FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Chapitre 2 : Les fichiers en Python

Licence Fondamentale : Tronc commun Informatique Appliquée

Pr: Y.Ouassit

Chapitre 2: Gestion des fichiers

1. Introduction à la gestion des fichiers
2. Manipulation des fichiers (textes / binaires) :
    a) Ouverture des fichiers
    b) Lecture / Ecriture dans les fichiers
    c) Fermeture des fichiers
    d) L'instruction **with**
3. Les modules de gestion des fichiers : **pathlib** et **shutil**
4. Applications

Fichier : Définition

Un fichier est un **ensemble de données enregistrées sous un nom dans un système de stockage**, que l'on peut lire, modifier ou supprimer à l'aide d'un ordinateur.

Un fichier permet de :

- conserver des données de manière permanente
- stocker des informations produites par un programme
- échanger des données entre utilisateurs ou programmes

Fichier : Caractéristiques

Un fichier se caractérise par :

1. **Nom :** Chaque fichier possède un nom qui permet de l'identifier.
2. **Extension :** L'extension indique le type de fichier et souvent le logiciel associé.

   (.txt → fichier texte

   .pdf → document PDF

   .docx → document Word

   .jpg → image)
3. **Taille :** L'espace occupé dans la mémoire.
4. **Chemin d'accès :** C'est l'endroit où le fichier est stocké dans l'ordinateur.
5. **Catégories :** Comment les données sont stockées dans le fichier.

Fichier : Catégories

Il existe deux catégories de fichiers :

- **Fichiers textes**
- **Fichiers binaires**

Fichier : Catégories

➢ **Fichiers textes** :

Un fichier texte contient des caractères lisibles par l'être humain.
On peut l'ouvrir avec le Bloc-notes ou n'importe quel éditeur de texte.

Exemples : .txt .csv .py

À l'intérieur, ce sont des lettres, des chiffres, des mots.

Fichier : Catégories

➢ **Fichiers binaires** :

Un fichier binaire, contient des données codées en langage machine.

On ne peut pas le lire directement avec un éditeur de texte, car il n'est pas destiné à être lisible par l'humain.

Exemples : .jpg (image) .mp3 (audio) .exe (programme)

Fichier : Catégories

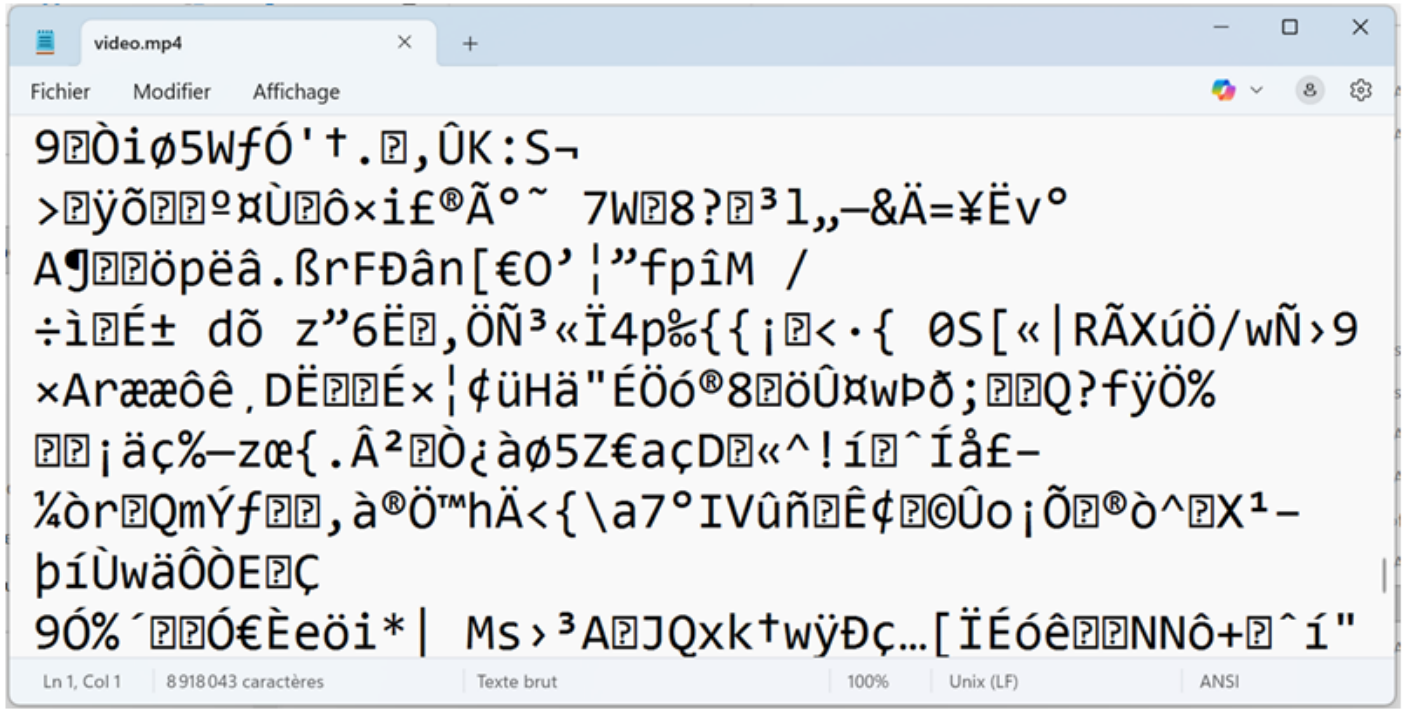
Fichier Texte

Ingrédients
150 g de chocolat
100 g de sucre en poudre
100 g de beurre
75 g de farine
1 oeuf
3 cuillères de lait

Ustensiles
2 cuillères à soupe
1 saladier
1 moule

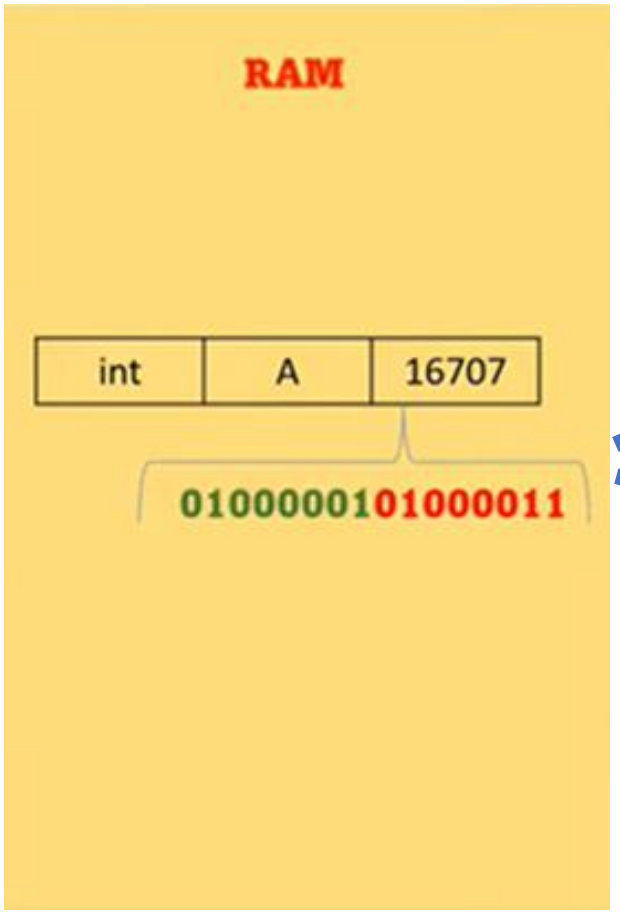
Recette.txt

Fichier Binaire



video.mp4

Fichier : Catégories



TRAITEMENT TEXTE

TRAITEMENT BINAIRE

Fichier texte

00110001 00110110 00110111 00110000 00110111

Fichier Binaire

0100000101000011

Pourquoi le besoin de manipuler les fichiers en Python ?

