

# STRUCTURES DE DONNEES A 2 DIMENSIONS (1) : TABLES DE DONNEES

*A remplir en fin de livret.*

Type de tableau	Implémentation
Tableau sans entêtes	
Tableau à entêtes colonnes	
Tableau à entêtes lignes	
Tableau à double entêtes lignes colonnes	

*Me prévenir de toute erreur éventuelle.*

I. Structures de données : faisons le point.	2
II. Des tableaux sans entêtes aux tables de données.	3
III. Des tableaux avec entêtes aux tables de données.	6
IV. Table de données et fichier CSV.	9
V. Manipuler un fichier à partir d'un programme.	11
VI. Manipuler des fichiers csv à l'aide du module csv.	16

➤ Logiciels et sites internet : pythontutor.com, éditeur et console Python (Thonny, Replit, VS Code etc.).

➤ Pré-requis pour prendre un bon départ :

	☹	☺	☺☺	☺☺☺
Listes.				
Compréhension de liste.				
Ecrire simplement un dictionnaire.				
Générer un dictionnaire par un programme.				
Lire la valeur d'un item d'un dictionnaire.				
Remplacer la valeur d'un item d'un dictionnaire.				
Ajouter des items à un dictionnaire.				



Lorsque le logo Python apparaît, cela signifie que l'activité doit être faite sur ordinateur.

**VERIFIEZ VOS REPONSES SUR ORDINATEUR !**

NOM et prénom : .....

Première spécialité NSI

# I. STRUCTURES DE DONNEES : FAISONS LE POINT.

Qu'a-t-on vu jusqu'à maintenant sur les structures de données c-à-d les façons d'organiser les données ?

Type d'informations	Exemple	Représentation schématique	Représentation informatique	Géométrie	
<b>Informations ponctuelles sans liens entre elles.</b>	vissable malgache 871	<b>Cases indépendantes :</b> 	<b>Variables indépendantes :</b> a = 'vissable' b = 'malgache' c = 871	Points non alignés : 	Dimension 0
Ces informations ponctuelles peuvent aussi être liées c-à-d avoir quelque chose en commun. Par exemple être des informations provenant d'une même entité :					
<b>Informations liées pour 1 seule entité.</b>	Monslyp Jean-Phil 65	<b>Liste (tableau 1 seule ligne) :</b> 	..... ou ..... : [ 'Monslyp' , 'Jean-Phyl' , 65 ] ou ( 'Monslyp' , 'Jean-Phyl' , 65 )	Points alignés : 	Dimension 1
La représentation de ces informations liées n'est pas toujours explicite. Dans l'exemple ci-dessus, on ne sait pas à quoi correspond 65 (est-ce l'âge ? le poids ? le QI ?). D'où l'idée de nommer (étiqueter, décrire, donner un attribut, labelliser) les données :					
<b>Informations liées nommées pour 1 seule entité.</b>	Nom : Bonheur Prénom : Jean Poids : 65	<b>Liste avec noms de colonne :</b> 	..... : { 'Nom' : 'Bonheur' , 'Prénom' : 'Jean' , 'Age' : 65 }	Points nommés alignés : 	Dimension 1
Evidemment, on peut avoir des informations liées (étiquetées ou non) mais pour plusieurs entités au lieu d'une seule :					
<b>Informations liées pour plusieurs entités.</b>	Notes de 2 élèves : ○ 5 ; 14 ; 17 ○ 12 ; 7 ; 14	<b>Tableau :</b> 	<b>Liste de listes (ou de tuples) de même taille :</b> [ [ 5 , 14 , 17 ] , [ 12 , 7 , 14 ] ] ou [ ( 5 , 14 , 17 ) , ( 12 , 7 , 14 ) ]	Quadrillage de points : 	Dimension 2
<b>Informations liées nommées pour plusieurs entités.</b>	2 fiches d'informations : Noms : Bono , Dantic Prénoms : Jean , Ali Ages : 16 , 59	<b>Tableau avec entêtes colonnes :</b> 	<b>Liste de dictionnaires ayant les mêmes clés :</b> [ { 'Nom' : 'Bono' , 'Prénom' : 'Jean' , 'Age' : 16 } , { 'Nom' : 'Dantic' , 'Prénom' : 'Ali' , 'Age' : 59 } ]	Quadrillages de points nommés : 	Dimension 2

Données en dimension 0 (variables simples) → cours 1 : Débuter en Python. Données en dimension 1 → cours 4 , 5 , 6 : Listes , Tuples , Dictionnaires.

Nous allons donc maintenant nous intéresser aux données en dimension 2 : les données sous forme de (vrais) tableaux.

En Informatique, un tableau de données s'appelle une **table de données** (data table en anglais).

## II. DES TABLEAUX SANS ENTETES AUX TABLES DE DONNEES.

### A. Exemples de tableaux sans entêtes dans la vraie vie :

Tableau de notes pour 2 élèves

11	2	13
4	15	11

Tableau farfelu

a	b
4	0
f	3

Tableau de personnes

Jean	Balle
Luce	Tucru
Vic	Tim de Lamaude

- notes du 1<sup>er</sup> élève : .....
- nombre d'évaluations : .....

- nb lignes = ....
- nb colonnes = ....

- nb de personnes : ....
- nb de critères : ....

### B. Implémentation (modélisation informatique) :

Puisqu'on a affaire à des tableaux sans entêtes, les données sont donc sans étiquette (non nommées).

Donc d'après la page précédente, **un tableau sans entêtes est modélisé informatiquement par une liste de listes (ou une liste de tuples)**. Cette liste de listes (ou de tuples) peut être écrite sous 2 formes :

- sous la forme d'une liste classique écrite en ligne horizontalement.
- sous la forme d'une liste « tableau » écrite en colonne verticalement.

les éléments sont écrits les uns sous les autres. ⚠ ne pas oublier la « , » après chaque élément !

Véritables tableaux

N	S	I
78	83	73

Liste écrite en ligne de listes (ou de tuples)

`[ ['N', 'S', 'I'], [78, 83, 73] ]`

Liste « tableau »

`[ ['N', 'S', 'I'],  
[78, 83, 73] ]`

a	97
b	98
c	99

--

--

Vador	Mir
Ulla	Houpe

--

--



### C. Entrée simple de tables sans entêtes :

- Entrer sur Pythontutor le tableau suivant sous la forme d'une liste de listes « Table » écrite en présentation « tableau ».
- Puis afficher à l'écran cette liste « Table ».

