

## ■ Les fichiers

- Création d'un objet fichier avec open

```
f = open(filename, mode = 'r', bufsize = -1)
```

- 'r' : le fichier, qui doit déjà exister, est ouvert en lecture seule.
- 'w' : le fichier est ouvert en écriture seule. S'il existe déjà, il est écrasé ; il est créé sinon.
- 'a' : le fichier est ouvert en écriture seule. Son contenu est conservé.
- l'option '+' : le fichier est ouvert en lecture et en écriture.
- l'option 'b' : ouverture d'un fichier binaire.

## ■ Les fichiers

- Les différents modes d'ouverture
  - ✓ r, pour une ouverture en lecture (READ).
  - ✓ w, pour une ouverture en écriture (WRITE), à chaque ouverture le contenu du fichier est écrasé.
  - ✓ a, pour une ouverture en mode ajout à la fin du fichier (APPEND). Si le fichier n'existe pas python le crée
  - ✓ b, pour une ouverture en mode binaire.
  - ✓ t, pour une ouverture en mode texte.
  - ✓ x, crée un nouveau fichier et l'ouvre pour écriture

## ■ Les fichiers

- Attributs et méthodes des objets fichiers
  - `f.close()` : ferme le fichier
  - `f.read()` : lit l'ensemble du fichier et le renvoie sous forme de chaîne.
  - `f.readline()` : lit et renvoie une ligne du fichier de `f`, la fin de ligne (`\n`) incluse.
  - `f.readlines()` : lit et renvoie une liste de toutes les lignes du fichier de `f`, où chaque ligne est représentée par une chaîne se terminant par `\n`
  - `f.write(s)` : écrit la chaîne `s` dans le fichier de `f`
  - `f.writelines(lst)` : écrit la liste de chaîne `lst` dans le fichier de `f`

# Langage Python

*Toyota*  
*Peugeot*  
*Fiat*  
*Mercedes*  
*Hundai*  
*Honda*  
*BMW*

## ■ Les fichiers

- Fichier Texte

- ✓ L'accès est séquentiel, il est non structuré

- Soit le fichier voitures.txt suivant contenant les marques de voitures :  
on va écrire un programme python pour lire le contenu et l'afficher

*#ouverture en lecture*

*f = open("d:/cours assale/python/tp/voitures.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()*

*\*\* contenu de s \*\**

*Toyota*

*Peugeot*

*Fiat*

*Mercedes*

*Hundai*

*Honda*

*BMW*

*\*\* fin contenu \*\**

*type de s : <class 'str'>*

*longueur de s : 45*

# Langage Python

## ■ Les fichiers

- Fichier Texte

- ✓ L'accès est séquentiel, il est non structuré

- Le programme ci-après lit le contenu du fichier et le met dans une liste

*#ouverture en lecture*

```
f = open("d:/cours assale/python/tp/voitures.txt", "r")
```

*#lecture*

```
lst = f.readlines()
```

*#affichage*

```
print("** contenu de lst **")
```

```
print(lst)
```

```
print("** fin contenu **")
```

*#information sur lst*

```
print("type de lst : ", type(lst))
```

```
print("longueur de lst : ", len(lst))
```

*#fermeture*

```
f.close()
```

*\*\* contenu de lst \*\**

```
['Toyota\n', 'Peugeot\n', 'Fiat\n',  
'Mercedes\n', 'Hundai\n',  
'Honda\n', 'BMW']
```

*\*\* fin contenu \*\**

*type de s : <class 'list'>*

*longueur de s : 7*

- La méthode **readlines()** lit le contenu du fichier et le stocke dans une liste, une ligne = un élément de la liste
- Le caractère « \n » est maintenu



# Langage Python

## ■ Les fichiers

- Fichier Texte

- ✓ L'accès est séquentiel, il est non structuré

- Lecture ligne par ligne

- La méthode **readline()** lit une ligne du fichier

#ouverture en lecture

```
f = open("d:/cours assale/python/tp/voitures.txt", "r")
```

