



Chap 6

Les fichiers

Enseignante: Mme Lamia MANSOURI



Définition d'un fichier

Collection d'informations stockées sur une mémoire de masse (non volatile, capacité plus importante que la mémoire vive)

Types de fichiers : On distingue couramment deux types de fichier, qui seront manipulés à des niveaux différents :

- Un *fichier texte* est constitué d'une séquence de caractères, permettant de stocker une chaîne de caractères sur disque. Un fichier .py contenant le code source d'un programme Python est un exemple d'un tel fichier.
- Un *fichier binaire* est constitué d'une séquence de bits, organisés en paquets de huit, appelés *octets*. Un fichier .png avec une image est un exemple d'un tel fichier.

2. Gestion des fichiers texte

3

Un fichier texte est fichier dans lequel les données sont stockées sous une forme lisible qui permet leur consultation ou leur modification à l'aide d'un éditeur de texte.

Ouverture d'un fichier

Lorsqu'on crée un objet-fichier, l'ouverture du fichier se fait grâce à la fonction intégrée **open()**.

On a plusieurs modes d'ouverture d'un fichier , à savoir :

Les modes d'ouverture	syntaxe	explication
Le mode lecture ('r')	file = open("utilisateur" , 'r')	Le mode lecture permet d'extraire les données du fichier.
Le mode écriture ('w')	file= open("utilisateur" , 'w')	'w' signifie write. Si le fichier n'existe pas, il est d'abord créé et il s'agit d'un fichier vide. Si le fichier existe déjà, celui-ci est vidé de son contenu au préalable
Le mode ajout ('a')	file = open("utilisateur" , 'a')	'a' signifie append. L'ouverture du fichier en mode ajout va créer un nouveau fichier vide si celui-ci n'existe pas. Si le fichier existe déjà, le mode 'a' permet l'écriture dans le fichier à la suite des données déjà présentes. Le contenu du fichier n'est donc pas écrasé au préalable.

Fermeture d'un fichier

4

Après avoir travaillé sur les données d'un fichier, il est nécessaire de refermer ce dernier afin qu'il puisse être disponible pour d'autres utilisations.

Cette opération s'effectue grâce à la méthode **close()**.

Ecrire dans un fichier

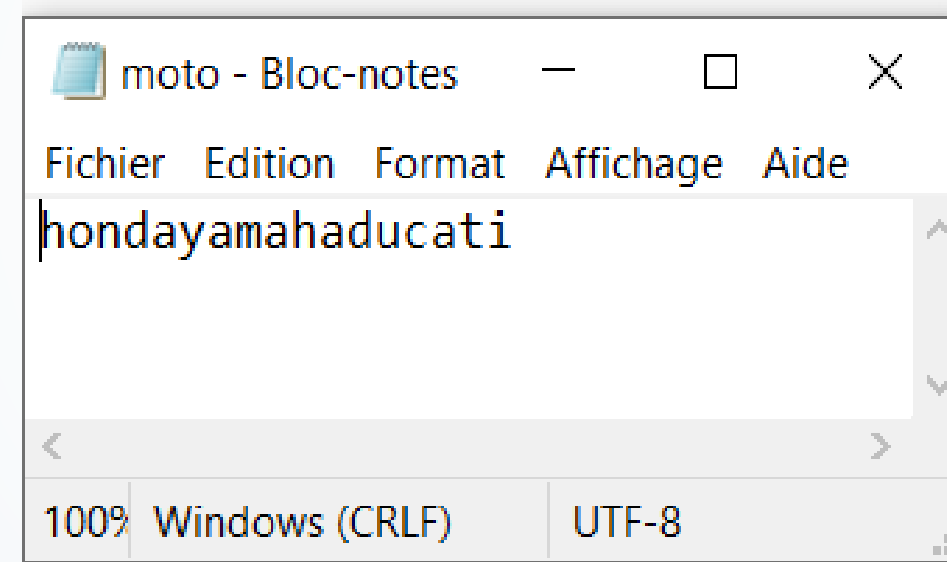
La méthode **write()**, appliquée sur un objet fichier, permet d'écrire dans un fichier.