Imaginons que vous avez développé un jeu, et que à la fin de chaque partie du jeu les joueurs obtiennent des points. Et un classement des joueurs est affiché la fin :

La table des scores doit être persistante, pour être chargé à chaque nouveau jeu.

POS	Joueur	POINTS
1	TEAM 1	123
2	TEAM 2	113
3	TEAM 3	106
4	TEAM 4	96
5	TEAM 5	91
6	TEAM 6	85
7	TEAM 7	81
8	TEAM 8	74
9	TEAM 9	71

Etapes pour manipuler un fichier :

La manipulation d'un fichier en Python passe par trois étapes :

1. **Ouvrir un fichier**
2. **Lire / Ecrire dans un fichier**
3. **Fermer un fichier**

Etape 1 : Ouvrir un fichier

Ouvrir un fichier en Python signifie établir une connexion entre le programme et le fichier afin de pouvoir lire, écrire ou modifier son contenu.

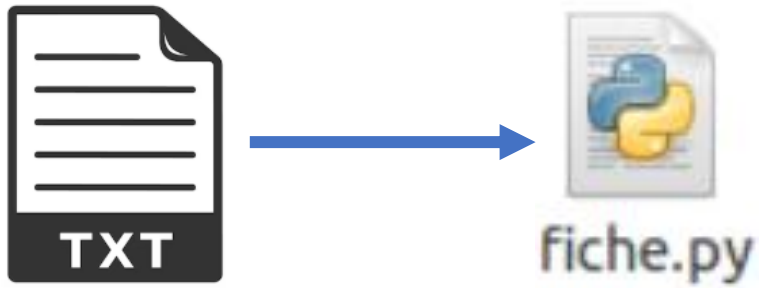
Concrètement quand un fichier est ouvert :

- Le programme peut **accéder aux données** du fichier,
- Le système d'exploitation **autorise** l'accès selon le mode choisi,
- Un **lien temporaire** est créé entre le fichier et le programme.

Etape 2 : Lire/Ecrire dans un fichier

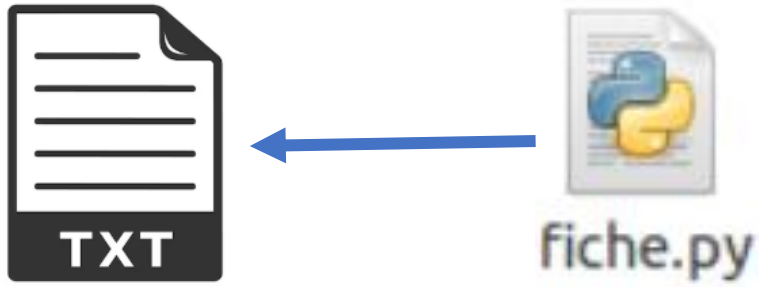
✔ Lecture d'un fichier :

Permet de récupérer le contenu d'un fichier.



✔ Écriture dans un fichier :

Permet d'enregistrer des données dans un fichier.



Etape 3 : Fermer un fichier

Fermer un fichier signifie mettre fin à la connexion entre le programme et le fichier après son utilisation.

Pourquoi fermer un fichier :

- Sauvegarder correctement les données écrites
- Libérer les ressources du système

La notion de curseur dans un fichier :

Définition :

Le curseur est un pointeur invisible qui indique la position actuelle dans un fichier à partir de laquelle la lecture ou l'écriture va se faire.

Rôle du curseur:

- Il indique où lire
- Il indique où écrire
- Il se déplace automatiquement après chaque lecture ou écriture

OPEN DEMO

Manipulation des fichiers

Les Fichiers Textes

Etape 1 : Ouvrir un fichier

- Pour ouvrir un fichier nous utilisons la fonction : **open()**

Syntaxe:

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

filepath : chemin du fichier que nous voulons ouvrir

mode : le mode d'ouverture (lire, écrire, les deux, …)

Cette fonction renvoie un objet **file**, également appelé descripteur, qui permet de lire ou de modifier le fichier.

Etape 1 : Ouvrir un fichier

Chemin absolu vs relatif :

1. Chemin absolu (Absolute path) :

Un chemin absolu indique l'emplacement complet d'un fichier ou dossier à partir de la racine du système.

Exemple : C:\\Users\\Ahmed\\Documents\\projet\\file.txt

2. Chemin relatif (Relative path) :

Un chemin relatif indique l'emplacement d'un fichier par rapport au dossier courant.

Il dépend de l'endroit où le programme est exécuté.

Exemple : file.txt

Etape 1 : Ouvrir un fichier

Comment trouver le répertoire de travail courant ?

```
import os
print(os.getcwd())
```

```
Output :
C:\Users\Lenovo
```

Etape 1 : Ouvrir un fichier

Comment changer le répertoire de travail courant ?

```
import os

os.chdir("C:/TP")  # Pour changer le répertoire courant

print(os.getcwd())
```

Output :

C:\TP

Etape 1 : Ouvrir un fichier

Chemin absolu vs relatif :

Remarque :

Sous Windows il faut séparer les dossiers avec **/** ou ****.

Exemples :

C:/tp/notes.txt	**est un chemin valide**
C:\\tp\\notes.txt	**est un chemin valide**
C:\tp\notes.txt	**est un chemin incorrecte**

Manipulation des fichiers textes

Etape 1 : Ouvrir un fichier

Il existe plusieurs mode d'ouverture :

Mode	Description
r	Lecture seule (le fichier doit exister)
w	Écriture (crée le fichier ou l’écrase)
a	Ajout à la fin du fichier(crée le fichier s'il n'existe pas)
r+	Lecture et écriture
w+	Ecriture et lecture
a+	Ajout à la fin du fichier et lecture

Etape 1 : Ouvrir un fichier

Une fois un fichier ouvert, nous avons un objet fichier contenant diverses informations relatives à ce fichier :

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

1. f.closed : Renvoie vrai si le fichier est fermé, faux sinon.
2. f.mode : Renvoie le mode d'accès utilisé lors de l'ouverture du fichier.
3. f.name : Nom du fichier

Manipulation des fichiers textes

Etape 1 : Ouvrir un fichier

Example :

```
f = open("notes.txt","r")
print("Nom du fichier:", f.name)
print("Fermé ou non:", f.closed)
print("Mode d'ouverture :", f.mode)
```

Output :

Nom du fichier : notes.txt

Fermé ou non : False

Mode d'ouverture : r

Etape 2 : Lire un fichier

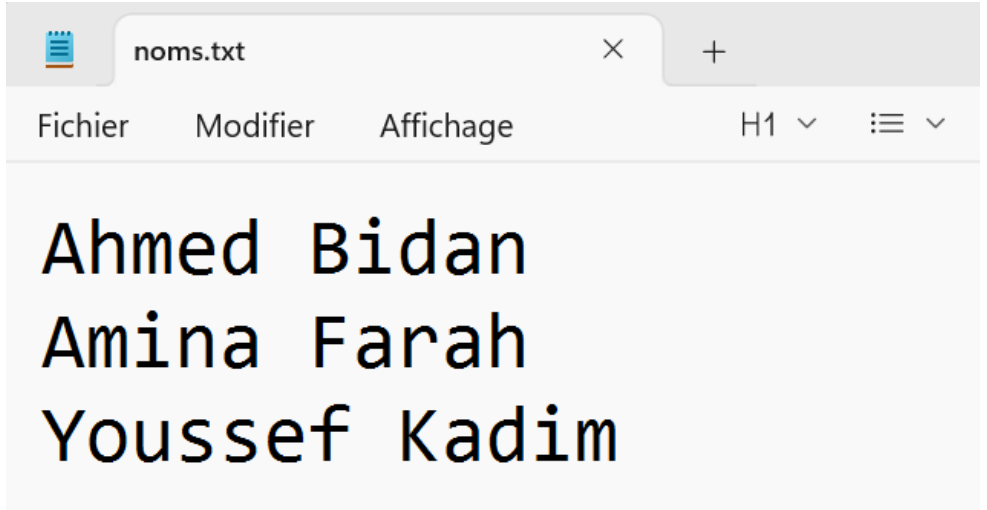
Pour lire un fichier texte on dispose de plusieurs méthodes :

```
f = open('noms.txt', 'r')
```

1. La méthode **f.read()** : Permet de lire le texte entier du fichier.
2. La méthode **f.read(n)** : Permet de lire n caractères.
3. La méthode **f.readline()** : Permet de lire une ligne du fichier.
4. La méthode **f.readlines()** : Permet de lire les lignes du fichier dans une liste.

