TRAITEMENT DE DONNEES EN TABLES :

- *Indexation de tables*
- *Recherche dans une table*
- *Tri dans une table*
- *Fusion de tables*

“

En 1978, Daniel Bricklin, étudiant à Harvard, devait établir des tableaux comptables pour une étude de cas sur Pepsi-Cola. Plutôt que de calculer à la main il préféra programmer « un tableau noir et une craie électroniques », selon sa propre expression. Son premier prototype pouvait manipuler un tableau de vingt lignes et cinq colonnes.

En janvier 1979, avec Bob Frankston, son associé, ils lancent la commercialisation de VisiCalc (« Visible Calculator »).

VisiCalc était vendu 100 $. Il avait déjà l’allure des tableurs d’aujourd’hui !

”

Numérique et Sciences Informatiques
1ère

Support de cours :
Jean-Christophe BONNEFOY

Objectifs :

- Importer une table depuis un fichier CSV.
- Rechercher les lignes d’une table correspondant à des critères exprimés en logique propositionnelle.
- Trier une table suivant un descripteur.
- Construire une nouvelle table en combinant des données de plusieurs tables.

1. Rappels de seconde

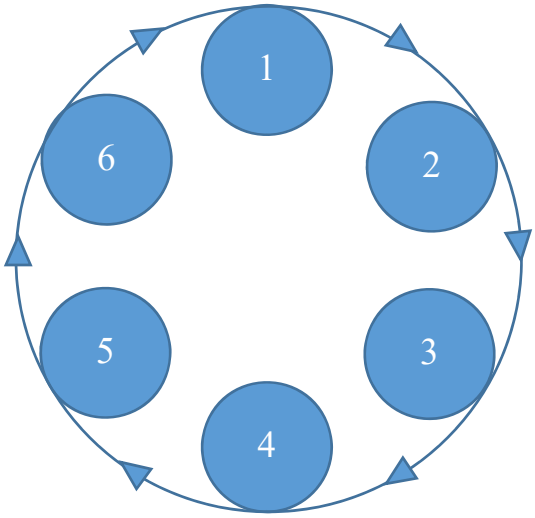
1.1. Une donnée, c'est quoi ?

Une **donnée (*data* en anglais)** est une valeur attribuée à une entité pour la décrire. Cette entité peut être un objet, une personne, un événement, etc.

Une donnée peut être élémentaire ou complexe.

- Une donnée **élémentaire** représente une caractéristique de base (un nom, un numéro, etc.). Cette donnée est caractérisée par un **descripteur** qui permet de donner le format dans lequel cette donnée est représentée.
- Une donnée **complexe** est constituée de plusieurs données élémentaires.

Les données constituent la matière première de toute activité numérique. Ces données ne sont cependant pas permanentes et possèdent un cycle de vie bien défini comme le montre la figure suivante :



1. **Collecte :** Il faut commencer par recueillir les données.
2. **Traitement et Partage :** Étant donné la grande quantité de données, il est indispensable de les traiter et de les partager aux différents acteurs afin de préparer leur analyse.
3. **Analyse :** L'analyse permet de donner du sens aux données afin de mettre en place des actions. A l'issue de cette phase, on parle d'informations.
4. **Sauvegarde :** Il est nécessaire de sauvegarder les données pour pouvoir les restaurer à l'identique en cas de panne ou de perte d'un support de stockage.
5. **Archivage :** Les données doivent être conservées uniquement pour la durée nécessaire à l'opération pour laquelle elles ont été recueillies et traitées.
6. **Destruction :** Les données sont détruites lorsqu'elles deviennent obsolètes.

1.2. Les données personnelles et leurs protections

Une **donnée personnelle** est une donnée qui identifie directement ou indirectement une personne physique.