Exemple:

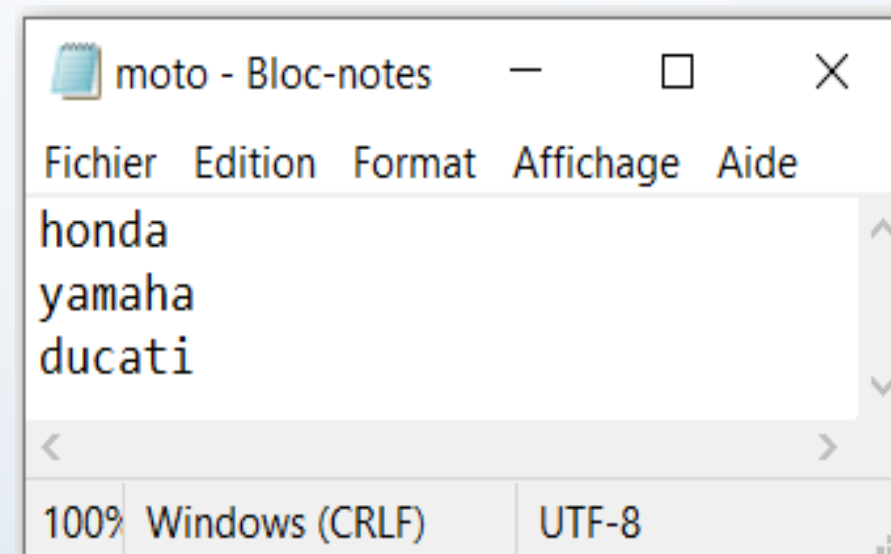
```
#ouverture en écriture
f = open("moto.txt","w")
#écriture
f.write("honda")
f.write("yamaha")
f.write("ducati")
#fermeture
f.close()
```

```
f = open("moto.txt","w")
#écriture
f.write("honda\n")
f.write("yamaha\n")
f.write("ducati")
#fermeture
f.close()
```

Contenu de moto.txt



- `open()` permet d'ouvrir un fichier en écriture avec l'option « w ». La fonction renvoie un objet de type fichier référencé par `f`
 - avec « w », le fichier est écrasé s'il existe déjà
- `write()` permet d'écrire la chaîne de caractères

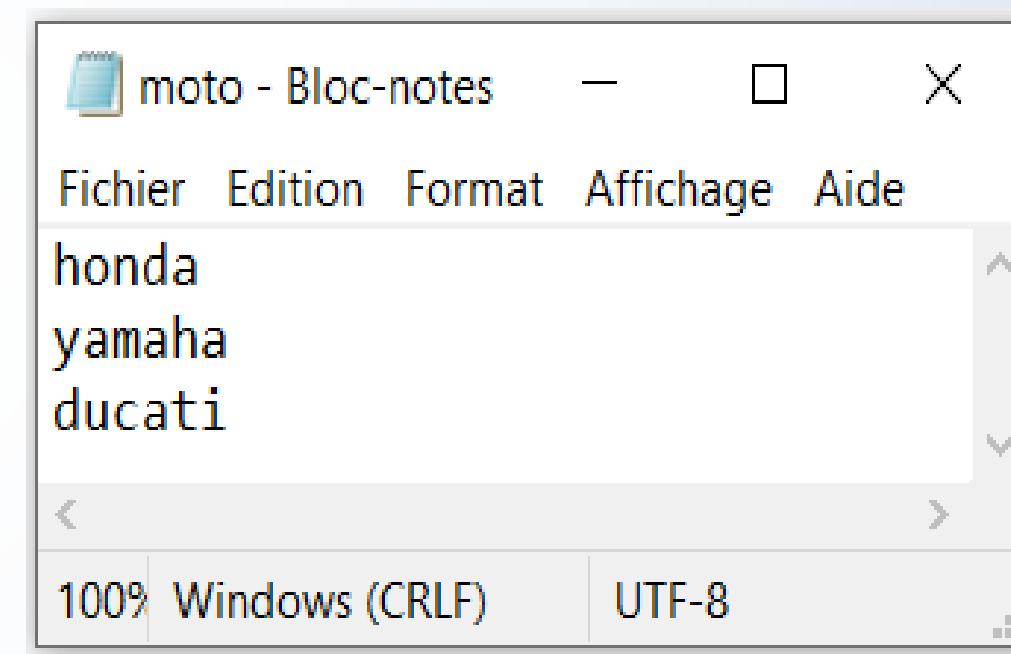


La méthode writelines()