Etape 2 : Lire un fichier

Exemples :

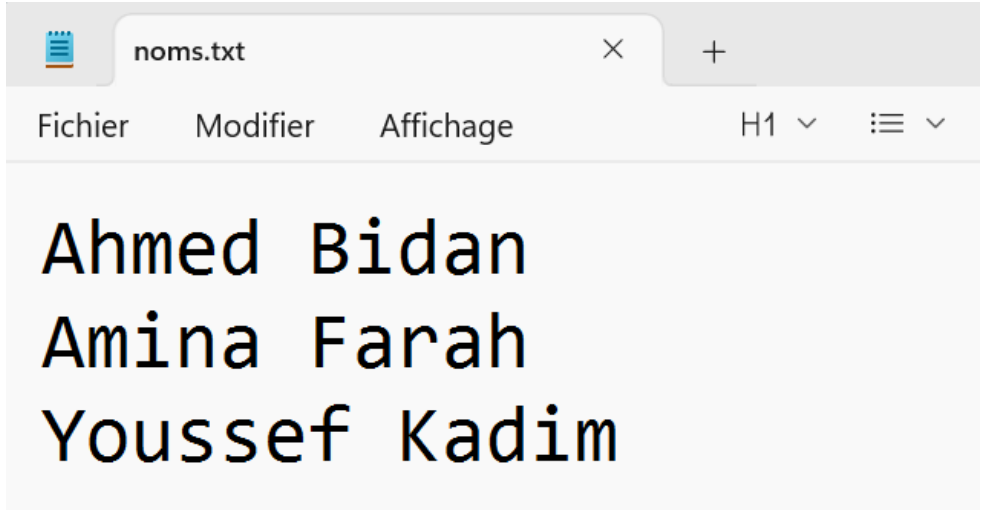



```
f = open('noms.txt', 'r')
contenu = f.read()
print(contenu)  # affiche tout le contenu du fichier
```

Etape 2 : Lire un fichier

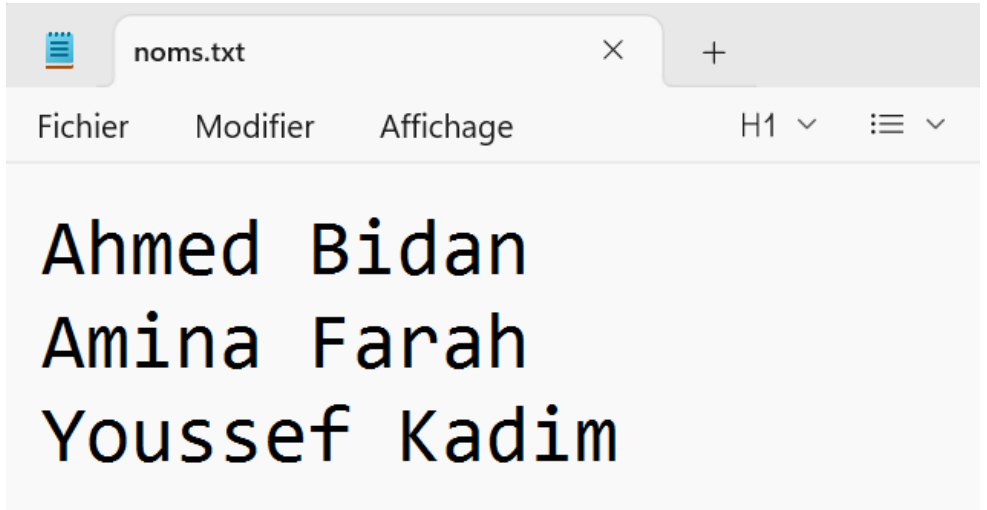
Exemples :

```
f = open('noms.txt', 'r')
ch = f.read(5)
print(ch)  # affiche :Ahmed
ch = f.read(3)
print(ch)  # affiche : Bi
ch = f.read(5)
print(ch)  # affiche : dan\nA
```



Manipulation des fichiers textes

Etape 2 : Lire un fichier



Exemples :

```
f = open('noms.txt', 'r')
```

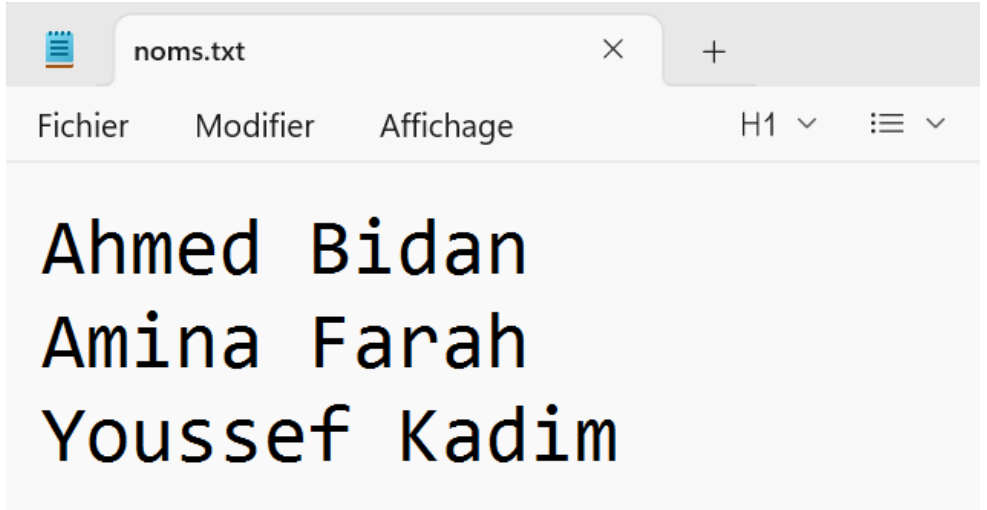
```
# Lire et affiche le fichier par bloc de 3 caractères
ch = f.read(3)
while ch!='' :
        print(ch)
        ch = f.read(3)
```

```
Ahm
ed
Bid
an
Ami
na
Far
....
```

Etape 2 : Lire un fichier

Exemples :

```
f =  open('noms.txt', 'r')
ch = f.readline()
print(ch)  # affiche :Ahmed Bidan
ch = f.readline()
print(ch)  # affiche :Amina Farah
```



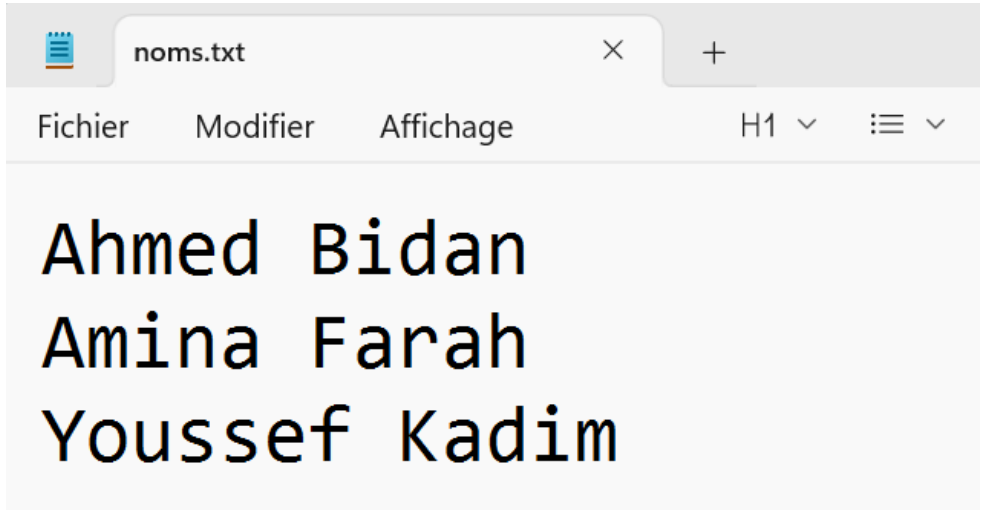
Manipulation des fichiers textes

Etape 2 : Lire un fichier

Exemples :

```
f = open('noms.txt', 'r')
```

```
# Lire et affiche le fichier ligne par ligne
ch = f.readline()
while ch != '' :
        print(ch)
        ch = f.readline()
```