Toyota

- Le caractère « \n » est pris en compte d'où les sauts de lignes

#lecture ligne itérativement

```
while True:
```

```
    s = f.readline()
```

```
    if (s != ""):
```

```
        print(s)
```

```
    else:
```

```
        break;
```

Peugeot

Fiat

Mercedes

Hundai

#fermeture

```
f.close()
```

Honda



# Langage Python

## ■ Les fichiers

- Fichier Texte

- ✓ L'accès est séquentiel, il est non structuré

- Lecture ligne par ligne méthode plus efficace et concise

- Le caractère « \n » est toujours présent

#ouverture en lecture

```
f = open("d:/cours assale/python/tp/voitures.txt", "r")
```

#lecture ligne itérativement

```
for s in f:  
    print(s, len(s))
```

#fermeture

```
f.close()
```

Toyota

7

Peugeot

8

Fiat

5

Mercedes

9

Hundai

7

Honda

6

BMW 3

# Langage Python

## ■ Les fichiers

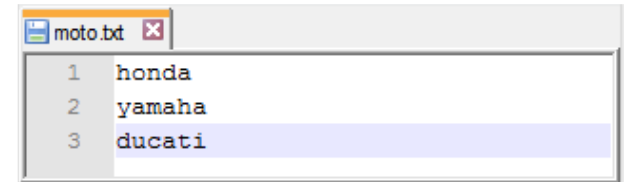
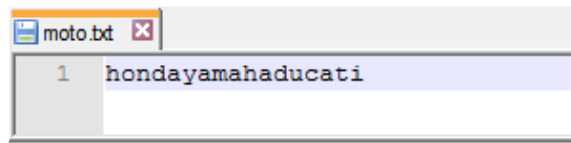
- Fichier Texte

- ✓ Écriture d'un fichier

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

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

- Le mode **w**, ouverture en écriture. Si le fichier existe il est écrasé





## ■ Les fichiers

- Fichier Texte

- ✓ Écriture d'un fichier

- Le mode **w**, ouverture en écriture. Si le fichier existe il est écrasé
- La méthode **writelines()** écrit directement le contenu d'une liste
- Insertion du caractère « **\n** » pour que le saut de ligne soit effectif

**#ouverture en écriture**

```
f = open("moto.txt", "w")
```

**#liste**

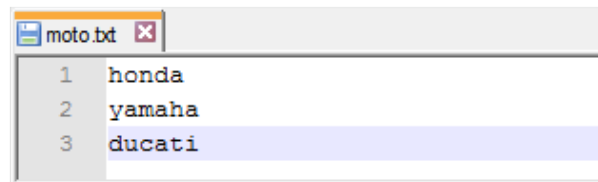
```
lst = ["honda\n", "yamaha\n", "ducati"]
```

**#écriture**

```
f.writelines(lst)
```

**#fermeture**

```
f.close()
```



# Langage Python

## ■ Les fichiers

- Fichier Texte

- ✓ Ajout d'un texte à un fichier

- Le mode **a**, ouverture en ajout.
- La méthode **write()** permet d'écrire la chaîne de caractères
- Ouverture en mode lecture/écriture est possible avec option « **r+** » mais il est très compliqué de se positionner sur une ligne pour y insérer ou modifier des informations

#ouverture en écriture

```
f = open("moto.txt", "w")
```

#ajouter un saut de ligne

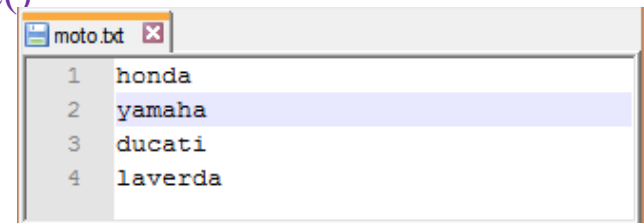
```
f.write("\n")
```

#écriture

```
f.write("laverda")
```

#fermeture

```
f.close()
```



# Langage Python

## ■ Les fichiers

- Fichier binaire

- ✓ Fichier texte traité comme un fichier binaire( fichier d'octets)