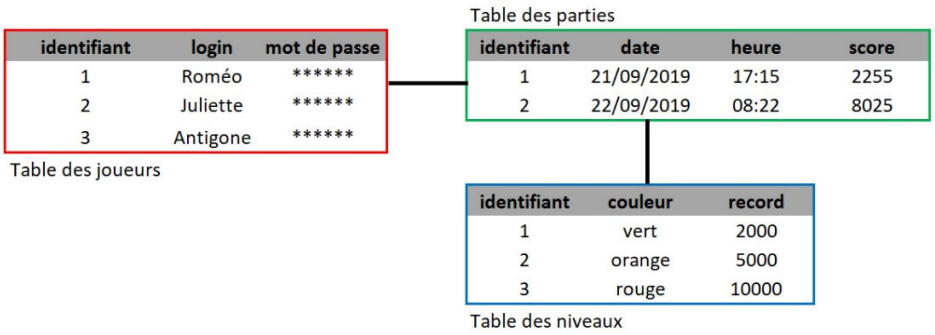
Les données personnelles sont protégées dans tous les états membres de l'Union Européenne par une **loi Informatique et libertés.** En effet, depuis 2018, le Règlement Général de la Protection des Données (RGPD) oblige tout organisme qui collecte des données à prouver la nécessité de cette collecte, à protéger les données recueillies et à être plus transparent sur leurs utilisations**.** En France, l'autorité compétente est la Commission Nationale de l'Informatique et des Libertés (**CNIL**). Elle est chargée de veiller à la protection de l'identité humaine, des droits de l'homme, de la vie privée et des libertés individuelles.

1.3. La structuration des données

Les données utilisées pour une application donnée sont souvent très nombreuses. Il est donc nécessaire de les organiser pour pouvoir les utiliser de manière efficace. Lorsque des données partagent les mêmes descripteurs, on les rassemble dans une **collection.**

On les présente souvent sous forme de tables dont les colonnes représentent les descripteurs et les lignes représentent les données. La valeur de cette donnée se lit ainsi à l'intersection de la ligne et de la colonne. D'une façon générale, lorsque l'on peut classer les données sous la forme de tables, on parle de **données structurées**. Lorsque l'on regroupe des collections de données reliées entre elles, on parle de **base de données**. Une base de données est représentée dans un format spécifique qui fait apparaitre les collections (sous forme de tables) et les liens entre ces collections (appelées relations).

Série	Titre de l'album	Dessinateur	Editeur	Année de parution
Tintin	Tintin au Tibet	Hergé	Casterman	1959
Astérix le gaulois	La serpe d'or	Uderzo	Dargaud	1962
Lefranc	La grande menace	Martin	Le Lombard	1954
Blake et Mortimer	La marque jaune	Jacobs	Le Lombard	1956



1.4. Les formats de fichiers des données structurées

Pour assurer la persistance des données, ces dernières sont stockées dans des fichiers. Il en existe une multitude, cependant les deux formats les plus utilisés sont le format CSV et le format JSON.

5.1.1. Le format CSV (*Comma-Separated Value*)

Dans un fichier au format CSV, les données sont présentées dans un fichier texte, dont les descripteurs sont séparés par un caractère spécifique. Les caractères les plus connus sont la virgule, le point-virgule (comme le montre la figure ci-après) ou encore la tabulation.

```
Nom ;prénom;login;mot de passe;service
MARTIN;Louis;ML;******;Administratif
DURAND;Claire;DC;******;Commercial
DUPOND;Georges;DG;******;Technique
DUBOIS;Odile;DO;******;Direction
```

Les fichiers CSV peuvent être convertis sous forme de tableau, notamment via un tableur (comme Excel par exemple). Il devient alors possible d'organiser et de trier les données.

	A	B	C	D	E
1	Nom	prénom	login	mot de passe	service
2	MARTIN	Louis	ML	******	Administratif
3	DURAND	Claire	DC	******	Commercial
4	DUPOND	Georges	DG	******	Technique
5	DUBOIS	Odile	DO	******	Direction
6					
7					

Le même fichier CSV mis sous forme de tableau

5.1.2. Le format JSON (*JavaScript Objet Notation*)

Dans un fichier au format JSON, les données sont présentées dans un fichier texte en utilisant la syntaxe d'un langage très utilisé sur internet, le JavaScript.

L'intérêt du format JSON est qu'il permet un stockage de données plus complexes que celles présentes dans un fichier CSV. Il associe des paires de *descripteur/valeur* séparées par le caractère « : », et chaque paire est séparée de la suivante par le caractère « , ».

```
{
        "espèce" : "Chien"
        "age" : 6,
        "race" : "Golden Retriever",
        "trait" : {
                "CouleurYeux" : "Marron",
                "CouleurPelage" : "Jaune"
        }
}
```

Remarque : la quatrième valeur de l'exemple ci-dessus est une donnée imbriquée qui comprend 2 paires de *descripteur/valeur*. On peut ainsi imbriquer les objets à l'infini !

Dans le cas de l'exemple ci-dessus, on voit très bien que l'on ne peut pas présenter ces données complexes sous la forme d'une seule table puisqu'il est impossible de représenter le 4e descripteur. Le fichier JSON se rapproche donc plus d'une base de données. Ce format est souvent utilisé pour récupérer ou échanger des données sur internet.