A	True
bon	jour
(2, 'v')	65



<ul style="list-style-type: none"> <li>En Mathématique, un tableau de nombres s'appelle une matrice et peut se noter entre 2 grandes parenthèses.</li> </ul> <p><u>Exemple</u> : <math>\begin{pmatrix} 1 &amp; -2 \\ 6 &amp; 3,5 \end{pmatrix}</math> est une matrice carrée 2-2.</p> <ul style="list-style-type: none"> <li>Un système de plusieurs équations peut s'écrire matriciellement.</li> </ul> <p><u>Exemple</u> : Le système d'équations ci-dessous</p> $\begin{cases} 2x + 3y = 5 \\ 4x - 7y = -1 \end{cases}$ <p>se traduit matriciellement par</p> $\begin{pmatrix} 2 & 3 \\ 4 & -7 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 5 \\ -1 \end{pmatrix},$ <p>ce qui est facilement manipulable par une machine.</p>	<p>→ Stocker cette matrice 3-3 sous forme d'une liste « tableau » appelée Mat. Puis afficher Mat.</p> $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \longleftrightarrow$ <p>→ A la main, traduire matriciellement le système d'équations suivant :</p> $\begin{cases} 8a - 2b = 1 \\ 3a + 7b = -7 \end{cases}$ <p><math>\longleftrightarrow</math></p> <p>Puis stocker ces 3 matrices dans 3 listes appelées mat_système, mat_inconnues et mat_2ndmembre.</p>
--	--

### D. Génération automatique d'une table de données sans étiquettes :

Contrairement aux exemples précédents, les tables de données dans la vraie vie sont de gigantesques tableaux qui comportent des centaines, des milliers voire des millions de lignes-colonnes ! Quelques exemples de ces immenses tables : .....

Or ces gigantesques tableaux, il faut bien les initialiser !

Aussi surprenant soit-il, cela est fait parfois à la main ! [Des armées de travailleurs sont prêts à effectuer sur Internet des micro-tâches payées des clopinettes](#) : étiqueter des images, rentrer des fiches clients etc.

Pas question pour nous d'exploiter la misère, nous allons créer un grand tableau automatiquement ☺.

<b>Génération d'une table d'éléments sans étiquette TOUS IDENTIQUES.</b>	
<ul style="list-style-type: none"> <li>On veut par exemple générer une table composée de 3 lignes, chaque ligne comportant 4 zéros.</li> <li>Ecrire à la main d'abord la liste « Table » : <math>\longrightarrow</math></li> <li>Compléter les scripts suivants qui permettent de générer cette table :</li> </ul>	
<b>Avec 2 boucles For imbriquées.</b>	<b>Avec 1 seule boucle For.</b>
<pre>1 Table = ..... # initialisation 2 for numero_ligne in range(.....) : 3     ligne = [ ] 4     for numero_colonne in range(.....) : 5         ligne.append(.....) 6     Table.append( ligne )</pre>	<pre>1 Table = ..... # initialisation 2 for numero_ligne in range(.....) : 3     ligne = [ 0 ] * ..... 4     Table.append( ligne )</pre> <p>La ligne 3 remplace les lignes ..... de gauche.</p>
	<b>Plus fort : avec 2 compréhensions de liste imbriquées.</b>
	<pre>[ [ ..... for col in range( ..... ) ] for ligne in range( ..... ) ]</pre>

Attention ! Dans la méthode par compréhension de listes imbriquées, l'ordre d'action des boucles For se lit de droite à gauche et non de gauche à droite.

Attention ! L'écriture séduisante `[ [0]*5 ]*3` génère une table de lignes toutes liées ! Tester sur pythontutor.

➤ Ecrire à la main les listes « tableau » générées par les 4 scripts suivants. Puis vérifier sous pythontutor.

<pre> 1 Tab1 = [ ] 2 for j in range(5) : 3     ligne = [ ] 4     for k in range(2) : 5         ligne.append('a') 6     Tab1.append(ligne) </pre>		<pre> 1 Mat2 = [ ] 2 for h in range(2) : 3     a = [ ] 4     for p in range(3) : 5         a.append(h) 6     Mat2.append(a) </pre>	
Mat3 = [ [ 1 for j in range(3) ] for ligne in range(2) ]		Mat4 = [ [ a for a in range(4) ] for h in range(2) ]	

### E. Afficher à l'écran une liste de listes comme un « vrai » tableau :

Afficher une table d'éléments sans étiquette comme un « vrai » tableau.	
<p>Soit par exemple la liste de listes Table = [ [ 1 , 2 , 3 ] , [ 4 , 5 , 6 ] ] qui modélise la matrice <math>\begin{pmatrix} 1 &amp; 2 &amp; 3 \\ 4 &amp; 5 &amp; 6 \end{pmatrix}</math>.</p> <p>Si on se contente juste de print(Table), il s'affiche évidemment .....</p> <p>On veut afficher cette liste de listes à la manière d'un vrai tableau (sans les bordures) : <math>\begin{matrix} 1 &amp; 2 &amp; 3 \\ 4 &amp; 5 &amp; 6 \end{matrix}</math>.</p> <p>Compléter le script suivant qui affichera Table comme un vrai tableau à l'écran :</p>	
<pre> 1 for ligne in Table : 2     for element in ..... : 3         print ( ....., end = '\t ' ) 4     print( ) </pre>	<pre> 1 Pour chaque sous-liste (ligne) de Table, 2 prendre chaque élément de cette sous-liste, 3 l'afficher. </pre>
Pourquoi « end = '\t ' » en fin de ligne 3 ?	Pourquoi un print vide en ligne 4 ?



### F. Applications :

A l'aide de compréhensions de liste imbriquées, générer les tableaux-matrices suivants.

Puis les faire afficher comme de vrais tableaux :

- ① Un tableau tab1 de 10 lignes 20 colonnes remplis de la lettre k.
- ② Une matrice mat1 contenant les 10 tables de multiplications.

- ③ La matrice mat2 suivante :  $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$ .

- ④ Une matrice mat3 10 lignes-10 colonnes avec des 0 partout sauf des 1 sur la diagonale :  $\begin{pmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{pmatrix}$ .
- On pourra d'abord créer une matrice de 0 puis remplacer les 0 de la diagonale par des 1.

### III. DES TABLEAUX AVEC ENTETES AUX TABLES DE DONNEES.

#### A. Exemples de tableaux avec entêtes dans la vraie vie :

Tableau de moyennes en Maths d'élèves

Trimestre1	Trimestre2	Trimestre3
20	0	19
2	19	1

- Nombre d'entêtes : .....
- Nombre d'élèves : .....

Tableau de personnes

	Sexe	Age	QI
Ali Ene	?	8	1 685
Paul Isson	M	16	76

- Nombre de personnes : ....
- Nombre de critères : .....

#### B. Implémentation (modélisation informatique) :

➤ Puisqu'on a affaire à des tableaux avec entêtes, les données sont donc nommées et ont une étiquette.

Donc d'après p.2, un tableau avec entêtes est implémenté (modélisé informatiquement) soit par une liste de dictionnaires ayant les mêmes clés, soit par un dictionnaire. Cela peut s'écrire sous 2 formes :

- sous la forme classique écrite en ligne horizontalement.
- sous la forme « tableau » écrite en colonne verticalement.

Les éléments sont écrits les uns sous les autres. ⚠ Ne pas oublier la virgule après chaque élément !

Vrai tableau à entêtes colonnes

a	b	c
3	5	2
4	1	0

Liste (écrite en ligne) de dictionnaires ayant les mêmes clés

[ { 'a':3 , 'b':5 , 'c':2 } , { 'a':4 , 'b':1 , 'c':0 } ]
---

Liste « tableau »

[ { 'a':3 , 'b':5 , 'c':2 } , { 'a':4 , 'b':1 , 'c':0 } ]
---

Vrai tableau à entêtes lignes

e	4	5
k	1	2

Dictionnaire (écrit en ligne) ayant pour valeurs des listes de même longueur

{ 'e': [ 4 , 5 ] , 'k': [ 1 , 2 ] }
-------------------------------------

Dico « tableau »

{ 'e': [ 4 , 5 ] , 'k': [ 1 , 2 ] }
-------------------------------------



#### C. Entrée simple d'un tableau avec entêtes colonnes et/ou lignes :

- Entrer le tableau à entêtes-colonnes suivant sous forme d'une liste « Pays » de dictionnaires ayant les mêmes clés. Puis afficher à l'écran cette liste Pays.

