

## Chapitre 4

# Programmation Python

## Chapitre 4 : Gestion des fichiers txt/csv en python

- ☐ Introduction
- ☐ Manipulation des fichiers
- ☐ Opérations sur les fichiers textes
- ☐ Les fichiers CSV
- ☐ Interaction avec le système d'exploitation (Module os)

# Introduction

- ❖ **Définition** : un fichier est un ensemble d'informations numériques réunies sous un même nom et enregistrées sur un support physique (disque dur, clé USB, CD, DVD, ...).
- ❖ Un fichier est désigné par :
  - son **nom** : **nom\_Fichier.extension** (e.g., **texte1.txt**, **image.jpg** et **TP1.py**)
  - son **emplacement** ou chemin d'accès. On distingue :
    - ✓ *Chemin absolu* : donne la position d'un fichier à partir de la **racine** du disque.  
Exemple : **C:**\répertoire1\sousrépertoire\
    - ✓ *Chemin relatif* : part du dossier en cours de lecture (répertoire courant).  
Exemple : Notre dossier de travail est 'sousrépertoire' (si C:\répertoire1 est le *répertoire courant*)
- ❖ Un fichier peut être utilisé pour un programme comme entrée et/ou sortie (i.e., lu et/ou créé ou modifié).

# Introduction

---

## ❖ Objectifs de ce chapitre :

**Depuis un programme Python**, on souhaite manipuler des données sauvegardées dans un fichier en particulier :

- ouvrir un fichier ;
- lire le contenu d'un fichier ;
- modifier le contenu d'un fichier ;
- fermer un fichier.

## Chapitre 4 : Gestion des fichiers txt/csv en python

☐ Introduction

### ☐ Manipulation des fichiers

☐ Ouverture d'un fichier

☐ Fermeture d'un fichier

☐ Opérations sur les fichiers textes

☐ Les fichiers CSV

☐ Interaction avec le système d'exploitation (Module os)

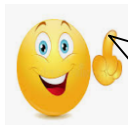
# Manipulation des fichiers



## Activité :

- Créez avec un éditeur de textes un fichier avec le nom **fruits.txt** dans le dossier **Exercices** de **C:\Python38**
- Ce fichier contient une liste de fruits par exemple :

```
Banane  
Kiwi  
Pomme  
Poire  
Orange  
Fraise
```



- Vous pouvez utiliser Bloc-notes comme éditeur de textes :  
« Démarrer → Accessoires → Bloc-notes »
- On peut l'ouvrir puis y écrire et enregistrer les données saisies.

# Manipulation des fichiers (suite)



Avant de se servir d'un **fichier**, il est nécessaire de l'**ouvrir**, puis le **fermer** après son utilisation.



## Ouverture d'un fichier

En Python, la fonction **open** permet d'ouvrir un fichier.

**Syntaxe :** `f = open("chemin\nom_fichier", mode='r' )`

**Exemple :** `f = open('fruits.txt', 'r')`

- ❖ La fonction **open** renvoie un *objet fichier* **f** de type **file** (fichier logique) manipulé par le programmeur.
- ❖ Les **paramètres** de la fonction **open** sont :
  - Nom du fichier (exemple : 'fruits.txt')
  - Mode d'accès 'mode' (exemple : 'r')

# Manipulation des fichiers (suite)



## Mode d'ouverture d'un fichier

- L'argument 'mode' est optionnel ; sa valeur par défaut est "r".

Mode	Description
<b>r</b>	Lecture (read) - Ouverture en mode <b>read</b> - Le fichier doit exister, sinon un message d'erreur est retourné.
<b>w</b>	Écriture (write) - Ouverture en mode <b>write</b> - Crée un nouveau fichier ou écrase le fichier existant
<b>a</b>	Ajout (append) - Ouverture en mode <b>append</b> - Ajoute du contenu à la fin du fichier ou crée un nouveau fichier
<b>r+</b>	Lecture/Écriture (read/write) - Ouverture en mode lecture/écriture - Le fichier doit exister, positionné au début du fichier
<b>rb</b>	Ouvrir le fichier en mode binaire : Lecture binaire (read binary) - Le fichier doit exister
<b>wb</b>	Écriture binaire (write binary)
<b>ab</b>	Ajout binaire (append binary)



# Manipulation des fichiers (suite)



## Fermeture d'un fichier

Une fois utilisé, on doit fermer le fichier en appelant la fonction **close** :

Syntaxe :

```
f.close()
```

Exemple :

```
# ouvrir et fermer un fichier
try :
    f = open ("test.txt", "r")
    # opérations sur le fichier
    f.close() # N'oubliez pas les () pour close!
except :
    print("fichier introuvable !")
```

## Chapitre 4 : Gestion des fichiers txt/csv en python

☐ Introduction

☐ Manipulation des fichiers

☐ **Opérations sur les fichiers textes**

☐ Création/écriture dans un fichier texte

☐ Lecture depuis un fichier texte

# Opérations sur les fichiers textes

---

Plusieurs opérations peuvent être réalisées avec des fichiers textes.

- On peut **créer de nouveaux fichiers** et y écrire des informations textuelles.
- On peut **modifier** le contenu d'un fichier existant.
- On peut **lire** des fichiers étant déjà créés (i.e., existants).