Ahmed Bidan

Amina Farah

Youssef Kadim

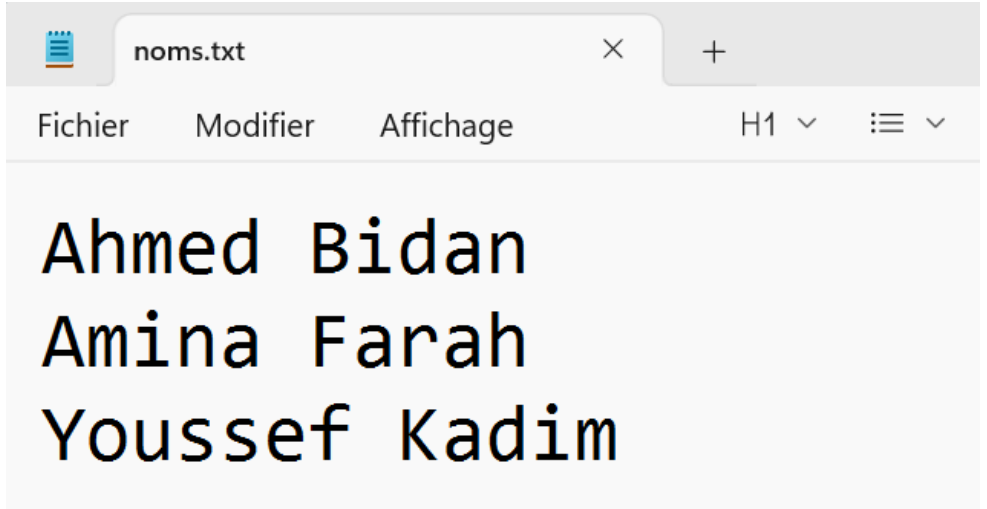
Manipulation des fichiers textes

Etape 2 : Lire un fichier

Exemples :

```
f = open('noms.txt', 'r')
```

```
# Lire et affiche le fichier ligne par ligne
for ligne  in f :
        print(ligne)
```



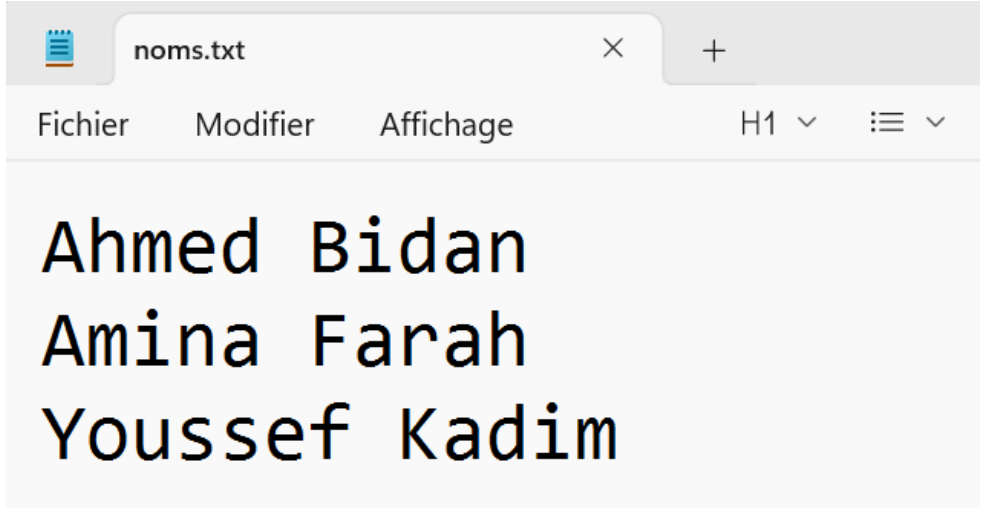
Ahmed Bidan

Amina Farah

Youssef Kadim

Manipulation des fichiers textes

Etape 2 : Lire un fichier



Exemples :

```
f =  open('noms.txt', 'r')


L = f.readlines()
print(L)  # affiche :  ["Ahmed Bidan", "Amina Farah", "Youssef Kadim"]
```

Etape 2 : Lire un fichier

Remarque :

Pour des fichiers très volumineux, il est recommandé d'utiliser :

read(n) → lecture par blocs

readline() → lecture ligne par ligne

☞ *Ces méthodes lisent le fichier progressivement, ce qui réduit l'utilisation de la mémoire.*

Etape 2 : Ecrire dans un fichier

Pour écrire dans un fichier texte on dispose de deux méthodes :

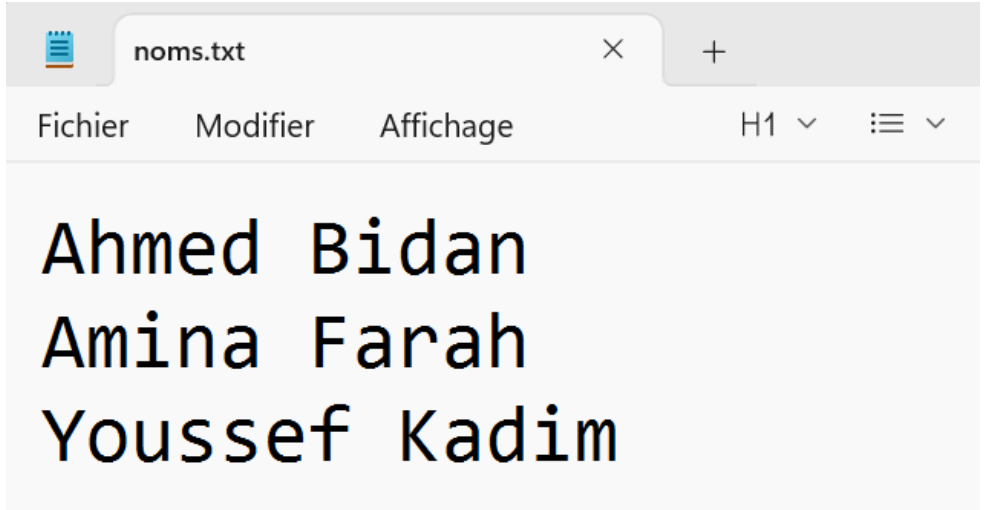
f = open('noms.txt', 'w')

1. La méthode **f.write(texte)** : Écrit une chaîne de caractères dans le fichier
2. La méthode **f.writelines(L)** : Écrit une **liste de chaînes (**la fonction l'ajoute pas un saut de ligne**)**

Etape 2 : Ecrire dans un fichier

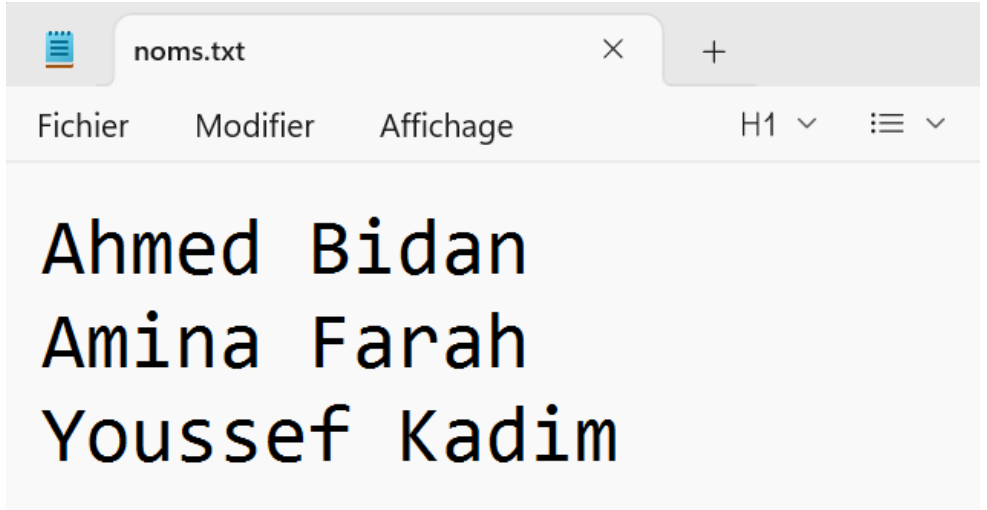
Exemples :

```
f = open('noms.txt', 'a')
f.write("Hanaa Youmi\n")
```