Capitale	Monnaie
Paris	Euro
Hanoï	Dong



- Entrer le tableau à entêtes-lignes suivant sous la forme d'un dictionnaire de listes qu'on appellera « Pays » :

- les clés seront les noms de pays.
- les valeurs associées seront les listes [capitale , monnaie].

Puis afficher ce dictionnaire « Pays ».

Albanie	Tirana	Lek
Estonie	Talinn	Euro



- Entrer le tableau à double-entêtes suivant sous la forme d'un dictionnaire « Pays » contenant lui-même des dicos :

- les clés seront les noms de pays.
- les valeurs associées seront elles-mêmes des dictionnaires !

Puis afficher ce dictionnaire « Pays ».

	Capitale	Monnaie
Samoa	Apia	Tala
Ghana	Accra	Cedi




## D. Entrée par l'utilisateur d'un tableau avec entêtes :

### *Entrée par l'utilisateur des valeurs contenues dans un tableau à entêtes colonnes.*

- Exemple : On veut que l'utilisateur (et non le programmeur !) entre au fur et à mesure les Noms, Prénoms et Ages de 4 personnes.

Ces informations sont collectées dans un tableau à entêtes colonnes.

**On sait qu'un tableau à entêtes colonnes est implémenté par une liste de dictionnaires ayant les mêmes clés.**

- Dessiner d'abord à la main le tableau avec entêtes colonnes correspondant à cette situation : 
- Compléter le script ci-dessous qui permet de générer la liste Table de dictionnaires correspondant à cette situation :

#### 2 boucles For imbriquées

```
1 Table = .....
2 liste_criteres = [ '.....' , ..... , ..... ]
3 for numero_personne in range(.....) :
4     dico_personne = .....
5     for critere in ..... :
6         print( f'Entrez la valeur de {critere} : ' )
7         dico_personne[critere] = input( )
8     Table.append(.....)
```

#### Explications

- Lignes 1 et 2 : initialisations.
- Lignes 3 et 8 : la boucle externe gère l'ajout des dictionnaires (représentant chaque personne) à la liste finale Tableau.
- Ligne 4 : initialisation des dictionnaires.
- Lignes 5 à 7 : la boucle interne gère l'entrée par l'utilisateur des items (critère : infos) pour chaque dictionnaire.

- Remarque : La liste des critères (entêtes) peut être elle-même entrée par l'utilisateur à l'aide d'une boucle.

- Application : Générer les 2 tables suivantes puis les afficher comme de vrais tableaux (sans les bordures) :

- ① Une liste de 2 chansons avec 4 critères entrés à l'avance par le programmeur mais entrée des informations par l'utilisateur :

Titre	Artiste	Durée	Année

- ② Liste de 4 films avec 3 critères entrés par l'utilisateur et entrée des informations aussi par l'utilisateur.

**Entrée par l'utilisateur des valeurs contenues dans un tableau à entêtes lignes.**

- Exemple : On veut que l'utilisateur (et non le programmeur) rentre la liste des notes à 3 tests. —————>

Ces informations sont collectées dans ce **tableau à entêtes lignes qui sera implémenté par un dictionnaire de listes**.

- Le script ci-dessous génère ce dictionnaire de listes.

Le compléter :

Test1				
Test2				
Test3				

2 boucles For imbriquées

```

1 dico_tests = { }
2 for numero_eval in range(3) :
3     Nom_eval = input( "Entrez le nom de l'évaluation : " )
4     liste_notes = [ ]
5     for numero_note in range(4) :
6         note = int( input( 'Entrez la valeur de la note : ' ) )
7         liste_notes.append(note)
8     dico_tests[Nom_eval] = liste_notes

```

Explications

- Lignes 1 : initialisation du dictionnaire.
- Pour chaque ligne du tableau (chaque item du dictionnaire) :
- Lignes 3 et 4 : initialisation de la clé (Nom\_eval) et de la valeur (liste\_notes).
- Lignes 5 à 7 : Entrée de la liste de notes.
- Ligne 8 : ajout de l'item (Nom\_eval : liste\_notes) au dictionnaire dico\_tests.

• Application :

Générer les 2 tables suivantes puis les afficher comme des tableaux à entêtes ligne (sans les bordures) :

- ③ Un tableau de relevé des températures entrées par l'utilisateur matin et soir pendant 3 jours :

Lundi		
Mercredi		
Vendredi		

- ④ Facultatif : Un tableau à entêtes ligne de 4 personnes (entrées par l'utilisateur) avec pour chacun la liste de ses 3 aliments préférés (entrés par l'utilisateur).

**Entrée par l'utilisateur des valeurs contenues dans un tableau à double entêtes colonnes et lignes.**

- Reprenons l'exemple fin page précédente : on veut toujours que l'utilisateur entre des informations sur 2 chansons. Cette fois, les informations récoltées seront organisées sous la forme d'un tableau à doubles entêtes lignes et colonnes :

	Artiste	Durée	Année
Titre1			
Titre2			

**Ce type de tableaux à doubles entêtes est implémenté par un dictionnaire de dictionnaires.**



- ⑤ Si on a le temps !

En s'inspirant du script plus haut, implémenter ce dictionnaire (entrée des titres et infos par l'utilisateur).



## E. Un seul type d'implémentation au final !

- Dans le dernier exemple page précédente, on remarque que le tableau initialement à entêtes colonnes a été transformé en tableau à doubles entêtes lignes et colonnes. L'inverse est évidemment vrai aussi :

Titre	Artiste	Durée	Année	Style

↔

	Artiste	Durée	Année	Style
Titre1				
Titre2				

**Plus généralement, n'importe quel type de tableau à entêtes peut être transformé en n'importe quel autre type de tableau à entêtes !**

- Application : Transformer les tableaux à entêtes suivants en tableaux à entêtes colonnes puis compléter :



	Titre	Année	Style
Artiste1	a	1	e
Artiste2	b	2	f
Artiste3	c	3	g



	Pays1	10	p	5
Pays2	11	q	6	
Pays3	12	r	7	
Pays4	13	s	8	

- Les différents types de tableaux à entêtes étant tous interchangeables, on va garder évidemment le type de tableau dont l'implémentation informatique est la plus simple ! Lequel ? Entourer le bon choix :

Tableau à entêtes colonnes



Liste de dictionnaires ayant les mêmes clés

Tableau à entêtes lignes



Dictionnaire ayant pour valeurs des listes

Tableau à doubles entêtes



Dictionnaire ayant pour valeurs des dictionnaires

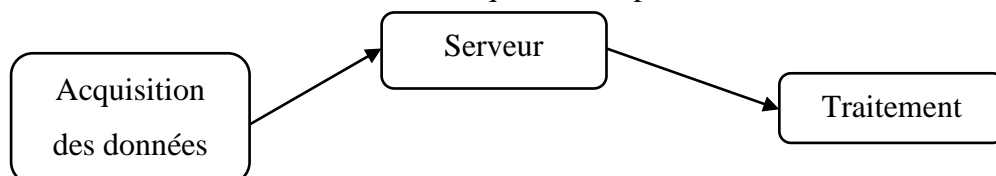
**And the winner is : .....**

Et c'est pourquoi dans toutes les bases de données, on se limite aux tables à entêtes .....