## Chapitre 4 : Gestion des fichiers txt/csv en python

- ☐ Introduction

- ☐ Manipulation des fichiers

- ☐ **Opérations sur les fichiers textes**

  - ☐ **Création/écriture dans un fichier texte**

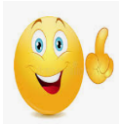
  - ☐ Lecture depuis un fichier texte

# Opérations sur les fichiers textes



## Création/écriture dans un fichier texte

- Pour écrire dans un fichier texte (nouveau ou existant), on doit **l'ouvrir en mode écriture** avec l'un des indicateurs suivants : **"w"** ou **"a"**.
  - La fonction **write** permet d'écrire dans le fichier texte ouvert ou créé.
  - Cette fonction prend en argument **obligatoirement** une *chaîne de caractères*.
  - Rappelons que la fonction **str(obj)** permet de convertir l'objet donné en une chaîne de caractères.
- 
- La fonction **write** permet de concaténer et écrire les chaînes de caractères passées en paramètres sans espaces ni saut de lignes.
  - Pour cela, il faut ajouter le séparateur **"\n"** afin d'ajouter le retour à la ligne (le séparateur **"\t"** permet d'insérer un espace de tabulation).



# Opérations sur les fichiers textes (suite)



## Exemple :

```
# créer un nouveau fichier
>>> fic1 = open ("fruits.txt", "w")

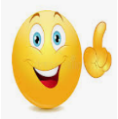
# y écrire un texte
>>> fic1.write ("Bonjour ! \n Ce fichier contient une liste de fruits. ")

# Le-fermer pour valider son contenu
>>> fic1.close()

# ouvrir un fichier existant et le modifier
>>> fic1 = open ("fruits.txt", "a")

# y ajouter un texte puis le fermer
>>> fic1.write ("\n #Ceci est un test \t Chapitre9")
>>> fic1.close()
```

# Opérations sur les fichiers textes (suite)



- Pour voir le résultat de l'exécution des commandes ci-dessus, vous devez aller à votre explorateur `WINDOWS`, et dans le dossier courant '`C:\Python38\Exercices`', vous trouverez le fichier texte que vous avez créé et qui s'appelle 'fruits.txt'.
- Ouvrez ce fichier avec l'outil Bloc-notes, vous trouverez l'écran de la figure ci-après :

A screenshot of a Windows Notepad application window. The title bar reads "fruits.txt - Bloc-notes". The menu bar includes "Fichier", "Edition", "Format", "Affichage", and "Aide". The text area contains the following content:

```
Bonjour !  
Ce fichier contient une liste de fruits.  
  
#Ceci est un test      Chapitre9|
```

The cursor is positioned at the end of the last line.

**Attention !** Si vous oubliez `fic1.close()`, le fichier est créé mais il est vide, l'ajout de données n'est validé qu'avec `fic1.close()`.

# Opérations sur les fichiers textes (suite)



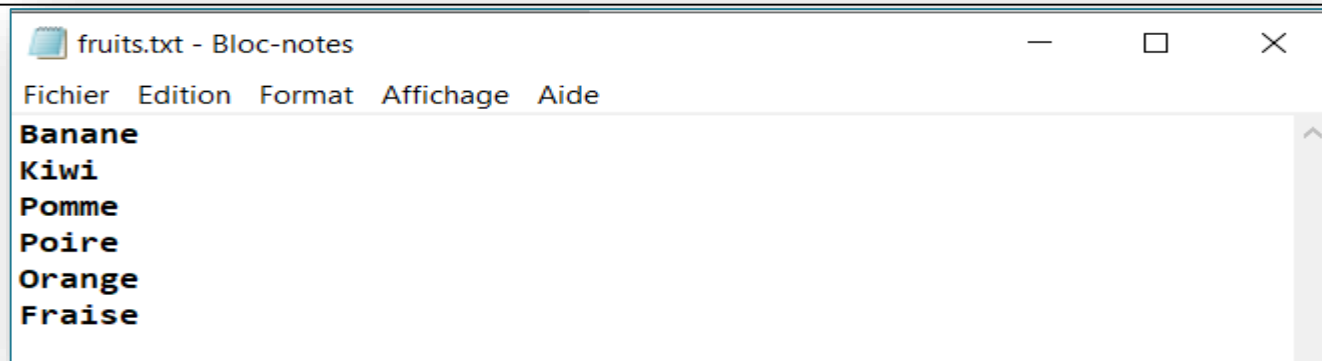
## Création/écriture dans un fichier texte (suite)

On peut aussi utiliser la fonction **writelines** permettant d'écrire le contenu d'une liste dans un fichier texte.



## Exemple :

```
>>> L = ["Banane\n", "Kiwi\n", "Pomme\n", "Poire\n", "Orange\n", "Fraise"]
>>> f = open("fruits.txt", "w")
>>> f.writelines(L)
>>> f.close()
```