Etape 2 : Ecrire dans un fichier

Exemples :



f = open('noms.txt', 'a')

L = ["Aymen Azaro\n","Fatine Roki\n","Hanaa Youmi\n"]

f.writelines(L)

Etape 3 : Fermer un fichier

Syntax:

```
file.close()
```

Example:

```
f=open('notes.txt', 'r')
print("Name of the file:",f.name)
f.close()
```

Manipulation des fichiers textes

L'instruction with :

Lorsqu'on manipule un fichier en Python, il faut généralement :

1. **ouvrir le fichier**
2. **effectuer des opérations** (lecture ou écriture)
3. **fermer le fichier**

```
f = open("output.txt","w)
...
f.write("Hello Python!")
...
f.close()
```

Problème possible :

- Si une erreur survient, le fichier peut rester ouvert.
- Cela peut provoquer des fuites de ressources.

Manipulation des fichiers textes

L'instruction with :

Python propose l'instruction **with** pour gérer automatiquement la fermeture des fichiers.

with crée un contexte : dès qu'on sort du bloc, Python fait automatiquement le nettoyage, par exemple :fermer le fichier (close()) même s'il y a une erreur dans le bloc.

Example:

```
f = open("output.txt","w)
f.write("Hello Python!")
f.close()
```

```
with open("output.txt","w) as f:
     f.write("Hello Python!")
```

Manipulation des fichiers textes

L'instruction with :

Il est possible d'ouvrir plusieurs fichiers dans une seule instruction with.

Syntaxe :

```
with open("fichier1.txt", "r") as f1, open("fichier2.txt", "w") as f2:
    # instructions
```

Les deux fichiers seront **fermés automatiquement** à la fin du bloc.

Manipulation des fichiers textes

Autres méthodes utiles

f = open("notes.txt", "r")

f.seek(n) : Déplace le **curseur** à une position précise.

f. tell() : Retourne la position actuelle du curseur.

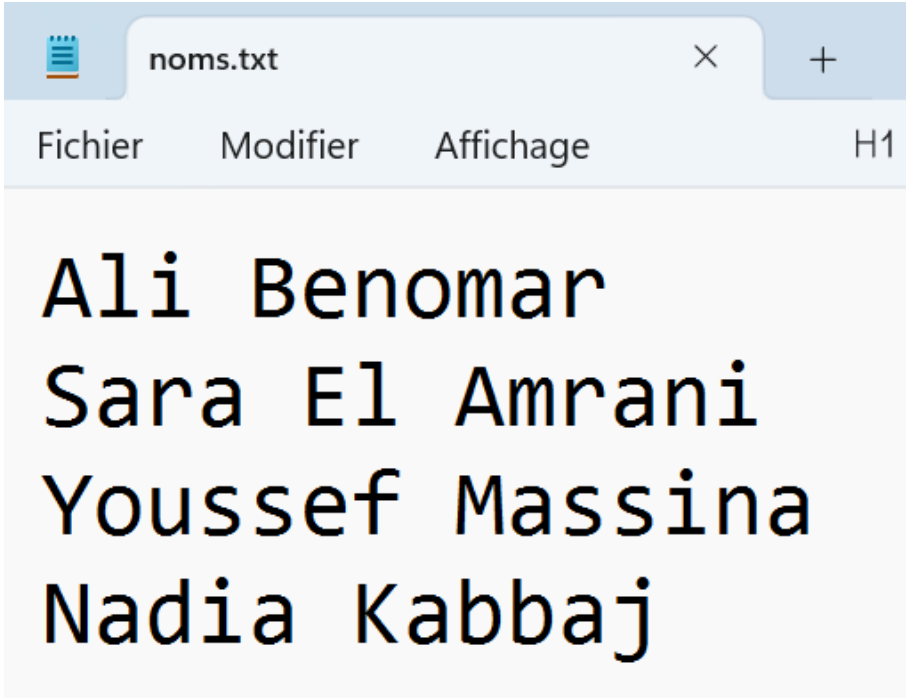
f. readable() : Vérifie si le fichier peut être lu.

f. writable() : Vérifie si le fichier peut être écrit.

f. seekable() : Vérifie si le curseur peut être déplacé.

Exemples : La fonction seek

```
with open("noms.txt", "r") as f:
    print(f.read(3))
    f.seek(0)        # aller au début
    print(f.read(3))   # lit 3 caractères
    f.seek(13)        # aller à la position 13
    print(f.read(5))   # lit 5 caractères
    print(f.tell())
```



Output :

```
Ali
Ali
Sara
18
```

Manipulation des fichiers textes

Exercice 1 :

Ecrire un programme Python qui affiche les noms des étudiants stockés dans un fichier "etudiants.txt" :

Ali
Sara
Youssef
Nadia

etudiants.txt

```
f = open("etudiants.txt", "r")
for ligne in f:
    print(ligne)
f.close()
```

```
with open("etudiants.txt", "r") as f :
    for ligne in f:
        print(ligne)
```

Manipulation des fichiers textes

Exercice 2 :

Ecrire un programme Python qui affiche les noms et notes des étudiants stockés dans un fichier "notes.txt" :

```
Ali  12
Sara  17
Youssef  5
Nadia  13
```

notes.txt

```
f = open("notes.txt", "r")
for ligne in f:
    nom, note = ligne.split()
    print(nom , 'a eu', note)
f.close()
```

```
with open("notes.txt", "r") as f :
    for ligne in f:
        nom, note = ligne.split()
        print(nom , 'a eu', note)
```

Manipulation des fichiers textes

Exercice 3 :

Ecrire un programme Python qui affiche les noms et notes des étudiants stockés dans un fichier "notes.txt" et n'afficher que les étudiants qui ont validé :

```
Ali  12
Sara  17
Youssef  5
Nadia  13
```

notes.txt

```
with open("notes.txt", "r") as f:
    for ligne in f:
        nom, note = ligne.split()
        if float(note) >= 10 :
            print(nom , 'a validé')
```

Manipulation des fichiers textes

Exercice 4 :

Ecrire un programme Python qui lit les noms et notes des étudiants depuis un fichier Notes.txt , puis enregistrer dans un autre fichier "Valide.txt" que les noms des étudiants qui ont validé le module.