On pourra donc toujours faire correspondre un fichier CSV à un fichier JSON. La réciproque, en revanche, est fausse.

2. Le traitement des données structurées

Le **traitement des données** consiste à effectuer sur les données structurées des actions ou des combinaisons d'actions afin d'obtenir un ou plusieurs résultats (des informations) correspondant à un besoin précis.

Ces actions peuvent être de nature complexe, il s'agit cependant dans la majorité des cas d'actions élémentaires comme l'affichage, la recherche, le tri et le filtrage. On peut traiter ces données à partir du tableur, mais il est important de comprendre comment ces fonctions peuvent être programmées.

Dans cette partie, on prendra comme données, le fichier du recensement de la population française en vigueur au 1er janvier 2019 issu de l'Institut National de la Statistique et des Etudes Economiques (INSEE).

	A	B	C	D	E	F	G	H	I
1	Code région	Nom de la région	Code département	Nom du département	Nombre d'arrondissements	Nombre de cantons	Nombre de communes	Population totale	Superficie (en km²)
2	84	Auvergne-Rhône-Alpes	01	Ain	4	23	407	655 171	5 762
3	32	Hauts-de-France	02	Aisne	5	21	804	549 587	7 369
4	84	Auvergne-Rhône-Alpes	03	Allier	3	19	317	349 336	7 340
5	93	Provence-Alpes-Côte d'Azur	04	Alpes-de-Haute-Provence	4	15	198	167 331	6 925
6	93	Provence-Alpes-Côte d'Azur	05	Hautes-Alpes	2	15	163	146 148	5 549
7	93	Provence-Alpes-Côte d'Azur	06	Alpes-Maritimes	2	27	163	1 098 539	4 299
8	84	Auvergne-Rhône-Alpes	07	Ardèche	3	17	339	334 591	5 529
9	44	Grand Est	08	Ardennes	4	19	452	283 004	5 229
10	76	Occitanie	09	Ariège	3	13	331	158 205	4 890
11	44	Grand Est	10	Aube	3	17	431	316 639	6 004
12	76	Occitanie	11	Aude	3	19	436	377 580	6 139
13	76	Occitanie	12	Aveyron	3	23	285	289 481	8 735

Extrait du fichier contenant ces données

2.1. Importation des données

L'objectif est d'extraire ces données puis de les enregistrer dans une liste composée de listes, de tuples ou de dictionnaires.

Voici un exemple de fonction permettant d'importer les données et qui retourne :

- Une liste des descripteurs (au format chaine de caractères)
- Une liste de liste contenant les données (avec les données de certains descripteurs en type entier)

```
def Importation():
    fichier = open("population_superficie_departements.csv", "r")
    ligne_descripteurs = fichier.readline()   # Lecture des descripteurs
    lst_descripteurs = ligne_descripteurs.rstrip().split(";")
    lignes = fichier.readlines()
    table = []
    for ligne in lignes:
        lst = ligne.rstrip().split(";")
        lst[4] = int(lst[4])
        lst[5] = int(lst[5])
        lst[6] = int(lst[6])
        lst[7] = int(lst[7])
        lst[8] = int(lst[8])
        table.append(lst)

    fichier.close()
    return lst_descripteurs, table
```

Remarques :

- la fonction *rstrip()* supprime ici les caractères de fin de ligne '\n' situés à droite. Elle ne se résume pas à cela, il faudra lire la documentation de cette fonction pour la maitriser parfaitement.
- la fonction *split(' ;')* convertir une chaine de caractères en liste en séparant avec le caractère ' ;'

2.2. Recherche dans une table

La **recherche** dans une table consiste à obtenir les valeurs de certains descripteurs (champs) avec éventuellement des critères les concernant.

Problématique : *Quels sont les départements dont la population est inférieure 300 000 habitants*

Il nous faut connaitre les indices des descripteurs 'Nom du département' et 'Population totale'.

Voici un exemple de fonction permettant la recherche suivant notre problématique à partir de la fonction *Importation()* vu en 2.1.

```
20 def Recherche(descripteurs, tbl):
21     indices_desc = [descripteurs.index('Nom du département'),
22                     descripteurs.index('Population totale')]
23     tabl_recherche = []
24     for ligne in tbl:
25         if ligne[indices_desc[1]] < 300000:
26             tabl_recherche.append( ligne[indices_desc[0]])
27     return tabl_recherche
28
29
30 champs, donnees = Importation()
31 resultat = Recherche(champs, donnees)
32 print(resultat)
```

Remarque :

Dans certain cas, il faudra vérifier également que dans le résultat de la recherche qu'il n'y a pas de doublons.

