple : fichier CSV

- Si un enregistrement existe seulement dans une liste il faut le **compléter** avec des notes 0
- Dans ce cas il faut **savoir combien de colonnes de notes** il y a dans l'autre liste
- Il faut donc traiter les listes de notes vides spécialement

```
Dubuc, Jean, 5, 10, 8  
Gadbois, Anne, 7, 8, 10  
Tremblay, Julie, 10, 9, 6  
Valois, Marie, 8, 5, 5
```

```
Dubuc, Jean, 7, 4  
Tremblay, Paul, 3, 2  
Valois, Marie, 9, 9
```



```
Dubuc, Jean, 5, 10, 8, 7, 4  
Gadbois, Anne, 7, 8, 10, 0, 0  
Tremblay, Julie, 10, 9, 6, 0, 0  
Tremblay, Paul, 0, 0, 0, 3, 2  
Valois, Marie, 8, 5, 5, 9, 9
```


Exemple : fichier CSV

```
var tableauNul = function (n) {  
  var tab = Array(n);  
  for (var i=0; i<n; i++) tab[i] = 0;  
  return tab;  
};
```

fonction utilitaire pour
créer des tableaux de 0
(aurait pu utiliser **fill**)

```
var fusion = function (liste1, liste2) {  
  
  if (liste1.length == 0) return liste2; // cas spéciaux  
  if (liste2.length == 0) return liste1;  
  
  var tabNul1 = tableauNul(liste1[0].notes.length);  
  var tabNul2 = tableauNul(liste2[0].notes.length);  
  
  var resultat = [];  
  var i = 0; // index du prochain élément de liste1  
  var j = 0; // index du prochain élément de liste2  
  
  ...
```


Exemple : fichier CSV

...

```
while (i < liste1.length && j < liste2.length) {  
    var rec1 = liste1[i];  
    var rec2 = liste2[j];  
    var comp = comparerRec(rec1, rec2);  
    if (comp < 0) {  
        resultat.push({id: rec1.id,  
                        notes: rec1.notes.concat(tabNul2)});  
        i++;  
    } else if (comp > 0) {  
        resultat.push({id: rec2.id,  
                        notes: tabNul1.concat(rec2.notes)});  
        j++;  
    } else {  
        resultat.push({id: rec1.id,  
                        notes: rec1.notes.concat(rec2.notes)});  
        i++;  
        j++;  
    }  
}
```

...

Exemple : fichier CSV

```
...  
  
while (i < liste1.length) {  
    var rec1 = liste1[i++];  
    resultat.push({id: rec1.id,  
                   notes: rec1.notes.concat(tabNul2) });  
}  
  
while (j < liste2.length) {  
    var rec2 = liste2[j++];  
    resultat.push({id: rec2.id,  
                   notes: tabNul1.concat(rec2.notes) });  
}  
  
return resultat;  
};
```