Notes.txt

Valide.txt

Manipulation des fichiers textes

Exercice 4 :

```
f = open("notes.txt", "r") # ouvrir le fichier en lecture
g = open("valide.txt", "w") # ouvrir le fichier en écriture

for ligne in f:
    nom, note = ligne.split()
    if float(note) >= 10 :
        g.write(nom+'\n')

f.close()
g.close()
```

Manipulation des fichiers textes

Exercice 4 :

```
with open("notes.txt", "r") as f, open("valide.txt", "w") as g :
    for ligne in f:
        nom, note = ligne.split()
        if float(note) >= 10 :
            g.write(nom+'\n')
```

Les Fichiers Binaires

Ouvrir un fichier binaire

Pour ouvrir un fichier en mode binaire, ajoutez 'b' au paramètre de mode dans la fonction open(). Par exemple, 'rb' ouvre un fichier pour la lecture en mode binaire, et 'wb' ouvre un fichier pour l'écriture en mode binaire.

```
# Pour lire en mode binaire un fichier
f = open('filename', mode='rb’)
# Pour écrire dans fichier en mode binaire
f = open('filename', mode=‘wb’)
```

Manipulation des fichiers textes

Etape 1 : Ouvrir un fichier

Il existe plusieurs mode d'ouverture :

Mode	Description
rb	Lecture seule (le fichier doit exister)
wb	Écriture (crée le fichier ou l’écrase)
ab	Ajout à la fin du fichier(crée le fichier s'il n'existe pas)
rb+	Lecture et écriture
wb+	Ecriture et lecture

Etape 2 : Lire un fichier binaire

Pour lire un fichier texte on dispose de plusieurs méthodes :

f = open('noms.txt', 'rb')

1. La méthode **f.read()** : Permet de lire tout le contenu du fichier, et renvoie un objet de type bytes.
2. La méthode **f.read(n)** : Permet de lire n octets.

Etape 2 : Lire un fichier binaire

Lors de la lecture d'un fichier binaire, les données sont renvoyées sous forme d'objets bytes, et non de chaînes de caractères. Ceci est important lors du traitement ou de la manipulation des données dans le fichier.

```
with open("example.dat", "rb") as f:
    binary_data = f.read()
    print(binary_data)
```

Etape 2 : Lire un fichier binaire

Pour écrire dans un fichier texte on dispose d'une seule méthode :

f = open('noms.txt', 'wb')

1. La méthode **f.write(data)** : écrit des **bytes** (pas du texte normal).

Etape 2 : Ecrire dans un fichier binaire

L'écriture dans un fichier binaire est similaire à l'écriture dans un fichier texte, sauf que les données doivent être sous forme d'octets :

```
data = b'This is binary data'
with open("example.dat", "wb") as f:
    f.write(data)
```

La sérialisation des objets :

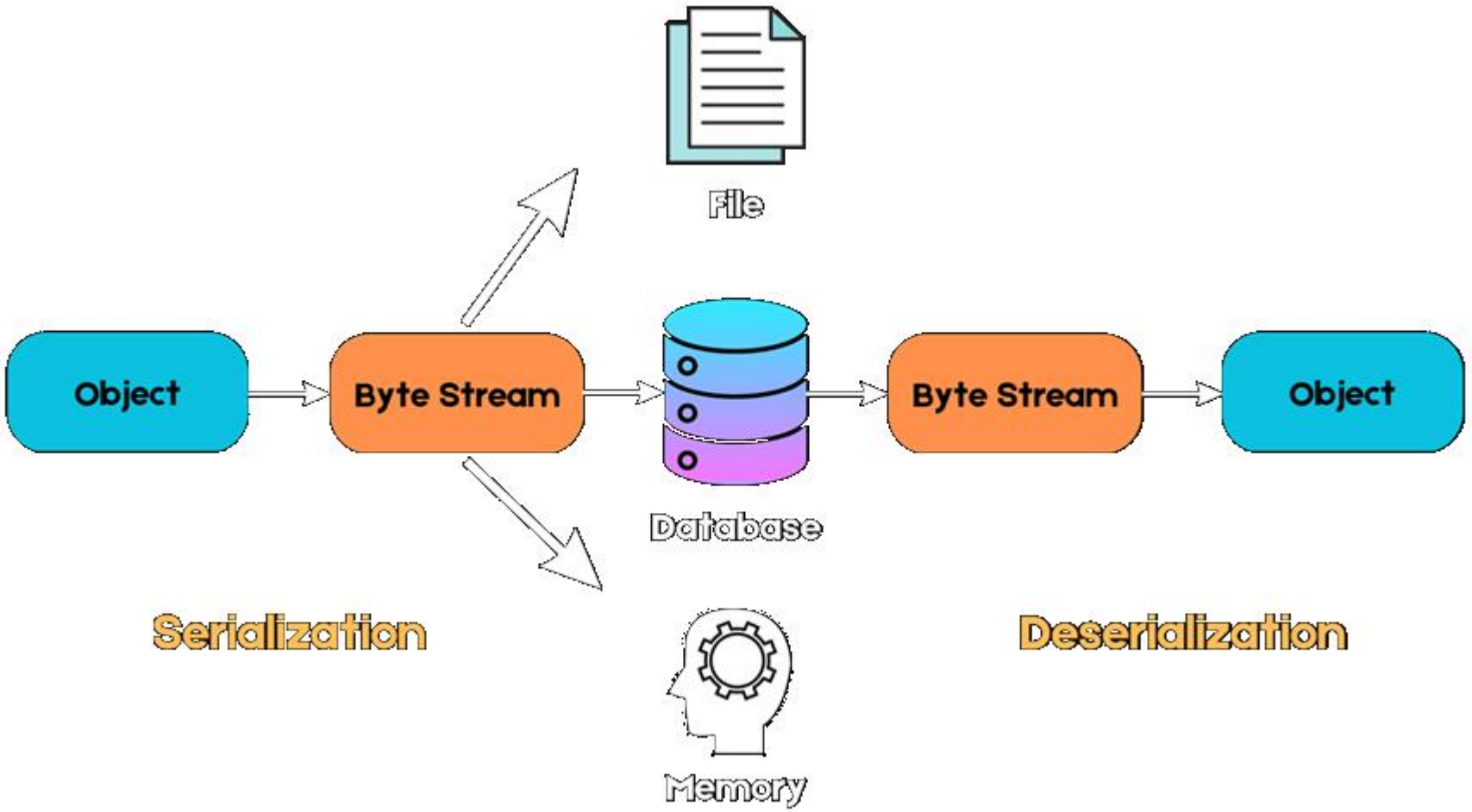
La sérialisation d'objets est le fait de transformer un objet (en mémoire) en un format qui peut être :

- stocké (fichier, base de données)
- envoyé sur le réseau (entre deux services)
- mis en cache

En général, on le transforme en :

- JSON
- XML
- binaire
- texte

La sérialisation des objets :



Manipulation des fichiers binaires

Le module pickle :

pickle est un module Python qui permet de :

1. Sérialiser un objet Python → le transformer en format binaire
2. Désérialiser → reconstruire l'objet Python à partir de ce format
3. Ils permet se sérialiser touts les types de données (str, list, dict,set,…)

Manipulation des fichiers binaires

Le module pickle :

Quelques méthodes du module pickle :

Méthode	Description
pickle.dump(object, file)	Écrire (**sérialiser**) un objet Python dans un fichier binaire.
object = pickle.load(file)	Lire (**désérialiser**) un objet Python depuis un fichier binaire.
….	

Manipulation des fichiers binaires

Exemples :

Sauvegarder une liste dans un fichier binaire:

```
import pickle

liste = [10, 20, 30, 40]

with open("liste.dat", "wb") as f:

        pickle.dump(liste, f)
```

Manipulation des fichiers binaires

Exemples :

Charger une liste depuis un fichier binaire:

```
import pickle

with open("liste.dat", "rb") as f :

        liste = pickle.load(f)

print(liste)
```

Manipulation des fichiers binaires

Exercice 1 :

Ecrire un programme qui sauvegarde une liste d'étudiants (liste de dictionnaires) dans un fichier binaire.

```
etudiants = [
    {"nom": "Alice", "age": 20, "moyenne": 15.5},
    {"nom": "Bob", "age": 22, "moyenne": 13.8},
    {"nom": "Charlie", "age": 19, "moyenne": 17.2}
]
```

```
import pickle

with open("etudiants.dat", "wb") as f:
    pickle.dump(etudiants, f)

print("Données enregistrées avec succès.")
```

Manipulation des fichiers binaires

Exercice 2 :

Ecrire un programme qui charge la liste d'étudiants (liste de dictionnaires) depuis le fichier "etudiants.dat", ajoute un nouveau étudiant et sauvegarde la nouvelle liste dans le même fichier..