# Opérations sur les fichiers textes (suite)



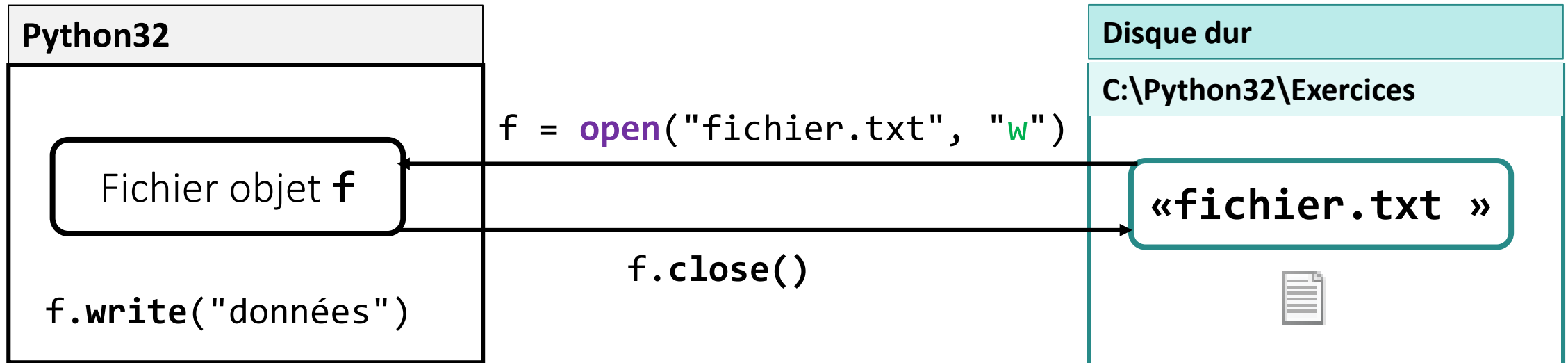
## Ecriture dans un fichier texte

	Commande	Rôle
<b>Ouverture</b> du fichier <code>f</code>	<code>f = open("fichier.txt", "w")</code>	Ouvre <code>f</code> en <b>mode ÉCRITURE</b>
<b>Ecriture</b> sur <code>f</code>	<code>f.write('données')</code>	Ecrit la chaîne de caractères ' <b>données</b> ' dans " <b>fichier.txt</b> " <ul style="list-style-type: none"><li>○ Si "<b>fichier.txt</b>" existe alors son contenu entier sera effacé et le programme Python y écrit les nouvelles données ;</li><li>○ Sinon "<b>fichier.txt</b>" est créé et le programme Python y écrit les nouvelles données.</li></ul>
<b>Fermeture</b> du fichier <code>f</code>	<code>f.close()</code>	Ferme <code>f</code>

# Opérations sur les fichiers textes (suite)



## Ecriture dans un fichier texte



# Opérations sur les fichiers textes (suite)



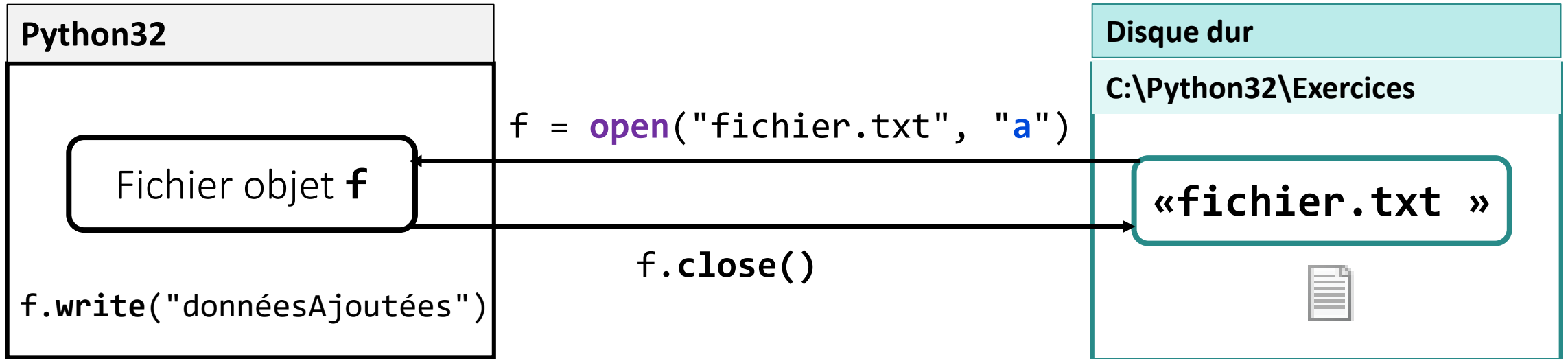
## Ajouter des données à un fichier texte

	Commande	Rôle
<b>Ouverture</b> du fichier <code>f</code>	<code>f = open("fichier.txt", "a")</code>	Ouvre <code>f</code> en <b>mode AJOUT</b> <ul style="list-style-type: none"><li>Le curseur est alors placé à la fin de la chaîne de caractères contenue dans le fichier.</li></ul>
<b>Ecriture</b> sur <code>f</code>	<code>f.write('donnéesAjoutées!')</code>	Ajoute sur <code>f</code> la chaîne de caractères <b>'donnéesAjoutées !'</b> <ul style="list-style-type: none"><li>Si <code>"fichier.txt"</code> n'existait pas alors il sera créé et on y écrit tout simplement <b>'donnéesAjoutées !'</b></li></ul>
<b>Fermeture</b> du fichier <code>f</code>	<code>f.close()</code>	Ferme <code>f</code>

# Opérations sur les fichiers textes (suite)



## Ajouter des données à un fichier texte



# Opérations sur les fichiers textes (suite)



## Activité :

1. On souhaite écrire un script Python permettant d'écrire les 10 premiers multiples de 2 dans un fichier texte '**multiple2.txt**', placé dans le répertoire courant.

Commencez le fichier par la phrase "*Liste des multiples de 2 :*" et chaque valeur sera placée dans une *nouvelle ligne*.