## IV. TABLE DE DONNEES ET FICHIER CSV.

En général, l'acquisition des informations ne se fait pas au même endroit que leur traitement.

Par exemple, cette acquisition peut se faire sur le web, les informations seront ensuite stockées sur un serveur. Puis le traitement se fera sur une autre machine qui devra rapatrier ces informations du serveur.



Le stockage des données sur le serveur et leur traitement ne sont en général pas faits dans le même langage. Par exemple, la base de données peut être écrite en SQL ou MongoDB alors que le traitement pourra être fait en Python ou PHP.

Le fichier de transfert doit donc être écrit dans un format compréhensible par la machine de traitement.

Pour nous, ce sera le format CSV (mais il en existe d'autres : xml par exemple).

## A. Découverte du format CSV :

Un fichier au format CSV est un simple fichier texte qui contient des données tabulées (des informations issues d'un tableau informel). Voici par exemple le contenu du fichier Préhistoire\_Informatique.csv :

```
Année,Nom,Prénom,A retenir
9ème siècle,Al Khwarizmi,,Père de l'Algorithmique
1642,Pascal,Blaise,1ère machine à calculer
1703,Leibniz,Gottfried,Formalisation du langage binaire
1801,Entreprise Jacquard,,1ère machine programmable par cartes perforées
1843,Lovelace,Ada,1er programme informatique
1844,Boole,Georges,Père de l'Algèbre binaire
1890,Entreprise Hollerith,,1ère machine à calculer électrique
```

- Que représente la première ligne ? .....
- Combien de critères (ou attributs ou descripteurs) ? .....
- Combien d'enregistrements (d'objets, de personnages, de lignes autres que celle des critères) ? .....
- Ce fichier CSV correspond à un tableau combien de lignes combien de colonnes ? .....

Compléter le tableau ci-dessous correspondant au fichier Préhistoire\_Informatique.csv :



- Dans le fichier CSV, quel caractère très important correspond aux traits de colonne du tableau ? .....

## B. Définition du format CSV :

- Un fichier **CSV (Comma Separated Values)** est un fichier texte traduisant un **tableau colonnes**.
- Les **entêtes** si elles existent sont mises **en 1ère ligne**.
- Les autres lignes s'appellent les **enregistrements** (ou objets) et sont constituées de valeurs.
- Les valeurs sont séparées par des virgules, **sans aucun espace avant ou après la virgule !**
- Si une valeur doit contenir une virgule, alors cette valeur est placée entre doubles guillemets droits " .

A partir de maintenant, la problématique est la suivante : à partir d'un programme Python, comment :

- extraire les données d'un fichier CSV.
- implémenter en Python la table de données résultant de ces données.
- exploiter cette table.

Il s'agit donc en premier lieu de lire des données dans un fichier.

Voyons donc comment plus généralement on manipule un fichier à partir d'un programme Python.

Le chapitre V qui suit est valable pour tout type de fichier, csv compris.

## V. MANIPULER UN FICHIER A PARTIR D'UN PROGRAMME.

Problématique : Manipuler à partir d'un programme Python les données contenues dans un fichier source.

La procédure sera la suivante :

- ouvrir d'abord le fichier source.
- y lire et/ou écrire des données.
- fermer le fichier source.

Mais avant cela, il faut être d'abord capable de décrire le chemin permettant d'accéder à ce fichier source.

### A. Chemin d'accès à un fichier : chemin absolu, chemin relatif.

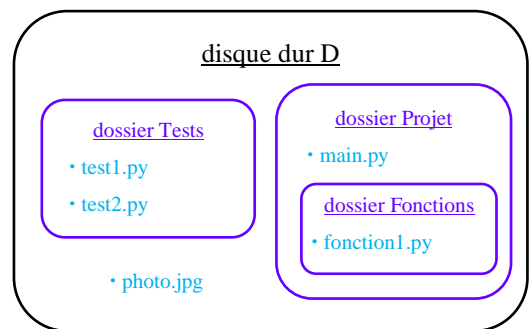
- Les fichiers sont souvent stockés dans des dossiers. Ces dossiers peuvent être eux-mêmes stockés dans des dossiers plus gros. Au final, tous ces dossiers (et fichiers) sont stockés dans une mémoire de masse non volatile : disque dur, clés usb, etc.

Quel est le nom du disque dur ci-contre ? .....

Combien de dossiers au total ? .....

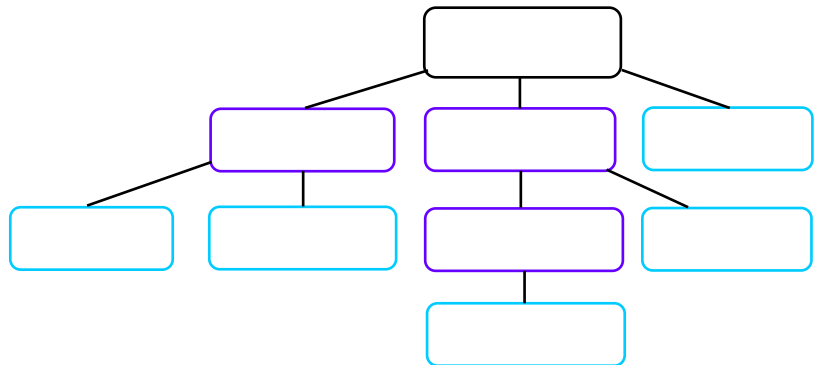
Combien de fichiers au total ? .....

Quel fichier n'est pas dans un dossier ? .....



Une représentation sous forme d'arbre est bien plus pratique et montre mieux la hiérarchie entre les dossiers et fichiers.

Compléter cet arbre de dépendances avec les noms des dossiers et fichiers précédents.



#### *Chemin absolu vers un fichier.*

- **Le chemin absolu d'accès à un fichier est le chemin qui mène de la racine de l'arborescence (le haut de l'arbre de dépendances, ici le disque dur D) jusqu'au fichier.**

Exemple : Le chemin absolu qui mène au fichier test1.py s'écrira D:/Tests/test1.py

Chemin absolu qui mène au fichier python main.py : .....

Chemin absolu qui mène au dossier Fonctions : .....

Chemin absolu qui mène au fichier fonction1.py : .....

Remarque : L'emplacement d'un fichier s'obtient en faisant clic droit sur ce fichier, propriétés.

Le chemin absolu vers un fichier est donné par : chemin donné par l'emplacement/nom du fichier



### Chemin relatif vers un fichier.

- Supposons que le fichier main.py accède au fichier fonction1.py grâce au chemin absolu.

Une autre personne participant au projet copie sur le disque dur C de son ordinateur le dossier Projet.