Manipulation des fichiers binaires

Exercice 2 :

```
import pickle

# Lecture
with open("etudiants.dat", "rb") as f:
    etudiants = pickle.load(f)

# Modification
etudiants.append({'nom':'Yassine', 'age':13, 'moyenne':15})

# Réécriture complète
with open("etudiants.dat", "wb") as f:
    pickle.dump(etudiants, f)

print("Données enregistrées avec succès.")
```

Modules de gestion de fichiers et dossiers

Les modules de gestion des fichiers

Introduction :

```
f = open("c:/TP/notes.txt" , "w")
```

Nous avons utilisé la fonction open() pour :

- lire un fichier
- écrire dans un fichier

Limites :

- Pas possible de manipuler les dossiers
- Certaines opérations (copier,…) nécessitent plusieurs instructions

Les modules de gestion des fichiers

Introduction :

Un programme pour copier un fichier :

```
with open("c:/TP/notes.txt", "r") as f :
    ch = f.read()

with open("c:/TP1/notes.txt", "w") as g :
    g.write(ch)
```

NB: 4 lignes de code pour un simple opération

Les modules de gestion des fichiers

Introduction :

Python met à notre disposition plusieurs module pour rendre facile la manipulation des dossiers et fichiers :

- ❑ Le module : os.path
- ❑ Le module : pathlib
- ❑ Le module : shutil
- ❑ …

Le module pathlib

Les modules de gestion des fichiers

Le module pathlib :

pathlib est un module standard de Python qui permet de manipuler les chemins de fichiers et de dossiers de manière simple, moderne et orientée objet.

Il a été introduit dans Python 3.4 pour remplacer progressivement les manipulations de chemins faites avec os et os.path.

Les modules de gestion des fichiers

Le module pathlib :

1. Classes “pure” (ne touchent pas au système de fichiers)
Ces classes servent seulement à manipuler des chemins (concaténation, extraction de parties, etc.) sans accéder au disque.

- PurePath
  Classe de base abstraite pour les chemins.
- PurePosixPath
  Chemins de type POSIX (Linux, macOS).
- PureWindowsPath
  Chemins de type Windows.

2. Classes “concrètes” (accès réel au système de fichiers)
Ces classes permettent de lire, créer, supprimer, vérifier des fichiers.

- **Path**
  Classe principale utilisée dans la plupart des cas.
- PosixPath
  Implémentation pour systèmes POSIX.
- WindowsPath
  Implémentation pour Windows.

Les modules de gestion des fichiers

Le module pathlib :

Importer la classe Path:

```
from pathlib import Path
```

Créer un chemin :

```
path1 = Path("c:/TP/notes.txt") # chemin du fichier notes.txt

path2 = Path("c:/TP")           # chemin du dossier TP
```

Les modules de gestion des fichiers

Le module pathlib :

Créer un chemin :

```
path1 = Path(".")   # dossier courant. Exemple (C:/Users/Youssef)

path2 = Path("..")  # dossier parent. Exemple (C:/Users)

path3 = Path("data") # C:/Users/Youssef/data

path4 = Path("data/fichier.txt") # c:/Users/Youssef/fichier.txt
```

Les modules de gestion des fichiers

Le module pathlib :

Combiner les chemins avec l'opérateur /:

```
base_path = Path("C:/Users/Youssef")
data_dir = Path("TP")

# Combiner plusieurs paths
file_path = base_path / data_dir / "prices.csv"

print(file_path) # C:/Users/Youssef/TP/prices.csv
```

Les modules de gestion des fichiers

Gestion des fichiers avec le module pathlib :

```
from pathlib import Path
path = Path("c:/notes.txt")
```

Propriétés	Description
path.name	Nom complet
path.stem	Nom sans extension
path.suffix	Extension du fichier
path.parent	Renvoie le dossier parent
path.parts	Diviser un path en ses composantes

Les modules de gestion des fichiers

Gestion des fichiers avec le module pathlib :

```
from pathlib import Path
f = Path("c:/tps/ia/notes.txt")

print(f.name)   # notes.txt

print(f.stem)   # notes

print(f.suffix) # .txt

print(f.parent) # c:/tps/ia
print(f.parent.parent) # c:/tps

print(f.parts) # ('c:\', 'tps', 'ia', 'notes.txt')
```

Les modules de gestion des fichiers

Exercice :

```python
from pathlib import Path

chemin = Path("documents/projets/python/rapport_final.pdf")
```

Écrire un programme Python qui affiche :

1. le nom du fichier
2. le nom du fichier sans extension
3. l'extension
4. le dossier parent
5. le dossier parent du dossier parent
6. Construire un nouveau chemin qui correspond au fichier :
documents/projets/python/backup/rapport_final.pdf

Les modules de gestion des fichiers

Exercice :

```
from pathlib import Path

chemin = Path("documents/projets/python/rapport_final.pdf")
```

1. le nom du fichier

```
print("Nom du fichier :", chemin.name)
```

2. le nom du fichier sans extension

```
print("Nom sans extension :", chemin.stem)
```

3. l'extension

```
print("Extension :", chemin.suffix)
```

4. le dossier parent

```
print("Dossier parent :", chemin.parent)
```

Les modules de gestion des fichiers

Exercice :

```
from pathlib import Path

chemin = Path("documents/projets/python/rapport_final.pdf")
```

6. le dossier parent du dossier parent

```
print("Parent du parent :", chemin.parent.parent)
```

7. Construire un nouveau chemin qui correspond au fichier :
documents/projets/python/backup/rapport_final.pdf

```
nouveau_chemin = chemin.parent / "backup" / chemin.name
print("Nouveau chemin :", nouveau_chemin)
```

Les modules de gestion des fichiers

Le module pathlib :

Le module **pathlib** permet de manipuler les chemins de fichiers et de dossiers en Python de manière simple et orientée objet.

Avec **pathlib**, on peut :

1. Manipuler des fichiers (lecture, écriture, vérification d'existence)
2. Manipuler des dossiers (création, navigation, liste de fichiers)
3. Construire et combiner des chemins de manière claire

Les modules de gestion des fichiers

Gestion des fichiers avec le module pathlib :

```
from pathlib import Path
path = Path("c:/notes.txt")
```

Méthodes	Description
path.exists()	Vérifier si un fichier existe ou non
path.is_file()	Vérifie si c'est un fichier
path.touch()	Crée un fichier vide
path.read_text()	Lit le contenu texte
path.write_text(text)	Écrit du texte
path.unlink()	Supprimer le fichier
path.rename(target)	Renommer ou déplacer un fichier

Les modules de gestion des fichiers

Gestion des fichiers avec le module pathlib :

Vérifie si le chemin existe :

```
from pathlib import Path
f = Path("c:/notes.txt")

if f.exists() :
    print("Le fichier existe")
else :
    print("Fichier introuvable!")
```

Les modules de gestion des fichiers

Gestion des fichiers avec le module pathlib :

Supprimer si le chemin est un fichier :

```
from pathlib import Path
f = Path("c:/notes.txt")

# Supprimer le fichier s'il existe
if f.exists() :
    f.unlink()
```

Les modules de gestion des fichiers

Gestion des fichiers avec le module pathlib :

Créer un fichier :

```
from pathlib import Path
f = Path("c:/notes.txt")

# Créer un fichier vide
f.touch()
```

Les modules de gestion des fichiers

Gestion des fichiers avec le module pathlib :

Ecrire un texte dans le fichier :

```
from pathlib import Path
f = Path("c:/notes.txt")

# Ecrire un texte
f.write_text("Bonjour je suis un texte!")
```

Les modules de gestion des fichiers

Gestion des fichiers avec le module pathlib :

Lire le contenu d'un texte :

```
from pathlib import Path
f = Path("c:/notes.txt")

# Lire un fichier
ch = f.read_text()

print(ch)
```

Les modules de gestion des fichiers

Gestion des fichiers avec le module pathlib :

Renommer un fichier :

```
from pathlib import Path
f = Path("c:/notes.txt")

# Renommer un fichier

f.rename("c:/moyennes.txt")
```

Les modules de gestion des fichiers

Gestion des fichiers avec le module pathlib :