# Opérations sur les fichiers textes (suite)



## Activité (suite):

Résultat après exécution du code

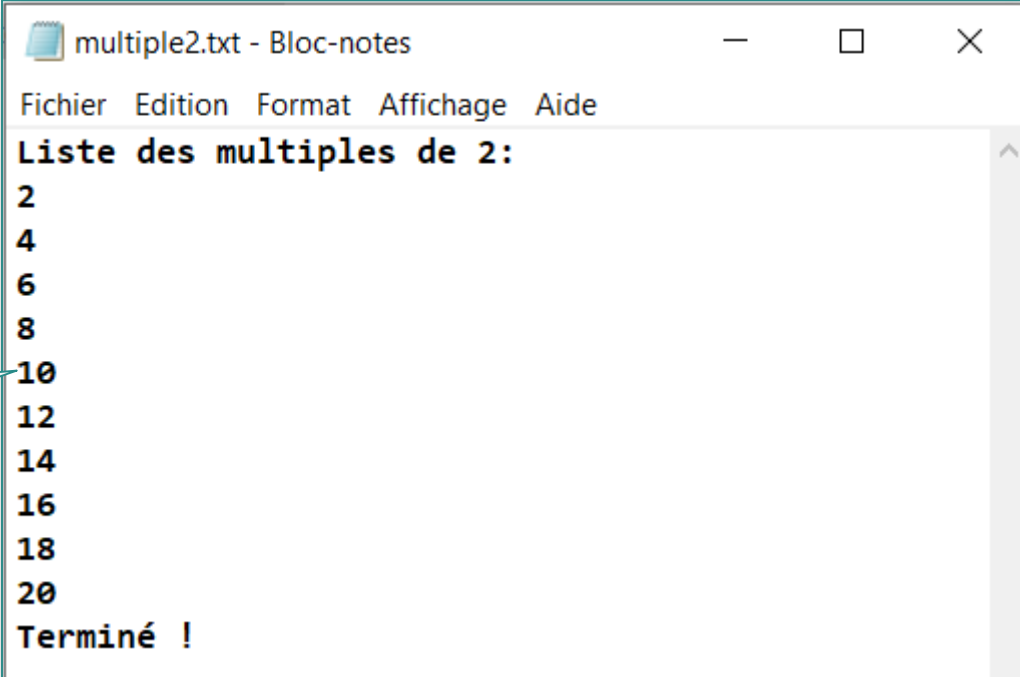
```
multiple2.txt - Bloc-notes
Fichier Edition Format Affichage Aide
Liste des multiples de 2:
2
4
6
8
10
12
14
16
18
20
```

# Opérations sur les fichiers textes (suite)

## activité **Activité (suite):**

2. On souhaite **ajouter** la chaîne « **Terminé!** » à notre fichier '**multiple2.txt**'.

Résultat après exécution du code 2



```
multiple2.txt - Bloc-notes
Fichier Edition Format Affichage Aide
Liste des multiples de 2:
2
4
6
8
10
12
14
16
18
20
Terminé !
```

## Chapitre 4 : Gestion des fichiers txt/csv en python

- ☐ Introduction

- ☐ Manipulation des fichiers

- ☐ **Opérations sur les fichiers textes**

  - ☐ Création/écriture dans un fichier texte

  - ☒ **Lecture depuis un fichier texte**



# Opérations sur les fichiers textes (suite)



## Lecture depuis un fichier texte

- Pour lire le contenu d'un fichier texte existant, on doit tout d'abord l'**ouvrir** en *mode lecture* (i.e., avec l'indicateur "r").
- La fonction **read** permet de lire le contenu du fichier texte ouvert.

## Méthodes de lecture d'un fichier

- Méthode 1 : Il est possible de lire *tout le contenu* d'un fichier **f** : **f.read()**
- Méthode 2 : lire *un nombre précis de caractères* : **f.read(nbreCaractere)**
- Méthode 3 : lire une seule ligne (celle en cours) : **f.readline()**
- Méthode 4 : retourner une liste contenant toutes les lignes du fichier texte ouvert :  
**f.readlines()**

# Opérations sur les fichiers textes (suite)



## Lecture depuis un fichier texte (suite)

- Méthode 1 : lire *tout le contenu* d'un fichier **f** : **f.read()**

### Exemple :

```
>>> f = open("fruits.txt", "r")
>>> ch = f.read()
>>> ch
'Banane\nKiwi\nPomme\nPoire\nOrange\nFraise'
>>> ch = f.read()
>>> ch
''
>>> f.close()
```

*# Après l'appel de la fonction read(), le curseur est placé à la fin du texte dans le fichier*

*# Un autre appel à f.read() renvoie la chaîne vide ''*

# Opérations sur les fichiers textes (suite)



## Lecture depuis un fichier texte (suite)

- Méthode 2 : lire *un nombre précis de caractères* : `f.read(nombreCaractere)`

### Exemple :

```
>>> f = open("fruits.txt", "r")
>>> ch7 = f.read(7)                                # Après l'appel de la fonction read(7), le curseur est placé après les
>>> ch7                                              7 premiers caractères.
'Banane\n'

>>> ch = f.read(7)                                # Un autre appel à f.read(7) permet de lire les 7 caractères suivants
>>> ch
'Kiwi\nPo'

>>> f.close()
```

# Opérations sur les fichiers textes (suite)



## Lecture depuis un fichier texte (suite)

Parfois, on a besoin d'appliquer des fonctions sur les chaînes de caractères comme :

- **rstrip()** qui enlève le caractère de saut de ligne à la fin d'une chaîne de caractères.
- **split()** qui retourne une **liste** contenant les chaînes de caractères séparées selon les espaces, sinon, on précisera le séparateur en argument (e.g., **ch.split(".")**).

Exemple :

```
>>> f = open("fruits.txt", "r")
>>> ch = f.read(7)
>>> ch
'Banane\n'
>>> ch.rstrip()
'Banane'
>>> ch = f.read(10)
>>> ch
'Kiwi\nPomme'
>>> ch.split("\n")
['Kiwi', 'Pomme']
```

```
>>> f.close()
```

# Opérations sur les fichiers textes (suite)



## Lecture depuis un fichier texte (suite)

- Méthode 3 : lire une seule ligne (celle en cours) : **f.readline()**

Exemple :

```
>>> f = open("fruits.txt", "r")
>>> ligneEnCours = f.readline()
>>> print(ligneEnCours)
Banane
>>> type(ligneEnCours)
<class 'str'>
>>> ligneEnCours = f.readline()
>>> print(ligneEnCours)
Kiwi
>>> f.close()
```