5

writelines() permet d'écrire directement le contenu d'une **liste**. On doit insérer le caractère « \n » pour que le saut de ligne soit effectif dans le fichier.

```
#ouverture en écriture
f = open("moto.txt","w")
#liste
lst = ["honda\n","yamaha\n","ducati"]
#écriture
f.writelines(lst)
#fermeture
f.close()
```



6

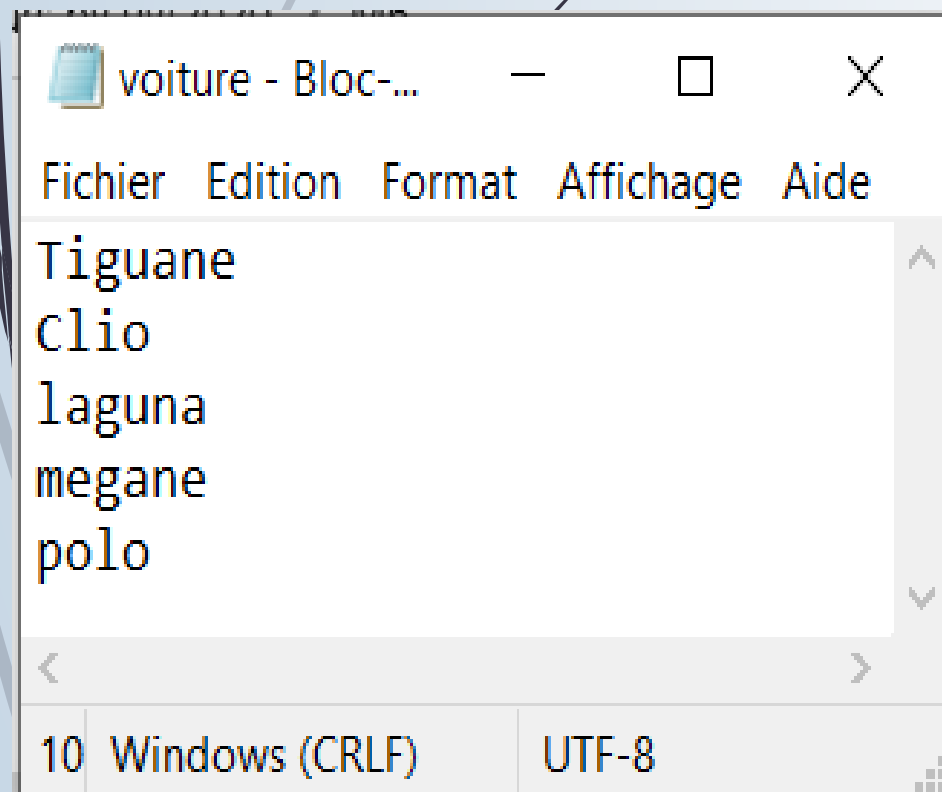
Extraction des données

Après avoir ouvert un fichier en mode lecture ('r'), on peut appliquer sur l'objet fichier diverses méthodes d'extraction de données.

■ La méthode *read()*

La méthode `read()` extrait **tout le contenu** du fichier qu'elle retourne sous la forme d'une chaîne de caractères.

Exemple :



```
#ouverture en lecture
f = open("voiture.txt","r")
#lecture
s = f.read()
#affichage
print("** contenu de s **")
print(s)
print("** fin contenu **")
#information sur s
print("type de s : ",type(s))
print("longueur de s : ", len(s))
#fermeture
f.close()
```