Problème : le fichier main.py ne peut plus accéder sur ce nouvel ordinateur à fonction1.py car le chemin absolu de fonction1.py n'est plus D:/Projet/Fonctions/fonction1.py mais .....

C'est bien dommage car le fichier fonction1.py, relativement à main.py n'a pas bougé : il est toujours dans le dossier Fonctions qui est toujours au même niveau que main.py, et ce même sur ce nouvel ordinateur.

- **On a donc inventé un autre type chemin qui part de l'emplacement du fichier appelant jusqu'à l'emplacement du fichier cherché : c'est le chemin relatif.**

On a 3 cas possibles de chemins relatifs suivant l'emplacement du fichier cherché par rapport au fichier appelant (on s'appuie toujours sur l'arbre de dépendances page précédente) :

- Cas 1 : Fichier cherché dans le même dossier que le fichier appelant.

Dans ce cas, le chemin relatif est juste le nom du fichier cherché.

Ex : Si test1.py a besoin de test2.py, le chemin relatif à écrire dans test1.py sera : test2.py.

Supposons que le dossier Fonctions contiennent en fait une autre fonction fonction2.py. fonction1.py a besoin de fonction2.py. Quel est alors le chemin relatif à utiliser dans fonction1.py : .....

- Cas 2 : Fichier cherché dans dossier interne au dossier du fichier appelant.

Ex : Le chemin relatif qui mène à fonction1.py à partir de main.py est Fonctions/fonction1.py.

En effet, on voit bien sur l'arbre que main.py et le dossier Fonctions sont tous les 2 dans le même dossier Projet. Il faut donc juste aller dans le dossier Fonctions pour accéder à fonction1.py.

- Cas 3 : Fichier cherché dans dossier externe au dossier du fichier appelant.

Ex : Le chemin relatif qui mène à test1.py à partir de fonction1.py est ../ ../Tests/test1.py.

2 petits points « ../ » signifient qu'on remonte d'un cran dans l'arborescence des fichiers (on va vers le dossier parent au-dessus). Comme il y a 2 fois écrit « ../ », on remonte donc de 2 crans dans l'arborescence, puis on redescend dans le dossier Tests pour trouver test1.py.

Ecrire le chemin relatif qui mène à test2.py à partir de fonction1.py : .....

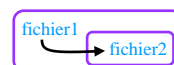
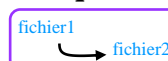
Ecrire le chemin relatif qui part de test1.py jusqu'à main.py : .....

Ecrire le chemin relatif qui mène à fonction1.py à partir de test1.py : .....

Ecrire le chemin relatif qui part de main.py jusqu'à photo.jpg : .....

**A retenir : Tout chemin relatif vers un fichier2 cherché qui se trouve :**

- dans le même dossier que le fichier1 appelant (cas 1),
- ou dans un dossier interne au dossier où se trouve le fichier1 (cas 2),



**reste valable lorsque le dossier entier est déplacé.**

**Les chemins relatifs seront donc privilégiés aux chemins absolus.**

Remarque : Windows utilise des anti-slash « \ » au lieu de slash « / » dans l'écriture des chemins. Les 2 marchent pour Windows mais **seuls les slashes « / » marchent pour Python.**

## B. Ouverture d'un fichier :

### Ouverture d'un fichier externe : fonction `open()`.

**Syntaxe :** `fichier_ouvert = open( chemin_fichier, mode = '...', encoding = '...', newline = ' ' )`

- **chemin\_fichier** : Le chemin (relatif si possible) qui mène au fichier qu'on veut ouvrir.
- **mode** : Les 3 principaux modes d'ouverture d'un fichier sont :

Mode	Signification	Que deviennent les données déjà présentes dans le fichier ?	Création d'un nouveau fichier ?
'r' (read)	• lecture simple • mode par défaut	• Données déjà présentes lues tout simplement. • Pas d'écriture de nouvelles données.	• Non. Si le fichier n'existe pas → Error.
'w' (write)	• écriture	• Δ Données déjà présentes toutes effacées ! • Ecriture de nouvelles données.	• Oui si le fichier n'existe pas.
'a' (append)	• ajout	• Données déjà présentes toutes conservées. • Ecriture de nouvelles données à la suite de celles présentes.	• Oui si le fichier n'existe pas.

- **encoding** : Préciser l'encodage pour ne pas se retrouver avec des caractères bizarres.  
'utf-8' en général (et beaucoup plus rarement 'ASCII', 'latin-9', etc.).
- **newline** : Préciser quel caractère (' ou \n ou etc.) indique la fin d'une ligne donc le début d'une autre.
- **fichier\_ouvert** : La fonction `open()` renvoie en fait un objet fichier (file en anglais). Les objets fichiers possèdent des méthodes (`.read()`, `.write()`, `.close()` etc.) qui permettent de manipuler le fichier ouvert.

#### Exemple et explications :

Soit par exemple le programme `Traitement.py` se trouvant dans un dossier `Exercices`.

- `Traitement.py` contient l'instruction suivante qui demande à ouvrir le fichier `base_données.py`

```
fic = open( Données/base_données.py , 'w' , encoding = 'utf-8' , newline=' ' )
```

`Données/base_données.py` est le chemin (relatif car ne commençant pas par une lettre de disque dur) qui indique que ce fichier `base_données.py` se trouve dans le dossier `Données` interne au dossier `Exercices` (pas de `../`). Si le fichier `base_données.py` n'existe pas, il sera créé (mode 'w').

Ce fichier `base_données.py` sera alors ouvert en mode écriture (mode 'w').

Les données déjà présentes seront effacées ('w'). Les nouvelles données seront écrites et codées en utf-8.

Les lignes dans `base_données.py` sont a priori séparées par des espaces vides (`newline = ' '`).

La manipulation du fichier `base_données.py` ouvert en écriture se fera à travers l'objet fichier nommé `fic`.

- De la même manière, expliquer cette instruction de `Traitement.py` (situé dans le dossier `Exercices`) :

```
fichier_ouvert = open( personnes.csv , 'a' , encoding = 'utf-8' , newline = '\n' )
```

## C. Lire un fichier ouvert :

### Méthode `.read()`.

**Syntaxe :** `fichier_ouvert = open( chemin_fichier , mode = 'r' , encoding = 'utf-8' , newline = ' ' )`  
`contenu = fichier_ouvert.read()`

- Le fichier est d'abord ouvert en mode lecture (mode 'r'). S'il n'existe pas → `FileNotFoundError`.
- Puis le fichier ouvert est lu jusqu'à la fin du fichier (EOF : end of file).
- **Cette lecture ne retourne qu'une unique chaîne de caractères (str) affectée à la variable contenu.**

**Exemple :** Soit le fichier csv data.csv contenant les 3 lignes de données suivantes :

data.csv	
Denrée,Prix	
Pain,1.5	
Sel,1	

Un fichier python (même dossier que data.csv) contient les instructions suivantes :

```
fichier_ouvert = open ( data.csv , mode = 'r' , encoding = 'utf-8' , newline = ' ' )
contenu = fichier_ouvert.read()
print (contenu)
```

contenu contient la chaîne 'Denrée,Prix\nPain,1.5\nSel,1'. `print(contenu)` affichera bien :

Denrée,Prix
Pain,1.5
Sel,1

Que représentent les `\n` dans la chaîne de caractères contenu ? .....