*# La fonction **readline()** renvoie une chaîne correspondante à la ligne courante et passe le curseur, initialement placé au début du fichier, dans la ligne suivante.*

*# Un autre appel à **f.readline()** permet de lire la ligne suivante*

# Opérations sur les fichiers textes (suite)



## Lecture depuis un fichier texte (suite)

- Méthode 4 : retourner une liste contenant toutes les lignes du fichier texte ouvert : `f.readlines()`

Exemple :

```
>>> f = open("fruits.txt", "r")
>>> listFruit = f.readlines()
>>> print(listFruit)
['Banane\n', 'Kiwi\n', 'Pomme\n', 'Poire\n', 'Orange\n', 'Fraise']
>>> type(listFruit)
<class 'list'>
>>> f.close()
```

# Opérations sur les fichiers textes (suite)



## Lecture depuis un fichier texte (suite)

On peut aussi considérer que le fichier est un objet *itérable* qu'on peut le parcourir tout comme les listes et les chaînes.

### Exemple :

```
>>> f = open("fruits.txt", "r")
>>> for line in f :
    print(line, line.upper(), sep=" * ")
>>> f.close()
```



```
Banane
* BANANE

Kiwi
* KIWI

Pomme
* POMME

Poire
* POIRE

Orange
* ORANGE

Fraise * FRAISE
```

# Opérations sur les fichiers textes (suite)



## Lecture depuis un fichier texte

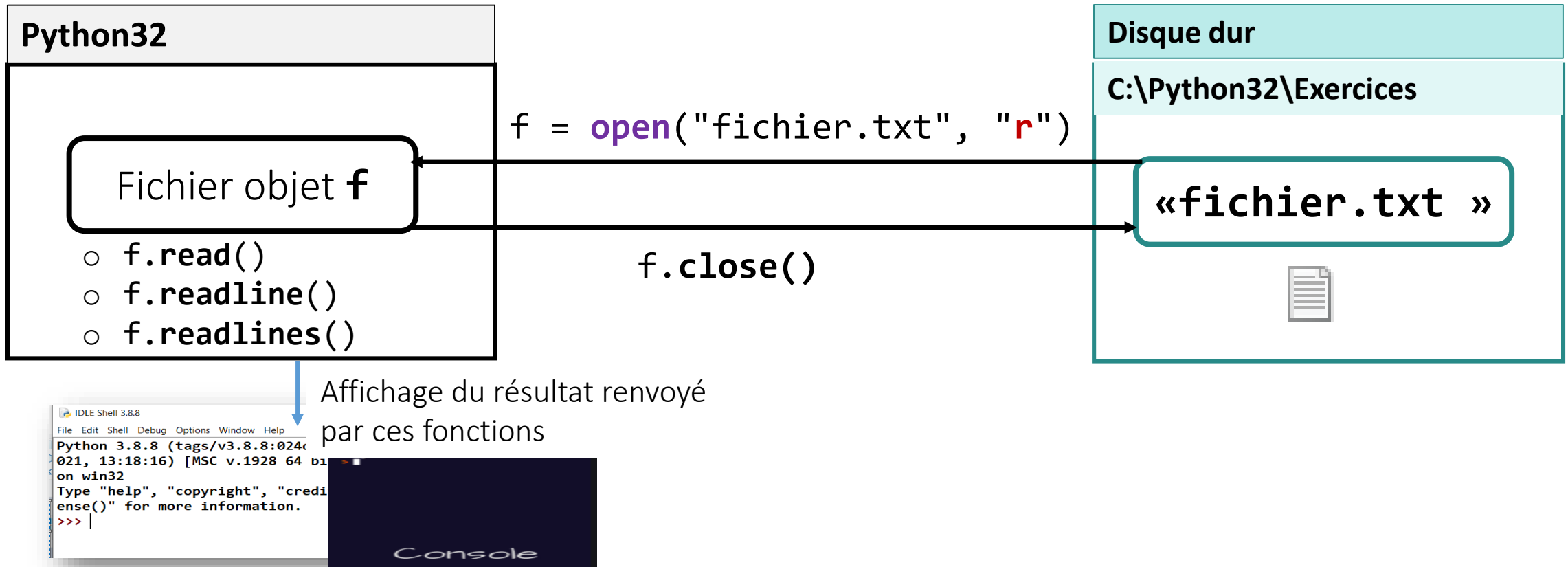
	Commande	Rôle
<b>Ouverture</b> du fichier <b>f</b>	<code>f = open("fichier.txt", "r")</code> <code>f = open("fichier.txt")</code>	Ouvre <b>f</b> en <b>mode</b> <b>LECTURE</b>
<b>Lire</b> dans <b>f</b>	<code>ch = f.read()</code>	Récupère tout le contenu de " <b>fichier.txt</b> " dans une chaîne de caractères <b>ch</b>
	<code>line = f.readline()</code>	Récupère la ligne courante de " <b>fichier.txt</b> " dans une chaîne de caractères <b>line</b>
	<code>L = f.readlines()</code>	Récupère toutes les lignes de " <b>fichier.txt</b> " dans une liste <b>L</b>
	<code>for line in f :</code>	On fait le parcours du fichier " <b>fichier.txt</b> " ligne par ligne ( <b>f</b> est <i>itérable</i> )
<b>Fermeture</b> du fichier <b>f</b>	<code>f.close()</code>	Ferme <b>f</b>