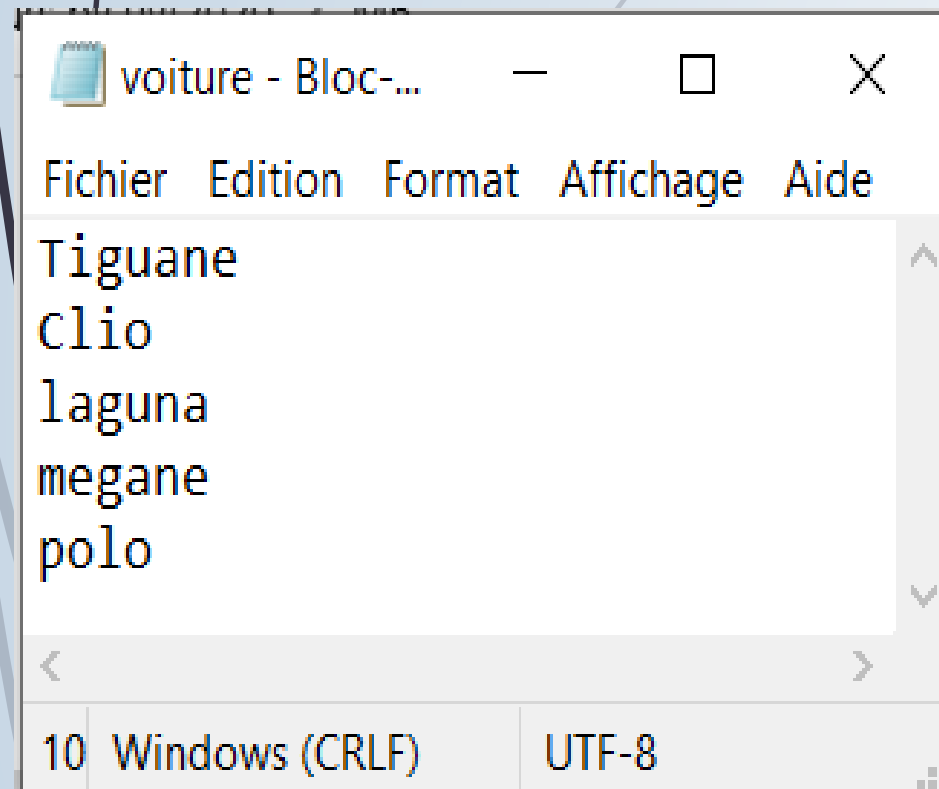
Affichage

```
** contenu de s **
Tiguan
Clio
Laguna
Megane
Polo
** fin contenu **
type de s : <class 'str'>
longueur de s : 31
```

■ *La méthode readline()*

Cette méthode ne lit qu'une seule ligne à la fois et ne prend pas en compte le caractère de fin de ligne. Elle retourne une chaîne de caractères.

Exemple :



```
#ouverture en lecture
f = open("voiture.txt","r")
#lecture ligne itérativement
while True:
    s = f.readline()
    if (s != ""):
        print(s)
    else:
        break;

#fermeture
f.close()
```

Affichage

Tiguan
Clio
laguna
megane
polo

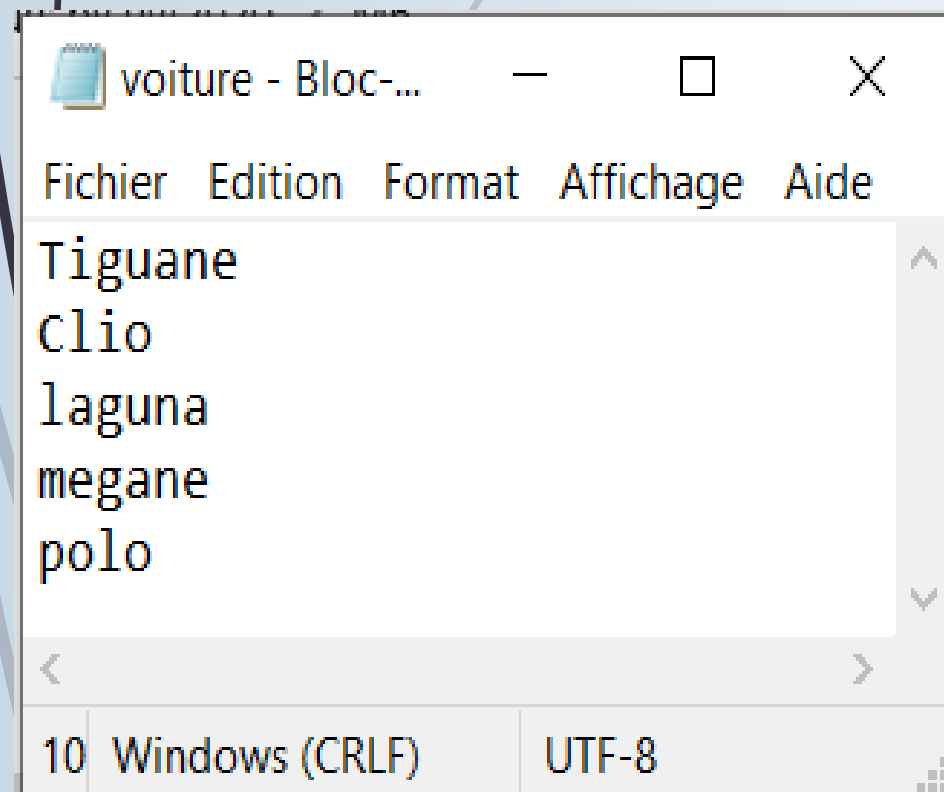
readline() :