### Application :

1. Ouvrir le Bloc-Notes (Windows) ou Text Edit (Mac) ou un éditeur de texte (Notepad++ par exemple).  
Ecrire un fichier csv avec 1 ligne de 3 critères (Prénom,Nom,Age) + 2 lignes correspondant à 2 personnes.  
Enregistrer sous le nom personnes.csv. Vérifier que l'extension du fichier est bien .csv.
2. Créer un fichier python `essai1.py` qui ira lire ce fichier personnes.csv et affichera son contenu.

## D. Effacer complètement un fichier ouvert puis écrire dedans :

### Méthode `.write()` Mode 'w'.

**Syntaxe :** `fichier_ouvert = open( chemin_fichier , mode = 'w' , encoding = 'utf-8' , newline = ' ' )`  
`fichier_ouvert.write( 'Trucs à écrire' )`

- Le fichier est d'abord ouvert en mode écriture ('w'). S'il n'existe pas, le fichier est créé.
- Toutes les données contenues auparavant dans le fichier sont effacées !
- Puis 'Trucs à écrire' est écrit dans le fichier. **⚠ `.write()` n'écrit que des données de type str !**

### 3 remarques :

- Pour juste effacer le fichier, il suffit d'écrire un `.write('')` avec une chaîne de caractères vide.
- La méthode `.write( 'Truc à écrire' )` renvoie le nombre de caractères qui sont écrits dans le fichier.  
On peut récupérer ce nb dans une variable : `nb_char_écrits = fichier_ouvert.write( 'Truc à écrire' )`.
- Le module `pickle` permet d'enregistrer des objets autres que str dans un fichier.



### Application :

- Modifier `essai1.py` précédent afin de tout effacer puis écrire 2 nouveaux personnages dans personnes.csv.
- Réouvrir en mode lecture et faire afficher le contenu de personnes.csv.

## E. Ajouter de nouvelles données à un fichier ouvert :

### Méthode `.write( )` Mode 'a'.

**Syntaxe :** `fichier_ouvert = open( chemin_fichier , mode = 'a' , encoding = 'utf-8' , newline = ' ' )`  
`fichier_ouvert.write( 'Trucs à écrire' )`

- Le fichier est d'abord ouvert en mode append ('a'). S'il n'existe pas, le fichier est créé.
- Toutes les données contenues auparavant dans le fichier sont conservées !
- Puis 'Trucs à écrire' obligatoirement de type str est écrit à la suite immédiate des données déjà présentes.

**Remarque :** On doit insérer des '\n' dans la chaîne de caractères 'Trucs à écrire' pour que les ajouts se fassent sur des nouvelles lignes et non à la queue leu leu !



### Application :

- Modifier `essai1.py` précédent afin d'ajouter à la suite 2 nouveaux personnages dans `personnes.csv`.
- Réouvrir en mode lecture et faire afficher le contenu de `personnage.csv`.

## F. Fermer un fichier ouvert :

### Méthode `.close( )`.

**Syntaxe :** `fichier_ouvert.close( )`

**Ne pas oublier à la fin des manipulations de fermer le fichier, sinon on ne peut plus y accéder vu qu'il est encore ouvert !**

Le problème de fermeture d'un fichier n'est pas anodin. Imaginons qu'entre l'instruction d'ouverture d'un fichier et sa fermeture, une instruction fasse planter le programme. Alors dans ce cas le fichier reste ouvert et n'est plus accessible ! 😞 Ah ben mince, mais comment on fait alors ?

## G. Instruction composée « `with open( ) as .. : » :`

### `with open( .. ) as .. :`

#### 2 syntaxes équivalentes

`fichier_ouvert = open(chemin,mode,encoding)`  
 instruction  
 instruction  
`fichier_ouvert.close( )`

`with open(chemin,mode,encoding) as fichier_ouvert :`  

instruction  
 instruction

#### Remarques sur l'instruction « `with open( ) as .. : » :`

- ⚠ « `with open(..) as .. : »` est une instruction composée → **ne pas oublier « : » + indentation du bloc.**
- **Pas besoin de `.close( )`**, la fermeture est automatique en fin d'exécution du bloc d'instructions.

**Enorme avantage : même s'il y a plantage lors de l'exécution du bloc, `fichier_ouvert` se fermera !**

**La syntaxe « `with open(..) as .. : »` sera donc celle dorénavant utilisée.**

**Application :** Modifier votre fichier python `essai1.py` en utilisant la syntaxe « `with open( .. ) as .. : »`.



Maintenant, vous savez ouvrir, lire et afficher le contenu d'un fichier (CSV par exemple) et avez obtenu une variable contenu du genre : "Prénom,Nom,Age\nTim,Ide,23\nPat,Apouf,16\nKim,Ememesuiv,16"

Ce contenu est-il exploitable en l'état ? ..... Pourquoi ? .....

En effet, ce contenu n'est pas du tout structuré. Impossible d'y rechercher des informations selon un critère, de faire des tris etc. Il faudrait pour cela refabriquer une liste à partir de cette chaîne de caractères ! Galère.

Heureusement, un module Python facilite la gestion des fichiers csv : le bien nommé module .....

## VI. MANIPULER DES FICHIERS CSV A L'AIDE DU MODULE CSV.

### A. Lire un fichier CSV :

Grâce aux fonctions et méthodes prédéfinies du module csv, au lieu de récupérer d'un fichier csv une chaîne non structurée de caractères, on obtiendra une liste de listes ou une liste de dictionnaires.

*Générer une liste de listes à partir d'un fichier CSV : fonction reader( ) du module csv.*

Syntaxe : `import csv`

`with open( chemin_fichier , mode = 'r' , encoding = 'utf-8' , newline = ' ' ) as fichier_ouvert :`

`Table = list( csv.reader( fichier_ouvert , delimiter= ',' ) )`

- Le module csv est importé dès le début du programme obligatoirement.
- Le fichier est alors ouvert en mode lecture (mode '.....'). S'il n'existe pas → FileNotFoundError.
- csv.reader lit fichier\_ouvert. **Chaque ligne de fichier\_ouvert renverra une liste d'éléments de type str.** delimiter précise par quel caractère sont séparés les éléments dans chaque ligne de fichier\_ouvert.
- A la fin, **Table contiendra donc une liste de listes correspondant aux lignes de fichier\_ouvert.**

**Les listes internes ont toutes la même taille et sont composées d'éléments tous de type caractère 😞.**

- Exemple : Soit le fichier csv data.csv contenant les 3 lignes suivantes :

data.csv	
Denrée,Prix	
Pain,1.5	
Sel,1	

Un programme python contient les instructions suivantes :

```
import csv
```

```
with open( data.csv , mode = 'r' , encoding = 'utf-8' , newline = ' ' ) as fichier_ouvert
```

```
table = list( csv.reader( fichier_ouvert , delimiter = ',' ) )
```

La variable table contiendra la liste de listes [ ['Denrée', 'Prix'], ['Pain', '1.5'], ['Sel', '1'] ].

Ce qui est quand même déjà mieux que l'unique chaîne de caractères 'Denrée,Prix\nPain,1.5\nSel,1'.