# Opérations sur les fichiers textes (suite)



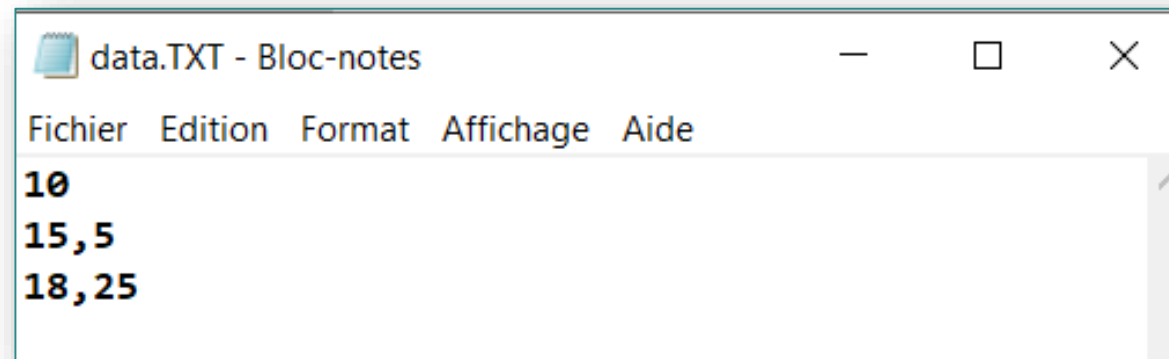
## Lecture depuis un fichier texte



# Opérations sur les fichiers textes (suite)

## activité **Activité :**

On dispose d'un fichier "**data.txt**" contenant des ***nombres réels***, chacun dans une ligne et ayant le séparateur décimal ','



On souhaite écrire un script Python permettant de récupérer ces **nombres réels** dans une **liste de flottants**.

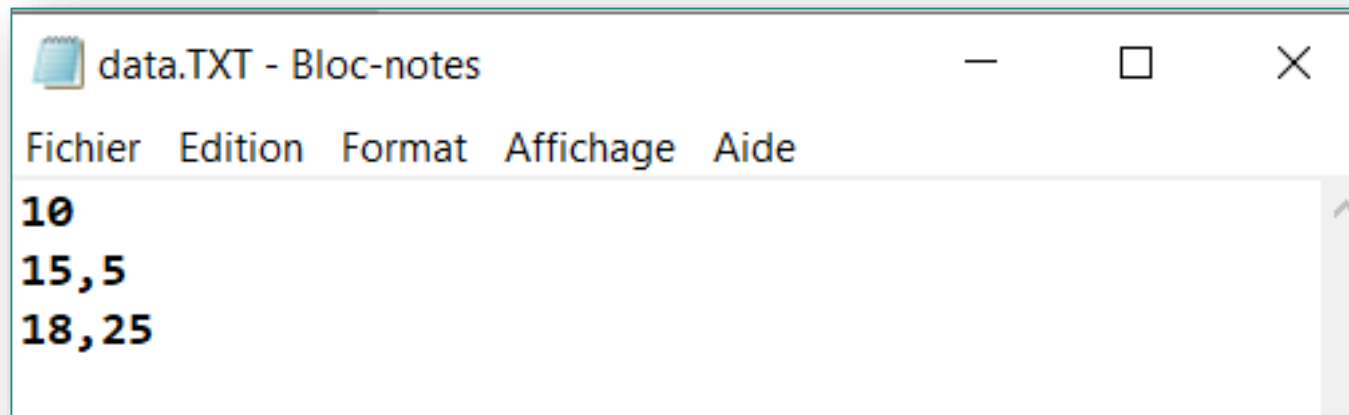
**Attention !** Le séparateur décimal en Python est le '.'

# Opérations sur les fichiers textes (suite)



Vous pouvez créer le fichier **data.txt** via ce code 😊

```
f = open("data.txt", "w")  
  
f.write("10\n15,5\n18,25")  
  
f.close()
```



# Opérations sur les fichiers textes (suite)



## Solution :

```
L=[]  
f = open("data.txt","r")  
for a in f :  
    a = a.rstrip()  
    L.append(a.replace(',','.'))  
  
f.close() # attention hors de la boucle sinon erreur !!  
print(L)
```

Le résultat en sortie : *une  
Liste de chaînes*

```
===== RESTART: C:/Python3  
['10', '15.5', '18.25']  
>>>
```

# Opérations sur les fichiers textes (suite)



## Solution :

Pour obtenir une liste de flottants, on peut ajouter au code précédent :

```
L=[]
f = open("data.txt","r")
for a in f :
    a = a.rstrip()
    L.append(float(a.replace(',', '.', ' ')))
f.close()
print(L)
```

Le résultat en sortie : *une  
Liste de réels*

```
===== RESTART: C:/Python3
[10.0, 15.5, 18.25]
>>>
```

# Opérations sur les fichiers textes (suite)



## Limitation de la fonction write

Toutes les données stockées dans les fichiers textes doivent être converties en chaînes de caractères :

```
>>> a, b, c = 5, 2.83, 67
>>> f = open("MonFichier", "w")
>>> f.write(str(a))
>>> f.write(str(b))
>>> f.write(str(c))
>>> f.close()
```

*f.write(str(a)+"\n"+str(b)+"\n"+str(c))*

```
>>> f = open("MonFichier", "r")
>>> print(f.read())
52.8367
>>> f.close()
```

# Opérations sur les fichiers textes (suite)



## Limitation de la fonction `write` (suite)

```
>>> a, b, c = 5, 2.83, 67
>>> f = open("MonFichier", "w")
>>> f.write(str(a)+"\n"+str(b)+"\n"+str(c))
>>> f.close()
```


```
>>> f = open("MonFichier.txt", "r")
>>> ch = f.read()
>>> ch
'5\n2.83\n67'
>>> type(ch)
<class 'str'>
>>> f.close()
```

# Opérations sur les fichiers textes (suite)



## Limitation de la fonction `write` (suite)

```
>>> f = open("MonFichier", "r")
>>> ch = f.read()
>>> print(ch)
52.8367
>>> type(ch)
<class 'str'>
>>> f.close()
```