- lit la ligne courante et place le curseur sur la ligne suivante.
- la fonction renvoie la chaîne vide lorsque nous arrivons à la fin du fichier
- (Remarque : s'il y a une ligne blanche entre 2 véhicules, readline() renvoie le caractère « \n », ce n'est pas une chaîne vide).

■ La méthode *readlines()*

La méthode `readlines()` extrait la totalité du fichier et retourne non pas une chaîne de caractères mais une **liste** sur laquelle il est possible ensuite d'effectuer une itération.

Exemple :



```
#ouverture en lecture
f = open("voiture.txt","r")
#lecture
lst = f.readlines()
#affichage
print("** contenu de lst **")
print(lst)
print("** fin contenu **")
#information sur lst
print("type de s : ",type(lst))
print("longueur de s : ", len(lst))
#fermeture
f.close()
```

Affichage

```
** contenu de lst **
['Tiguane\n', 'Clio\n', 'laguna\n', 'megane\n', 'polo']
** fin contenu **
type de s : <class 'list'>
longueur de s : 5
```

Le contenu du fichier est stocké dans une liste, une ligne = un élément.
Le caractère \n est maintenu. Il n'est pas présent sur la dernière ligne de notre fichier exemple.

Exemple 2 : On veut créer un fichier user qui contient les noms des utilisateurs et afficher son contenu

9

```
#ouverture en mode w pour remplir le fichier
```

```
nom = input("Entrez votre nom: ")
```

```
f= open('user.txt','w')
```

```
while 1:
```

```
    if nom=="":
```

```
        break
```

```
    f.write(nom+ '\n')
```

```
    nom = input("Entrez votre nom: ")
```

```
f.close()
```

```
#lecture du contenu
```

```
f = open("user.txt",'r')
```

```
print("la liste des utilisateurs :\n ")
```

```
lignes = f.readlines()
```

```
for l in lignes:
```

```
    print(l)
```

```
f.close()
```

Lecture ligne par ligne en itérant sur l'objet fichier

```
#ouverture en lecture
f = open("voiture.txt","r")
#lecture ligne itérativement
for s in f:
    print(s,len(s))
#fermeture
f.close()
```

Affichage

```
Tiguane
8
Clio
5
laguna
7
megane
7
polo 4
```

C'est la forme la plus efficace et la plus concise pour une lecture ligne à ligne.
Le caractère `\n` est présent toujours, noter la longueur de la chaîne (+1 pour toutes sauf la dernière

La syntaxe with open() as

11

Dans la mesure du possible, il est conseillé vivement d'utiliser cette syntaxe pour ouvrir vos fichiers car elle a un gros avantage : il n'y a pas besoin d'utiliser la méthode close() pour refermer le fichier.

Cette opération est effectuée automatiquement, à la sortie du bloc indenté.

with open() as fonctionne comme ceci:

```
with open("fichier", 'mode') as objet_fichier:  
    variable = objet_fichier.readlines()
```

Exemple1 :

```
with open("utilisateur", 'r') as f:  
    name = f.readlines()
```

```
with open("utilisateur", 'w') as f:  
    for i in name:  
        f.write(i.upper())
```


On peut avec l'instruction **with** ouvrir deux fichiers (ou plus) en même temps.