- Remarque : Ce n'est pas encore parfait !

- table n'est pas une liste de dictionnaires mais une liste de listes (**les critères sont en 1<sup>ère</sup> liste**).
- les nombres ont été transformés en type caractère, ils ne sont pas de type ..... ou ..... !



### Application :

1. Comme ci-dessus, écrire un script essai2.py affichant une liste de listes Table à partir de personnes.csv.
2. Génération de la liste de dictionnaires liste\_dicos\_personnes à partir de la liste de listes Table :

Compléter la compréhension de listes suivante utilisant la fonction zip (livret Dictionnaires p.15) :

```
liste_dicos_personnes = [ dict ( zip ( Table[.....] , Table[.....] ) ) for k in range(..... , len(Table) ) ]
```



3. La fonction `change_en_vrais_nombres()` ci-dessous retransformera tous les âges en nombres entiers :

```
for dico in ..... :
    for clé,valeur in dico. .... :
        if valeur.isdecimal() :
            dico[.....] = int(.....)
```

On prend chaque dico dans la liste de dictionnaires,  
On prend alors (clé, valeur) de chaque item du dico.  
Si la valeur est numérique,  
alors cette valeur est convertit en nombre entier.

La fonction `read()` est tout à fait appropriée pour des données *sans* étiquettes.

Pas pour des données *avec* étiquettes ! En effet, on vient juste de voir (application p.16 question 2) qu'il faut encore générer une liste de dictionnaires à partir de la liste de listes renvoyée par la fonction `read()`.

Heureusement, il existe une fonction toute faite du module `csv` pour générer une liste de dictionnaires ayant les mêmes clés à partir de données avec étiquettes contenues dans un fichier `csv`.

### Générer une liste de dictionnaires à partir d'un fichier CSV : fonction `DictReader()` du module `csv`.

Syntaxe : `import csv`

`with open( chemin_fichier , mode = 'r' , encoding = 'utf-8' , newline = ' ' ) as fichier_ouvert :`

`Table = list( csv.DictReader( fichier_ouvert , fieldnames = [critères] , delimiter = ',' ) )`

- Le module `csv` est importé dès le début du programme obligatoirement.
- Le fichier est alors ouvert en mode lecture (mode `'r'`). S'il n'existe pas → `FileNotFoundError`.
- `csv.DictReader()` lit `fichier_ouvert`. **Chaque ligne de `fichier_ouvert` renverra un dictionnaire.**  
**`fieldnames` donne la liste des critères donc indique les clés communes. Si ce paramètre facultatif `fieldnames` est omis, les clés communes sont alors les éléments de la 1<sup>ère</sup> ligne de `fichier_ouvert`.**  
Les valeurs sont données par les éléments des autres lignes (donc sauf la 1<sup>ère</sup>) de `fichier_ouvert`.  
`delimiter` précise par quel caractère sont séparés les éléments dans chaque ligne de `fichier_ouvert`.
- A la fin, **Table contient donc une liste de dictionnaires ayant les mêmes clés.**

• Exemple : Soit le fichier `csv` `data.csv` contenant les 3 lignes suivantes :

data.csv	
Denrée,	Prix
Pain,	1.5
Sel,	1

Un programme python contient les instructions suivantes :

```
import csv
```

```
with open( data.csv , mode = 'r' , encoding = 'utf-8' , newline = ' ' ) as fichier_ouvert
```

```
liste_dicos = list( csv.DictReader( fichier_ouvert , delimiter = ',' ) )
```

Comme `fieldnames` est ici omis, les clés communes sont les éléments de la 1<sup>ère</sup> ligne de `data.csv`.

`liste_dicos` contient la liste de dictionnaires `[ {'Denrée': 'Pain', 'Prix': '1.5'}, {'Denrée': 'Sel', 'Prix': '1'} ]`.

Ce qui est bien mieux que la liste de listes `[ ['Denrée', 'Prix'], ['Pain', '1.5'], ['Sel', '1'] ]`.

• Remarque : Ce n'est pas encore parfait !

Les nombres ont tous été transformés en type `.....`, ils ne sont plus de type `int` ou `float` ! 😞



### Application :

1. Modifier `essai2.py` afin qu'il affiche une liste de dictionnaires ayant même clés comme ci-dessus.
2. Retransformer tous les âges en vrais nombres à l'aide de la fonction `change_en_vrais_nombres()`.

## B. Ecrire dans un fichier csv :

### 1. Comment écrire dans un fichier CSV une table sans entêtes :

***Ecrire dans un fichier CSV à partir de listes : fonction writer( ) du module csv.***

Syntaxe : **import csv**

**with open( chemin\_fichier , mode = '..' , encoding = 'utf-8' , newline = ' ' ) as fichier\_ouvert :**

**fichier\_ecriture = csv.writer( fichier\_ouvert , delimiter= ',' )**

**fichier\_ecriture.writerow( itérable obligatoirement )**

- Le module csv est importé dès le début du programme obligatoirement.
- Le fichier est alors ouvert en mode écriture (mettre 'w' ou 'a'). S'il n'existe pas, il est créé.  
Si le mode est 'w', toutes les données déjà présentes dans le fichier seront effacées.  
Ne pas oublier ici newline = ' ', sinon des listes vides [ ] parasites apparaîtront plus tard à la lecture.
- csv.writer met le fichier ouvert en écriture.  
delimiter précise par quel caractère seront séparés les éléments de chaque ligne.
- fichier\_ecriture.writerow( itérable ) écrit dans le fichier sur une nouvelle ligne (row veut dire ligne) tous les éléments de l'itérable (liste, tuple, dico, chaîne de caractères etc.), tels quels, séparés par le délimiteur.

- Exemple : Soit le fichier csv data.csv contenant les 3 lignes suivantes :

data.csv	
Denrée,Prix	
Pain,1.5	
Sel,1	

Un programme python contient les instructions suivantes :

```
import csv
```

```
with open( data.csv , mode = 'a' , encoding = 'utf-8' , newline = ' ' ) as fichier_ouvert
```

```
fichier_en_ecriture = csv.writer( fichier_ouvert , delimiter = ',' )
```

```
fichier_en_ecriture.writerow( [ 'Eau' , 2 ] )
```

Le fichier data.csv contiendra désormais les 4 lignes suivantes :

data.csv	
Denrée,Prix	
Pain,1.5	
Sel,1	
Eau,2	

- Remarque :

Pour écrire plusieurs lignes, on peut faire une boucle sur fichier\_ecriture.writerow( itérable ).



#### Application :

1. Créer un fichier csv vierge tables\_multiplications.csv.
2. Compléter la liste image suivante afin qu'elle génère la table de multiplication de 5 :  
[ ..... for ..... in range (.....) ]
3. Ecrire un programme remplissage\_tables.py qui permet d'écrire dans tables\_multiplications.csv la liste des tables de celle de 1 jusqu'à celle de 10. (Aide : utiliser l'exemple + une boucle + la question 2.)

## 2. Comment écrire dans un fichier CSV une table avec entêtes :

Soit une table avec entêtes implémentée par une liste de dictionnaires ayant tous les mêmes clés.