- ✓ Un accès indicé est possible

- Le mode **rb**, ouverture en lecture mode binaire.
- La méthode **read()** prend en paramètre le nombre d'octets à lire
- La méthode **decode()** transforme le tableau d'octets en chaîne de caractères
- La méthode **tell()** donne la position du curseur (début de fichier indice 0)
- Dans le résultat **b** ajouté pour dire que c'est un tableau de bytes

```
b'T'  
<class 'bytes'>  
T  
<class 'str'>  
b'o'  
position : 2
```

```
#ouverture en lecture  
f = open("d:/.../voitures.txt", "rb")  
#lire un octet  
a = f.read(1)  
print(a)  
#type de a → array de bytes  
print(type(a))  
#transformer en chaîne de caractères  
s = a.decode("utf-8")  
print(s)  
print(type(s))  
#lire une 2nde fois  
a = f.read(1)  
print(a)  
#position du curseur  
print("position : ", f.tell())
```

## ■ Les fichiers

- Fichier binaire suite

- La méthode `seek()` permet de positionner le curseur, le 1er paramètre est la position, le 2nd est la référence : 0 à partir du début du fichier, 2 à partir de la fin, 1 à partir de la position courante
- noter le `seek(-1, 2)` avec un indice négatif, comme pour les listes ou les tuples

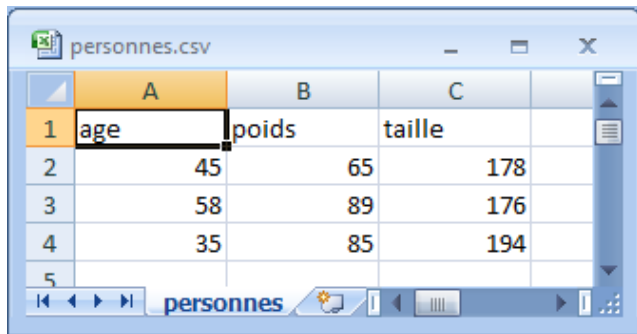
```
b'Toyota'  
longueur = 6  
b'a'  
b'd'
```

```
#positionner le curseur  
f.seek(0,0)  
#lire un bloc d'octets  
a = f.read(6)  
print(a)  
print("longueur = ", len(a))  
#aller à l'octet n°5  
#à partir du début  
f.seek(5,0)  
a = f.read(1)  
print(a)  
#lire le dernier octet  
f.seek(-1, 2)  
a = f.read(1)  
print(a)  
#fermeture  
f.close()
```

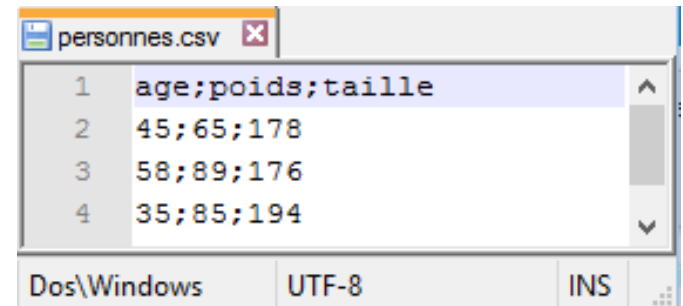
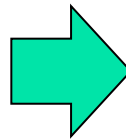
# Langage Python

## ■ Les fichiers

- Fichier CSV (Comma Separated Values)
  - ✓ Fichier texte structuré (structure tabulaire)
    - Fichier généré à partir d'un tableur (ex: Excel)
    - Format privilégié pour traitement statistique des données



	A	B	C
1	age	poids	taille
2	45	65	178
3	58	89	176
4	35	85	194



1	age;poids;taille
2	45;65;178
3	58;89;176
4	35;85;194

**Remarques :** (1) Le passage d'une ligne à l'autre est matérialisé par un saut de ligne ; (2) ";" est utilisé comme séparateur de colonnes (paramétrable, ça peut être tabulation "\t" aussi souvent) ; (3) le point décimal dépend de la langue (problème potentiel pour les conversions) ; (4) la première ligne joue le rôle d'en-tête de colonnes souvent (nom des variables en statistique).