Exemple 2 :

```
with open("zoo.txt", "r") as fichier1, open("zoo2.txt", "w") as fichier2:  
    for ligne in fichier1:  
        fichier2.write("* " + ligne)
```

Si le fichier zoo.txt contient le texte suivant :

girafe
lion
singe

alors le contenu de zoo2.txt sera :

* girafe
* Lion
* Singe

3. Gestion des fichiers binaires

13

Python peut traiter un fichier en mode binaire, il peut lire octet par octet, ou par blocs d'octets. Un accès indicé est possible dans ce cas. Cette fonctionnalité ouvre la porte au traitement d'autres types de fichiers (ex. fichier image).

Les modes d'ouverture de fichiers binaires

En mode texte, un fichier est lu caractère par caractère. En mode binaire, un fichier est lu octet par octet.

Pour ce faire, il existe des modes d'ouverture en binaire :

- Le mode lecture : 'rb'
- Le mode écriture avec effacement préalable du contenu : 'wb'
- Le mode ajout : 'ab'

- Les chaînes de caractères peuvent facilement être écrites et lues dans un fichier. Les nombres demandent un peu plus d'effort, puisque la méthode **read()** renvoie seulement les **chaînes de caractères**, qui devront être passées vers une fonction de conversion.

Cependant, quand vous voulez sauvegarder des types de données plus complexes comme des listes, des dictionnaires, ou des instances de classe, les choses deviennent beaucoup plus compliquées.

- **La sérialisation de données est le concept de conversion de données structurées dans un format qui lui permet d'être partagé ou stocké de manière à ce que sa structure d'origine puisse être récupérée.**
- Dans certains cas, l'intention secondaire de sérialisation de données est de minimiser la taille des données sérialisé, ce qui minimise alors les exigences d'espace disque.

- Pickle est un module de python qui permet de sauvegarder **n'importe quel objet** dans un fichier et de le récupérer ultérieurement.
- Il est extrêmement pratique pour sauvegarder dans un fichier des structures de données comme les listes (type list) ou les dictionnaires (type dict).
- Un des avantages du module **pickle** est de pouvoir gérer les références circulaires : il est capable d'enregistrer et de relire une liste qui se contient elle-même, ce peut être également une liste qui en contient une autre qui contient la première...
- Le module **pickle** peut aussi gérer les classes définies par un programmeur à condition qu'elles puissent convertir leur contenu en un dictionnaire dans un sens et dans l'autre.

Pickling (sérialisation)

16

Pour sérialiser un objet on utilise la syntaxe suivante :

`pickle.dump (objet, fichier, protocole)` # Pour sérialiser un objet

ou

`pickle.dumps (objet, protocole)` # Pour sérialiser un objet en octets

où :

objet : L'objet à stocker

fichier : Le fichier ouvert qui contiendra l'objet

protocole : Le protocole utilisé pour décrocher l'objet (paramètre facultatif)

Un exemple de sauvegarde d'un dictionnaire :

```
import pickle
```

```
# création d'un dictionnaire
```

```
departement = {36:'Indre',30:'Gard',75:'Paris'}
```

```
# enregistrement du dictionnaire dans un fichier
```

```
Fichier = open('data','wb')
```

```
pickle.dump(departement,Fichier)      # sérialisation
```

```
Fichier.close()
```

Unpickling (désérialisation)

L'opération inverse est la désérialisation. La syntaxe est la suivante :

```
pickle.load (file)      #Pour désérialiser un objet  
ou  
pickle.loads (buffer)   # Pour désérialiser un objet à partir d'octets  
import pickle
```