Déplacer un fichier :

```
from pathlib import Path
f = Path("c:/notes.txt")

# Déplacer un fichier
if f.exists() :
      f.rename("c:/tp/moyennes.txt")
```

Les modules de gestion des fichiers

Gestion des dossiers avec le module pathlib :

```
from pathlib import Path
path = Path("c:/projets")
```

Fonction	Description
path.exists()	Vérifier si le dossier "projets" existe ou non
path.is_dir()	Vérifie si c'est un dossier
path.mkdir()	Crée un le dossier s'il n'existe pas
path.rmdir()	Supprimer le dossier
path.rename(target)	Renommer ou déplacer un dossier
path.iterdir()	Retourne un itérateur sur le contenu du dossier
path.glob(pattern)	Recherche avec motif dans le dossier (*.txt)
path.rglob(pattern)	Recherche récursive

Les modules de gestion des fichiers

Gestion des dossiers avec le module pathlib :

Vérifier si un dossier existe :

```
from pathlib import Path
dir = Path("c:/projets")

if dir.exists() :
    print('Le répertoire existe!')
else :
    print('Répertoire introuvable!')
```

Les modules de gestion des fichiers

Gestion des dossiers avec le module pathlib :

Crée un le dossier s'il n'existe pas:

```
from pathlib import Path
dir = Path("c:/tp/projets")

if not dir.exists() :
    dir.mkdir(parents=True)
```

Les modules de gestion des fichiers

Gestion des dossiers avec le module pathlib :

Supprimer le dossier :

```
from pathlib import Path
dir = Path("c:/tp/projets")

if dir.exists() :
    dir.rmdir()
```

NB : Le dossier ne sera supprimé que s'il est vide.

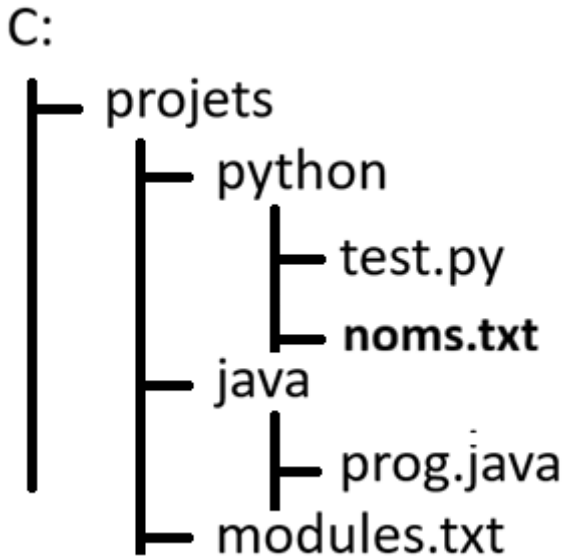
Les modules de gestion des fichiers

Gestion des dossiers avec le module pathlib :

Un itérateur sur le contenu du dossier :

```
from pathlib import Path
dir = Path("c:/projets")

for element in dir.iterdir():
    print(element)
```



Output :

```
python
java
modules.txt
```

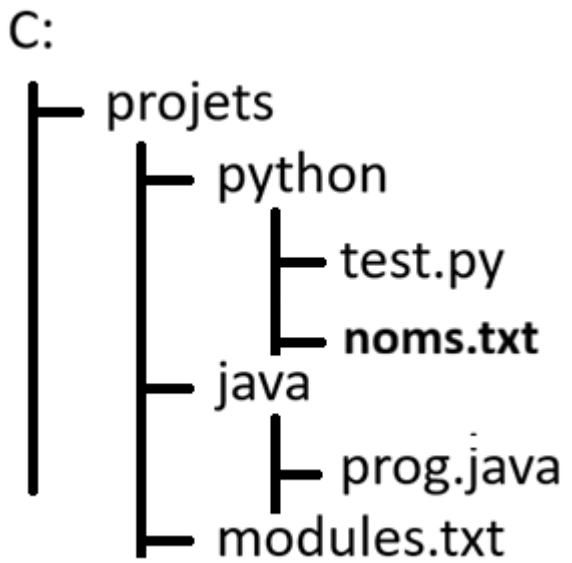
Les modules de gestion des fichiers

Gestion des dossiers avec le module pathlib :

Recherche avec motif dans le dossier (*.txt) :

```
from pathlib import Path
dir = Path("c:/projets")

for fichier in dir.glob("*.txt"):
    print(fichier)
```



Output :

```
modules.txt
```

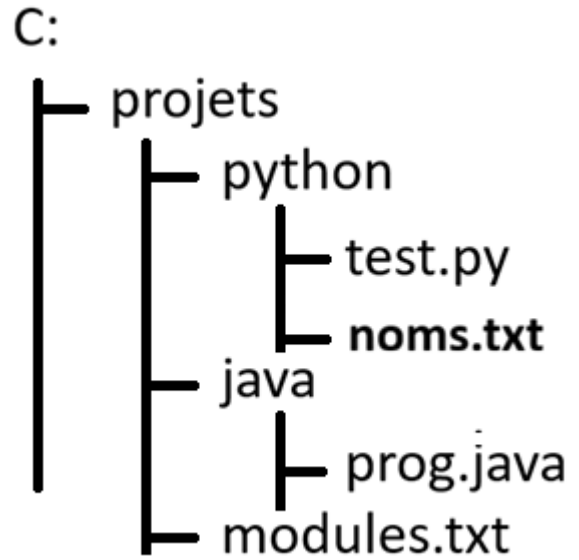
Les modules de gestion des fichiers

Gestion des dossiers avec le module pathlib :

chercher récursivement dans un dossier et tous ses sous-dossiers :

```
from pathlib import Path
dir = Path("c:/projets")

for fichier in dir.rglob("*.txt"):
    print(fichier)
```



Output :

```
modules.txt
noms.txt
```

Le module shutil

Les modules de gestion des fichiers

Le module shutil :

shutil est un module standard de Python qui fournit des fonctions de haut niveau pour manipuler des fichiers et des dossiers.

Il permet notamment de :

- Copier des fichiers et répertoires
- Déplacer des fichiers/dossiers
- Supprimer des dossiers complets
- Créer et extraire des archives (zip, tar)

👉 shutil et pathlib se complètent

Les modules de gestion des fichiers

Gestion des dossiers avec le module shutil :

```
import shutil
```

Méthode	Description
shutil.copytree(src, dst)	Copie un dossier entier avec tout son contenu
shutil.rmtree(path)	Supprime un dossier et tout son contenu (récursif)
shutil.move(src, dst)	Déplace un dossier vers un autre emplacement
shutil.make_archive(base_name, format, root_dir)	Crée une archive (zip, tar…) à partir d'un dossier
shutil.unpack_archive(filename, extract_dir)	Extrait une archive dans un dossier

Les modules de gestion des fichiers

Gestion des dossiers avec le module shutil :

Supprime un dossier et tout son contenu (récursif)

```
import shutil

# Copier le dossier "source_folder" vers "destination_folder"
shutil.copytree("c:/source_folder", "c:/tp/destination_folder")
```

Les modules de gestion des fichiers

Gestion des dossiers avec le module shutil :

Supprime un dossier et tout son contenu (récursif)

```
import shutil

# Supprime un dossier et tout son contenu (récursif) "projets"
shutil.rmtree("c:/tp/projets")
```

Les modules de gestion des fichiers

Gestion des dossiers avec le module shutil :

Déplace un dossier vers un autre emplacement :

```
import shutil

# Déplacer un fichier
shutil.move("c:/source.txt", "c:/tp")
```

Les modules de gestion des fichiers

Comparaison des modules pathlib et shutil :

Besoin	pathlib	shutil
Construire des chemins	☑ meilleur	✗
Lire/écrire un fichier	☑	✗
Parcourir dossiers	☑	✗
Copier un fichier	⚠ possible (à la main)	☑ simple
Copier un dossier entier	✗	☑ (copytree)
Supprimer dossier récursif	⚠ (rmdir seulement si vide)	☑ (rmtree)
ZIP / TAR	✗	☑