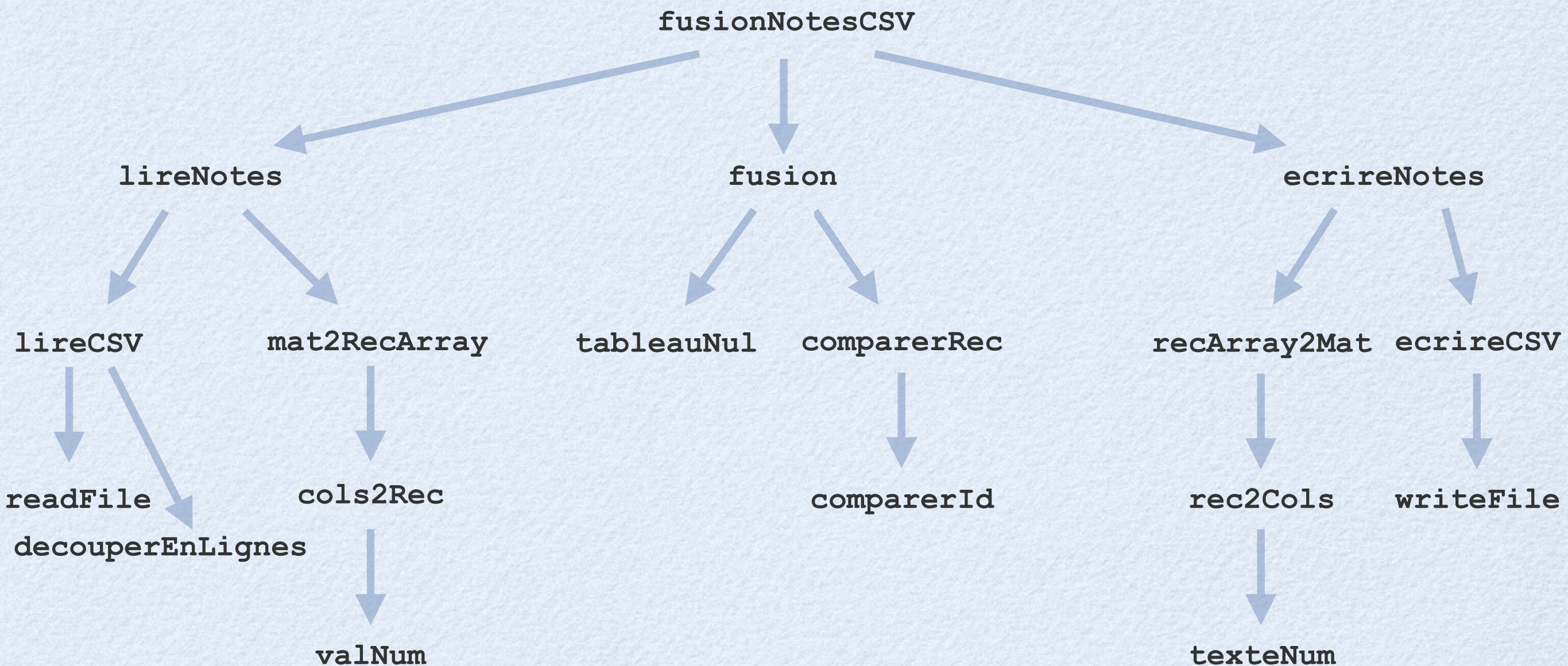

Exemple : fichier CSV

- Programme final :

```
var lireNotes = function (path) {  
    return mat2RecArray(lireCSV(path));  
};  
  
var ecrireNotes = function (path, notes) {  
    ecrireCSV(path, recArray2Mat(notes));  
};  
  
var fusionNotesCSV = function (entree1, entree2, sortie) {  
  
    var notes1 = lireNotes(entree1);  
    var notes2 = lireNotes(entree2);  
  
    var notes3 = fusion(notes1, notes2);  
  
    ecrireNotes(sortie, notes3);  
};  
  
fusionNotesCSV("notes1.csv", "notes2.csv", "notes3.csv");
```


Graphe d'appel

- **Graphe de décomposition fonctionnelle** (aussi connu sous le nom **graphe de la relation appelant-appelé**, ou **graphe d'appel**):



Exemple : concaténation

- **Spécification** : lire un fichier **fichiers.txt** contenant une liste des chemins d'accès de fichiers à concaténer et créer le fichier **concatenation.txt** contenant la concaténation de ces fichiers

fichiers.txt

```
intro.txt  
corps.txt  
conclusion.txt
```

intro.txt

```
Il était  
une fois
```

corps.txt

```
une belle  
princesse
```

conclusion.txt

```
et une grenouille  
charmante.
```



concatenation.txt

```
Il était  
une fois  
une belle  
princesse  
et une grenouille  
charmante.
```


Exemple : concaténation

- Composition de **readFile**, **map** et **join** :

```
writeFile("concatenation.txt",  
    decouperEnLignes(readFile("fichiers.txt"))  
    .map(function (path) { return readFile(path); })  
    .join(""));
```