Exemple

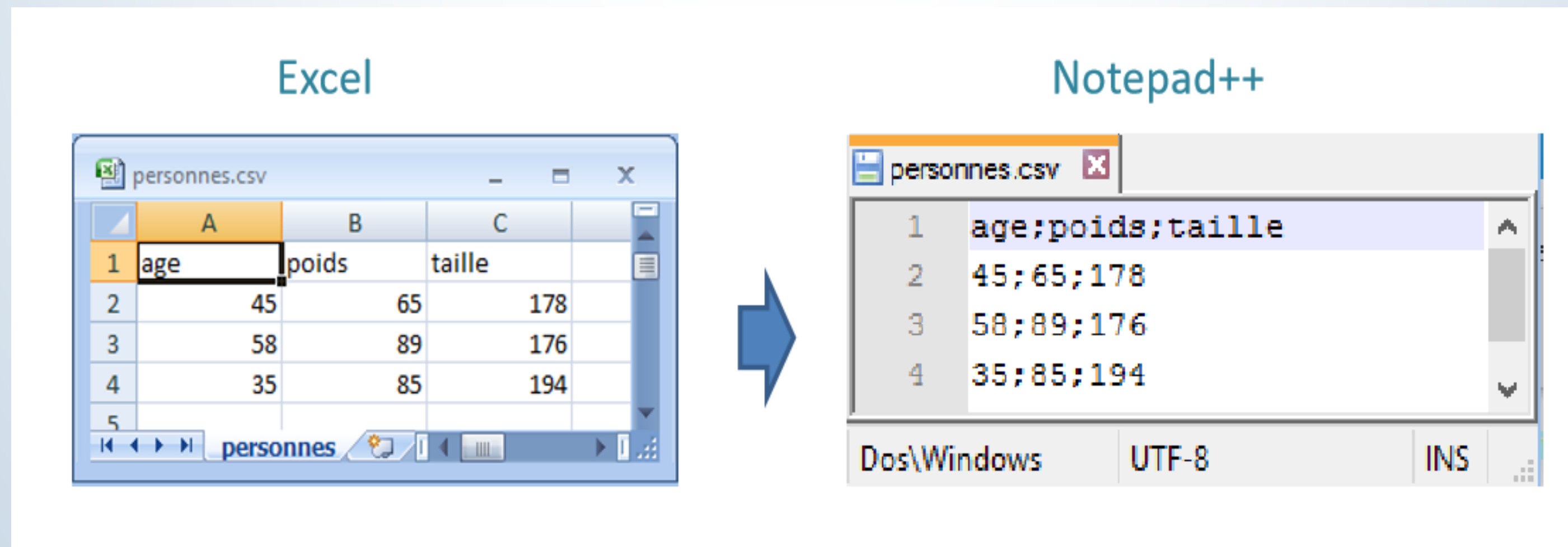
```
# récupération du dictionnaire  
Fichier = open('data','rb')  
Dept = pickle.load(Fichier)  # désérialisation  
Fichier.close()  
print(Dept)                  # affiche {75: 'Paris', 36: 'Indre', 30: 'Gard'}  
print(Dept[36])              #affiche Indre
```


Le format CSV (Comma Separated Values, valeurs séparées par des virgules) est le format le plus commun dans l'importation et l'exportation de feuilles de calculs et de bases de données.

C'est un fichier texte avec une structure tabulaire, dans lequel les données sont enregistrées ligne par ligne, et où les valeurs sont séparées par une virgule.

Le module `csv` implémente des classes pour lire et écrire des données tabulaires au format CSV.

Les objets `reader` et `writer` du module `csv` lisent et écrivent des séquences.



Remarques :

19

Le passage d'une ligne à l'autre est matérialisé par un saut de ligne ;

"," est utilisé comme séparateur de colonnes (paramétrable, ça peut être tabulation "\t" aussi souvent) ;

La première ligne joue le rôle d'en-tête de colonnes

Lecture du format csv : structure liste

```
#ouverture en lecture
f = open("personnes.csv", "r")

#importation du module csv
import csv

#lecture - utilisation du parseur csv
lecteur = csv.reader(f, delimiter=";")

#affichage - itération sur chaque ligne
for ligne in lecteur:
    print(ligne)

#fermeture du fichier
f.close()
```

Paramétrage du séparateur de colonnes avec l'option **delimiter**.

Chaque ligne est une liste.

```
['age', 'poids', 'taille']
['45', '65', '178']
['58', '89', '176']
['35', '85', '194']
```

On peut lire/écrire les données dans un dictionnaire en utilisant les classes DictReader et DictWriter.

```
#ouverture en lecture
f = open("personnes.csv", "r")


#importation du module csv
import csv

#lecture
lecteur = csv.DictReader(f, delimiter=";")

#affichage
for ligne in lecteur:
    print(ligne)

#fermeture
f.close()
```

Chaque ligne est
un **Dict**



```
{ 'poids': '65', 'age': '45', 'taille': '178' }
{ 'poids': '89', 'age': '58', 'taille': '176' }
{ 'poids': '85', 'age': '35', 'taille': '194' }
```