- 
- Dans le fichier que nous venons de créer, on trouve une chaîne de caractères contenant des chiffres concaténés.
  - On a perdu l'information sur leurs valeurs et leurs types numériques.
  - Le problème est alors comment conserver les types qui ne sont pas des `str` ?  
→ La solution : utilisation des fichier binaire (module `pickle`) 😊



# Opérations sur les fichiers textes (suite)



## Méthode optimisée d'ouverture et fermeture

Syntaxe :

L'utilisation de `with` avec `open()` garantit la bonne fermeture du fichier

```
with open('nom_fichier', 'mode') as variable:  
    # Code à l'intérieur du bloc with  
    # Vous pouvez lire ou écrire dans le fichier en utilisant la variable  
    # Le fichier sera automatiquement fermé à la fin du bloc with
```

```
f = open("data.txt", "w")  
f.write("10\n15,5\n18,25")  
f.close()
```

Équivalent à



```
with open("data.txt", "w") as f:  
    f.write("10\n15,5\n18,25")
```

## Chapitre 4 : Gestion des fichiers txt/csv en python

- ☐ Introduction
- ☐ Manipulation des fichiers
- ☐ Opérations sur les fichiers textes
- ☐ Le Module pickle
- ☒ **Les fichiers CSV**

# Les fichiers CSV



- Un fichier **CSV** (*Comma-Separated Value*) est un fichier tableur, contenant des données (lettres et chiffres).
  - Pour chaque ligne, les données sont séparées par un caractère de séparation qui est généralement une virgule ",", un point-virgule ";" ou une tabulation "\t".
- 
- On peut lire ou créer un fichier CSV à l'aide d'un tableur comme EXCEL ou OPENOFFICE.
  - En Python, on peut juste traiter ces fichiers comme des fichiers textes, sinon, utiliser le *module* csv de Python qui contient des fonctions spécifiques pour manipuler ce type de fichier.

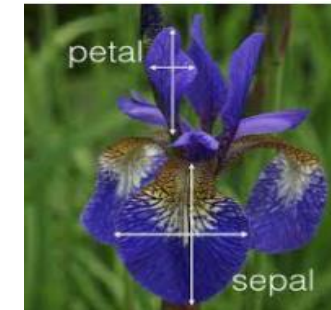
# Les fichiers CSV (suite)

- On peut créer un fichier **.csv** sous EXCEL (en choisissant le type CSV (séparateur : point-virgule) lors de l'enregistrement).

**Exemple** : Le tableau suivant contenu dans "fleurs.csv" :



	A	B	C
1	<b>petal_length</b>	<b>petal_width</b>	<b>species</b>
2	1.4	0.2	iris_setosa
3	1.4	0.3	iris_setosa
4	5.6	2.4	iris_virginica
5	4.1	1.3	iris_versicolor



iris setosa



iris virginica



iris versicolor

Avec BLOC-NOTES, "fleurs.csv" correspond à :

```
*fleur.TXT - Bloc-notes
Fichier Edition Format Affichage Aide
petal_length;petal_width;species
1.4;0.2;iris_setosa
1.4;0.3;iris_setosa
5.6;2.4;iris_virginica
4.1;1.3;iris_versicolor
```

# Les fichiers **CSV** (suite)



## Activité :

On souhaite écrire un script Python permettant de :

- lire le fichier "fleurs.csv",
- récupérer les valeurs de petal\_length dans une liste de réels **PL**,
- récupérer les valeurs de petal\_width dans une autre liste de réels **PW**,
- récupérer les types de fleurs (species) dans une dernière liste de chaînes de caractères **species**.

# Les fichiers **CSV** (suite)



## **Solution :**

On peut aussi construire le fichier '**fleurs.csv**' en tapant les commandes suivantes :

```
fleurs = open('fleurs.csv', 'w')  
  
fleurs.write('petal_length;petal_width;species\n')  
fleurs.write('1.4;0.2;iris_setosa\n')  
fleurs.write('1.4;0.3;iris_setosa\n')  
fleurs.write('5.6;2.4;iris_virginica\n')  
fleurs.write('4.1;1.3;iris_versicolor')  
fleurs.close()
```

# Les fichiers CSV (suite)



## Solution :

```
f = open("fleurs.csv", "r")
f.readline()          # on n'a pas besoin de la 1ère ligne
L = f.readlines()
PL, PW, species = [], [], []
for line in L :
    V = line.split(';')
    PL += [float(V[0].replace(',', '.', ''))]
    PW += [float(V[1].replace(',', '.', ''))]
    species += [V[2].rstrip()]
f.close()
print("Liste des petal_lenght :", PL)
print("Liste des petal_width :", PW)
print("Liste des types de fleurs :", species)
```

# Les fichiers **CSV** (suite)



## Solution :

Résultat après exécution

```
===== RESTART: C:/Python38/Exercices/fleursIris.py =====  
Liste des petal_lenght : [1.4, 1.4, 5.6, 4.1]  
Liste des petal_width : [0.2, 0.3, 2.4, 1.3]  
Liste des types de fleurs : ['iris_setosa', 'iris_setosa', 'iris_virginica', 'iris_versicolor']  
>>>
```



# Module **csv** Lecture

```
from csv import *
```

```
f=open('chemin/exemple2.csv', 'r+')  
data= reader(f, delimiter=",")  
# Par défaut delimiter=",",  
for c in data:  
    print(c) # retourne une liste qui comporte les éléments  
              de la ligne courante  
  
# Il faut obligatoirement fermer le fichier après utilisation  
f.close()
```