## ■ Les fichiers

- Fichier CSV (Comma Separated Values)

- ✓ Fichier texte structuré (structure tabulaire)

- Soit à lire le fichier CSV précédent

- Il faut importer le module csv : `import csv`
- fonction `reader` du module csv lit tout le fichier et convertir en structure de liste
- On précise le séparateur de colonne avec l'option `délimiter`
- Chaque ligne est une liste
- Toutes les valeurs sont considérées comme chaînes de caractères
- Conversion automatique des chiffres est possible

`#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()`

`['age', 'poids', 'taille']`

`['45', '65', '178']`

`['58', '89', '176']`

`['35', '85', '194']`

## ■ Les fichiers

- Fichier CSV (Comma Separated Values)

- ✓ Fichier texte structuré (structure tabulaire)

- Soit à lire le fichier CSV précédent

- fonction `DictReader` du module `csv` lit tout le fichier et convertir en structure de dictionnaire
- On précise le séparateur de colonne avec l'option `délimiter`
- Chaque ligne est un dictionnaire
- On utilise les clés pour accéder aux valeurs

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

*#ouverture en lecture*

```
f = open("personnes.csv","r")
```

*#importation du module csv*

```
import csv
```

*#lecture – utilisation du parseur csv*

```
lecteur = csv.DictReader(f,délimiter=";")
```

*#affichage – itération sur chaque ligne*

```
for ligne in lecteur:
```

```
    print(ligne)
```

*#fermeture du fichier*

```
f.close()
```

## ■ Les fichiers

- Fichier JSON



## ■ Les fichiers

- Fichier XML

## ■ Travaux Pratiques N°5

- Ecrire un programme qui recopie le contenu du fichier f1.txt dans un fichier f2.txt en insérant le caractère ‘\*’ au début de chaque ligne
- Soit un fichier csv1 au format csv qui contient âge, poids et taille d’un certain nombre de personnes. Les informations sont séparées par des virgules ‘,’. Un fichier csv2 qui contient les noms des personnes. Fusionner les 2 fichiers en un fichier csv3 qui contiendra les nom, âge, taille et poids des personnes. Les informations seront séparées par des points virgules ‘;’

## ■ Travaux Pratiques N°5

- Ecrire un programme qui lit une liste de n (saisi au clavier) étudiants (on saisira le matricule, le nom, le prénom et l'âge de l'étudiant) et qui les enregistre dans un fichier texte au format csv avec la virgule ',' comme séparateur

## ■ Gestion des fichiers et dossiers

- Module os
  - ✓ Bibliothèque dédiée à la gestion des fichiers et des dossiers
  - ✓ On l'importe par : `import os`
- Chemin d'un fichier/dossier
  - ✓ Chemin absolu : part de la racine du système de fichiers
  - ✓ Chemin relatif : part du dossier en cours de lecture
  - ✓ Exemples : si on se trouve dans le dossier `/home/louis`, un fichier nommé *projets.txt* dans un dossier *scripts* aura comme :
    - Chemin absolu : `/home/louis/scripts/projets.txt`
    - Chemin relatif : `scripts/projets.txt`

## ■ Gestion des fichiers et dossiers

- Manipuler les chemins
  - ✓ Chemin (ou `path`) est une chaîne de caractères
  - ✓ Il existe des méthodes du module (`os.path`) pour la manipuler :

`abspath(path)` → Retourne un chemin absolu

`basename(p)` → Retourne le dernier élément d'un chemin

`commonprefix(list)` → Retourne le chemin commun le plus long d'une liste de chemins