*Ecrire dans un fichier CSV à partir de dictionnaires : fonction DictWriter( ) du module csv.*

```
import csv
```

```
with open( chemin_fichier , mode = '..' , encoding = 'utf-8' , newline = ' ' ) as fichier_ouvert :
```

```
    fichier_en_ecriture = csv.DictWriter( fichier_ouvert , fieldnames = [liste_clés] , delimiter= ';' )
```

```
    fichier_en_ecriture.writeheader( )
```

```
    fichier_en_ecriture.writerow( dictionnaire ayant ses clés dans liste_clés )
```

- Le module csv est importé dès le début du programme obligatoirement.
- Le fichier est alors ouvert en mode écriture (mettre 'w' ou 'a'). S'il n'existe pas, il est créé.  
Si le mode est 'w', toutes les données déjà présentes dans le fichier seront effacées.  
Ne pas oublier ici newline = ' ', sinon des dicos vides { } parasites apparaîtront plus tard à la lecture.
- csv.DictWriter met le fichier ouvert en écriture.

**fieldnames : obligatoire ! Ce paramètre donne la liste des clés communes des dictionnaires.**

delimiter précise par quel caractère seront séparés les éléments écrits à chaque ligne.

- **fichier\_en\_ecriture.writeheader( ) écrit dans une ligne la liste des clés donnée dans fieldnames.**
- **fichier\_en\_ecriture.writerow(dico) écrit sur une nouvelle ligne (row veut dire ligne) toutes les valeurs de dico correspondantes aux clés de fieldnames, telles quelles, séparés par le délimiteur.**

- Exemple : Soit la liste suivante de dictionnaires ayant tous les mêmes clés :

```
liste_courses = [ { 'Denrée' : 'Pain' , 'Prix' : '1.5' } , { 'Denrée' : 'Sel' , 'Prix' : '1' } ].
```

Un programme python contient les instructions suivantes :

```
import csv
```

```
with open( data.csv , mode = 'w' , encoding = 'utf-8' , newline = ' ' ) as fichier_ouvert :
```

```
    fichier_en_ecriture = csv.DictWriter( fichier_ouvert , fieldnames = liste_courses[0].keys , delimiter= ';' )
```

```
    fichier_en_ecriture.writeheader( )
```

```
    fichier_en_ecriture.writerow( liste_courses[0] )
```

Le fichier data.csv contiendra alors seulement les 2 lignes suivantes :

- Remarque :

Pour écrire plusieurs lignes, on peut faire une boucle sur fichier\_en\_ecriture.writerow( dico ).

data.csv	
Denrée,Prix	
Pain,1.5	



### Application :

1. Soient les 3 listes suivantes :  

```
liste_criteres = [ 'Nom du fichier' , 'Taille en ko' ]
```

```
liste_noms = [ 'Cours.pdf' , 'Interro.pdf' , 'Exercices.pdf' ]
```

```
liste_tailles = [ 20 , 7 , 15 ]
```

Reconstituer au brouillon le tableau à entêtes colonnes correspondant à ces 3 listes.
2. A partir de ces 3 listes, écrire un programme qui génère Table la liste de dictionnaires ayant même clés.
3. Compléter ce programme afin que le fichier CSV tableau.csv soit rempli à partir de la liste Table de dictionnaires précédente.

Où en sommes-nous ? Nous savons charger les données tabulaires d'un fichier CSV vers un programme python afin d'en tirer une table de données sous la forme d'une liste de dictionnaires ayant les mêmes descripteurs (les mêmes clés) ou d'une liste de listes ayant toutes la même taille.

La dernière chose avant d'utiliser cette table est de s'assurer qu'elle soit présentable.

### C. Mise en beauté de la table de données :

#### 1. S'assurer que les nombres soient bien de type nombre :

On a vu que lors de l'importation des données d'un fichier, les nombres se retrouvaient sous format str.


En utilisant la fonction `change_en_vrais_nombres()` (p.17), on reformate tous ces nombres en int ou float.

#### 2. Eliminer les doublons de la table :

Mais comment des doublons peuvent-ils apparaître dans une table ? Ben ouais, ça arrive ! Par exemple :

- suite à une erreur de manipulation humaine lors de l'écriture de données en mode 'a' : mêmes données réécrites plusieurs fois à la suite si l'opérateur appuie plusieurs fois sur le bouton !
- suite à des enregistrements mal synchronisés de plusieurs appareils vers le cloud.
- etc.

Il faut donc une fonction qui élimine dans une table les doublons (les lignes égales c-à-d les dicos égaux).

<i>Eliminer les doublons d'une table.</i>	
<p><u>Algorithme</u></p>  <p>Soit une liste. Soit <code>liste_sansdoublons</code> une liste vide. On prend chaque élément de liste à la suite. Si cet élément n'est pas dans <code>liste_sansdoublons</code>, on l'ajoute à <code>liste_sansdoublons</code>.</p>	<p>Créer la fonction <code>elimine_doublons</code> qui reçoit liste et qui renvoie la liste sans doublons issue de liste :</p> <pre>def elimine_doublons( liste ) :</pre>
<p><u>Remarque :</u> Supposons que cette fonction <code>elimine_doublons()</code> soit écrite dans un fichier <code>utilitaires.py</code>. Pour pouvoir utiliser cette fonction dans un autre fichier, il faudra y écrire en début de programme :</p> <pre>from utilitaires import elimine_doublons</pre> <p>⚠ pas d'extension au nom du fichier (pas de <code>.py</code>) et pas de parenthèses <code>()</code> au nom de la fonction !</p>	

#### 3. Valider les données :

Il s'agit de voir si les données dans la table vérifient bien certaines conditions de validité.

Par exemple, si l'un des critères est « Mois de naissance », vérifier que les valeurs correspondantes sont bien entre 1 et 12 inclus, sinon avertir que la table contient des données invalides.

C'est un peu l'équivalent du domaine de définition en Mathématique. On parle en Informatique de domaine de validité des données. Cette étape de vérification est plus ou moins implémentée selon les besoins.

Pour finir, compléter le tableau en page de garde de ce livret.

Ai-je tout compris ? Données en table.	☹	☺	☺☺	☺☺☺
Implémenter un tableau sans entêtes.				
Implémenter un tableau avec entêtes colonnes.				
Créer soi-même un fichier CSV.				
Ouvrir et lire dans un fichier en utilisant le module csv.				
Ouvrir et écrire dans un fichier en utilisant le module csv.				
Ouvrir et lire dans un fichier en utilisant des méthodes de dictionnaire du module csv.				
Ouvrir et écrire dans un fichier en utilisant des méthodes de dictionnaire du module csv.				
Nettoyer une table : convertir les nombres au format str en vrais nombres.				
Nettoyer une table : supprimer les doublons.				

Maintenant que nos tables de données sont fin prêtes, on va pouvoir les exploiter pleinement dans le prochain livret : .....

Perle des hotlines informatiques :

- « J’essaie d’envoyer mon premier e-mail !
- Très bien, et quel est votre problème ?
- J’arrive bien à faire le « a », mais comment fait-on le cercle autour ? »

Vu sur le Bon Coin :

#### Tapis de Souris PC en Rat beige

Numéro de l'objet : PRBK



Voilà un chouette (mais long, très long à écrire) cours qui prend fin. 🐭