# Équivalent à

```
with open('chemin/exemple2.csv', 'r+') as f:  
    data= reader(f, delimiter=",")  
    for c in data:  
        print(c)
```

exemple2.csv

```
xx, zz, ee  
YY, 14, 17
```

```
['xx', 'zz', 'ee']  
['YY', '14', '17']
```

# Module `csv` Ecriture

```
from csv import *
```

```
with open('chemin/exemple3.csv', 'w', newline="") as f:
```

```
    wr= writer(f, delimiter=';')
```

```
    wr.writerow(['XXX','ZZZ',14])
```

```
    wr. writerow(['RRR', '2AZE', 123,77])
```

#Si **newline=""** n'est pas spécifié, une ligne vide sera ajouter après chaque appel de **writerow()**

*\* Ecriture à partir d'un dictionnaire*

```
with open('chemin/exemple4.csv', 'w', newline="") as f:
```

```
    wr= DictWriter(f, fieldnames=['col_1', 'col_2'], delimiter=";")
```

```
    wr.writeheader()
```

```
    wr.writerow({'col_1': 'XXX', 'col_2': 'YYY'})
```

```
    wr.writerow({'col_1': '456', 'col_2': '123'})
```

# Assemblage Google Drive dans Colaboratory

1



# Ou bien



```
from google.colab import drive
drive.mount('/content/drive')
```

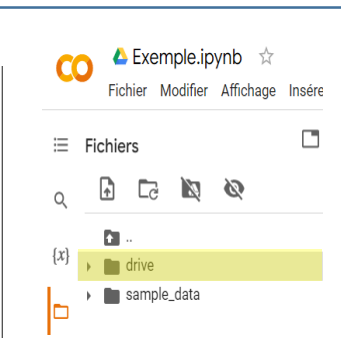
Mounted at /content/drive

2

**Autoriser ce notebook à accéder à vos fichiers Google Drive ?**

La connexion à Google Drive autorisera le code exécuté dans ce notebook à modifier les fichiers contenus dans votre Google Drive jusqu'à ce que l'accès soit révoqué.

[Non, merci](#) [Se connecter à Google Drive](#)



# Assemblage Google Drive dans Colaboratory

```
from google.colab import drive
drive.mount('/content/gdrive')
drive_path = '/content/gdrive/My Drive/ExercicesPython/fruit.TXT'

f = open(drive_path, mode='r')
```

## Chapitre 4 : Gestion des fichiers txt/csv en python

- ☐ Introduction
- ☐ Manipulation des fichiers
- ☐ Opérations sur les fichiers textes
- ☐ Les fichiers CSV
- ☒ **Interaction avec le système d'exploitation (Module os)**

# Le module os

- Le **module os** est un **module** fournit par **Python** dont le but d'*interagir avec le système d'exploitation*. Il permet ainsi de ***gérer l'arborescence des fichiers***.

- Le **module os** peut être **chargé** simplement avec la commande :

```
>>> import os
```

- Les fonctions les plus utilisées dans le **module os** sont :

Fonction	
<code>getcwd()</code>	Renvoie le répertoire courant
<code>chdir(newrep), mkdir(rep)</code>	Change le répertoire courant par <i>newrep</i> , Crée le répertoire <i>rep</i>
<code>rename(src,dest)</code>	Renomme un fichier ou un répertoire <i>src</i> en <i>dest</i>
<code>remove(rep), rmdir(rep)</code>	Supprime le répertoire <i>rep</i>
<code>listdir(rep)</code>	Liste des fichiers de <b>rep</b>

# Le module **os** (suite)



## Exemple : Répertoire courant

### **#Méthode 1**

```
>>> import os          #importer le module os
>>> os.getcwd()        #Indique le répertoire de travail en cours
'C:\\Python32'
```

### **#Méthode 2**

```
>>> from os import getcwd #importer la fonction getcwd du module os
>>> rep = getcwd()
>>> print(rep)
C:\\Python32
```

# Le module **os** (suite)



## Exemple : Changement du répertoire courant

```
>>> import os
>>> os.chdir("C:\\") # change Le répertoire courant C:\\Python32 par C:\\
>>> rep = os.getcwd()
>>> print(rep)
C:\\
```



## La fonction **listdir** liste le contenu du répertoire courant

```
>>> os.listdir(rep) #ici rep = C:\\
>>> os.listdir(os.getcwd())
```



# Le module **os** (suite)



Revenir au répertoire courant par défaut **C:\\Python32** :

```
>>> os.chdir("C:\\Python38")
```



Sous C:\\Python32, créez le dossier « Exercices » avec la fonction **mkdir** :

```
>>> os.mkdir("Exercices")
>>> os.mkdir("Exercices")
```

Traceback (most recent call last):

File "<pyshell#2>", line 1, in <module>  
mkdir("Exercices")

FileExistsError: [WinError 183] Impossible de créer un fichier déjà existant: 'Exercices'

```
>>> os.chdir("C:\\Python38\\Exercices") #Change le répertoire courant
```

```
>>> os.getcwd() #On peut vérifier le nouveau répertoire courant
```

```
'C:\\Python38\\Exercices'
```

*#La commande mkdir génère une erreur si le dossier Exercices existe*

# Le module **os** (suite)

- Pour faciliter l'accès aux fichiers manipulés, on suppose qu'ils sont placés dans le même répertoire de travail, c'est-à-dire où on enregistre notre script Python qui le manipule (en général c'est '**C:\\Python38**').
- Dans notre script, on ne fait **import os** que lorsqu'on veut utiliser l'une des fonctions précitées (**chdir**, **listdir**, **mkdir**, ...) .





**Fin Chapitre 4**

**Fatma Ben Said**

[fatma.bensaid@iit.ens.tn](mailto:fatma.bensaid@iit.ens.tn)