2.3. Tri d'une table

Le **tri** dans une table consiste à modifier l'ordre des données pour qu'elles soient présentées dans un ordre croissant ou décroissant selon le choix d'un ou plusieurs critères.

Problématique : *Trier les données par région, puis au sein de ces régions, trier par département du moins peuplé au plus peuplé.*

Il nous faut là aussi connaitre les indices de certains descripteurs : 'Nom de la région', 'Nom du département' et 'Population totale'.

Voici un exemple de fonction permettant le tri suivant notre problématique à partir de la fonction *Importation()* vu en 2.1. On utilisera la fonction *itemgetter()* du module *Operator* et la fonction *sorted()* qui réalise le tri. On verra d'autre méthode de tri dans une autre partie du programme.

```
from operator import itemgetter
def Tri(descripteurs, tbl):
    ind_desc = [descripteurs.index('Nom de la région'),
                descripteurs.index('Nom du département'),
                descripteurs.index('Population totale')]

    tri = sorted(tbl, key=itemgetter(ind_desc[0],ind_desc[2]))
    res = []
    for l in tri:
        res.append([l[ind_desc[0]], l[ind_desc[1]], l[ind_desc[2]]])
    return res


champs, donnees = Importation()
resultat = Tri(champs, donnees)
print(resultat)
```

Remarque :

Dans notre cas, nous sélectionnons uniquement les 3 descripteurs plutôt que de renvoyer tous les champs.

2.4. Fusion de tables

On peut distinguer 2 cas :

- **Concaténation** : on ajoute des lignes de données présentes dans la table 2 à la table 1
- **Jointure** : On ajoute des descripteurs présents dans la table 2 à ceux présents dans la table 1

Remarque :

Il faut bien évidement faire attention à ce que les données soient compatibles entres elles.

Exemple de Concaténation :

On a inventé une nouvelle région imaginaire issue de l'univers de Tintin :

	A	B	C	D	E	F	G	H	I	J
1	Code région	Nom de la ré	Code départ	Nom du dép	Nombre d'ar	Nombre de c	Nombre de c	Population t	Superficie (en km²)	
2	IM1	Tintinoland	234	Syldavie	4	1	407	100000	1200	
3	IM1	Tintinoland	230	Bordurie	3	1	317	120000	1500	
4	IM1	Tintinoland	232	Khemed	2	1	339	53000	5529	
5	IM1	Tintinoland	233	Rawajpoutal	3	1	247	25000	280	
6	IM1	Tintinoland	231	Gopal	2	1	8	12500	7520	
7										

Les descripteurs sont identiques entre les 2 tables.

```
def Concatenation(tbl1, tbl2):
    res = tbl1 + tbl2
    return res

champs, donnees = Importation("population_superficie_departements.csv")
champs2, donnees2 = Importation("tintinoland.csv")
resultat = Concatenation(donnees, donnees2)
print(resultat)
```

Remarque :

On a modifié au préalable notre fonction *Importation* pour qu'elle prenne en argument le nom d'un fichier CSV.

Exemple de Jointure :

Prenons maintenant comme deuxième table, un fichier CSV listant les préfectures des département français. Après avoir importé ces données dans une deuxième table, on souhaite faire la jointure pour mettre la préfecture au bon département dans la première table.

```
53 def Importation_simple(fic_csv):
54     fichier = open(fic_csv, "r")
55 
56     ligne_descripteurs = fichier.readline()   # Lecture des descripteurs
57     lst_descripteurs = ligne_descripteurs.rstrip().split(";")
58     lignes = fichier.readlines()
59     table = []
60     for ligne in lignes:
61         lst = ligne.rstrip().split(";")
62         table.append(lst)
63 
64     fichier.close()
65     return lst_descripteurs, table
66 
67 def Jointure(tbl1, desc1, tbl2, desc2):
68     desc1.append(desc2[2])
69     for l in tbl2:
70         for ligne in tbl1:
71             if ligne[3] == l[1]:
72                 ligne.append(l[2])
73 
74 champs, donnees = Importation_simple("population_superficie_departements.csv")
75 champs2, donnees2 = Importation_simple("prefectures.csv")
76 Jointure(donnees, champs, donnees2, champs2)
77 print(donnees)
78 
```

Il existe également des modules pour traiter directement des tables comme par exemple le module *pandas*.