`dirname(p)` → Retourne le dossier parent de l'élément

`exists(path)` → Teste si un chemin existe

`getatime(filename)` → Retourne la date du dernier accès au fichier [`os.stat()`]

`getctime(filename)` → Retourne la date du dernier changement de métadonnées du fichier

`getmtime(filename)` → Retourne la date de la dernière modification du fichier

`getsize(filename)` → Retourne la taille d'un fichier (en octets)

`isabs(s)` → Teste si un chemin est absolu

## ■ Gestion des fichiers et dossiers

- Manipuler les chemins

✓ Quelques méthodes :

`isdir(s)` → Teste si le chemin est un dossier

`isfile(path)` → Teste si le chemin est un fichier régulier

`islink(path)` → Teste si le chemin est un lien symbolique

`ismount(path)` → Teste si le chemin est un point de montage

`join(path, s)` → Ajoute un élément au chemin passé en paramètre

`normcase(s)` → Normalise la casse d'un chemin

`normpath(path)` → Normalise le chemin, élimine les doubles barres obliques, etc.

`realpath(filename)` → Retourne le chemin canonique du nom de fichier spécifié (élimine les liens symboliques)

`samefile(f1, f2)` → Teste si deux chemins font référence au même fichier réel

`sameopenfile(f1, f2)` → Teste si deux objets de fichiers ouverts font référence au même fichier

`split(p)` → Fractionne un chemin d'accès. Retourne un tuple

## ■ Gestion des fichiers et dossiers

- Manipuler les chemins

✓ Quelques exemples d'utilisation de méthodes :

```
>>> import os
>>> path = "/home/olivier/scripts/cgi-bin/action.py"
>>> os.path.dirname(path)
'/home/olivier/scripts/cgi-bin'
>>> os.path.basename(path)
'action.py'
>>> os.path.join(path, "func")
'/home/olivier/scripts/cgi-bin/action.py/func'
>>> os.path.split(path)
('/home/olivier/scripts/cgi-bin', 'action.py')
>>> os.path.abspath(".")
'/home/olivier'
```

## ■ Gestion des fichiers et dossiers

- Les éléments d'un dossier
  - ✓ La méthode `listdir`
    - Récupère dans une liste tous les éléments d'un dossier y compris dossiers et fichiers cachés
    - Exemple :
      - ```
>>> os.listdir("/home/olivier")  
['.bash_history', 'Images', 'script.py']
```
  - ✓ La méthode `walk` (récupère récursivement)
    - Récupère éléments d'un dossier ainsi que ses dossiers enfants
      - ```
folder_path = "/mnt/box/files"  
for path, dirs, files in os.walk(folder_path):  
    for filename in files:  
        print(filename)
```





# Langage Python

## ■ Gestion des fichiers et dossiers

- Les éléments d'un dossier
  - ✓ Recherche d'éléments par motif
    - On utilise les caractères suivants :
      - \* → n'importe quelle séquence de caractères
      - ? → n'importe quel caractère
      - [] → n'importe quel caractère listé entre les crochets
    - On importe la bibliothèque glob : `import glob`
    - Quelques méthodes :
      - `glob.glob(motif)` → Liste les dossiers et les fichiers
      - `glob.iglob(motif)` → Idem que glob mais retourne un itérateur
      - Exemple : `>>> glob.glob("/home/olivier/scripts/*.txt")`  
`['/home/olivier/scripts/data.txt']`

## ■ Gestion des fichiers et dossiers

- Les éléments d'un dossier
  - ✓ Manipuler les éléments
    - Quelques méthodes pour manipuler des éléments d'un dossier :
      - `os.makedirs(path)` → Créer récursivement tous les dossiers d'un path si ceux-ci n'existent pas
      - `os.mkdir(path)` → Créer le dernier dossier d'un path. Si un des dossiers n'existe pas une erreur est retournée
      - `os.remove(path)` → Supprime le fichier / dossier indiqué
      - `os.rename(old, new)` → Renomme le fichier / dossier indiqué