Département Génie Informatique et Mathématiques
Filière : DUT Informatique Décisionnelle et Science des Données

Polycopié de Cours : Programmation en Python

Préparé par :

Fatiha Bendaida

Année universitaire : **2021/2022**

Table des matières

1 Introduction au Python

Python est l'un des langages de programmation les plus populaires au monde. Selon le PYPL-Index, sa part est d'environ 25.95%, le plaçant devant Java, C#, PHP, et R.

1.1 Pourquoi Python ?

Python est un langage de programmation Open Source, orienté objet, de haut niveau. Il s'agit d'un langage généraliste. Cela signifie qu'il peut être utilisé pour développer à peu près n'importe quoi, grâce à de nombreux outils et bibliothèques.

Ce langage est particulièrement populaire pour :

— L'analyse de données et l'intelligence artificielle

— Le développement web backend

— Le computing scientifique

— Le développement d'outils de productivité, de jeux ou d'applications

1.2 Versions de Python

Il existe 2 versions majeures de Python :

— Python 2.x (obsolète depuis 2020)

— Python 3.x (version actuelle)

2 Les Bases de Python

2.1 Variables et Affectation

2.1.1 Définition

Une variable est un conteneur d'information identifié par un nom (identificateur) et ayant un contenu.

2.1.2 Règles pour les identificateurs

— Doivent commencer par une lettre ou le caractère _

— Ne peuvent contenir que des lettres, chiffres et/ou le caractère _

— Ne peuvent pas être un mot réservé de Python

— Sont sensibles à la casse (ma_var $\neq$ Ma_Var)

Exemples valides : toto, proch_val, max1, MA_VALEUR, r2d2, bb8, _mavar

Exemples non valides : 2be, C-3PO, ma var

2.1.3 Affectation

Pour mémoriser une valeur dans une variable, on utilise le signe = :

```
n = 33
a = 42 + 25
ch = "bonjour"
euro = 6.55957

```

En Python, le typage est dynamique :

```
a = 20
print(type(a))  # <class 'int'>
a = "salut"
print(type(a)) # <class 'str'>
a = 3.14
print(type(a)) # <class 'float'>

```

2.2 Opérateurs

2.2.1 Opérateurs arithmétiques

— + (addition)
— - (soustraction)
— * (multiplication)
— ** (puissance)
— / (division)
— % (modulo)
— // (division entière)

2.2.2 Opérateurs de comparaison

— == (égalité)
— != (différence)
— < (inférieur)
— > (supérieur)
— <= (inférieur ou égal)
— >= (supérieur ou égal)

2.2.3 Opérateurs logiques

— and (et)
— or (ou)
— not (non)

2.3 Structures Conditionnelles

2.3.1 Instruction if

```
if condition:
    # bloc d'instructions si vrai

```

2.3.2 Instruction if...else

```
if condition:
    # bloc si vrai
else:
    # bloc si faux

```

2.3.3 Instruction if...elif...else

```
if condition1:
    # bloc 1
elif condition2:
    # bloc 2
else:
    # bloc 3

```

2.4 Boucles

2.4.1 Boucle while

```
while condition:
    # bloc d'instructions

```

2.4.2 Boucle for

```
for element in sequence:
    # bloc d'instructions

```

Avec range() :

```
for i in range(5):      # 0  4
for i in range(2, 6):   # 2  5
for i in range(0, 10, 2): # 0, 2, 4, 6, 8

```

2.5 Fonctions

2.5.1 Définition

```
def nom_fonction(param1, param2):
    # instructions
    return resultat

```

2.5.2 Exemple

```
def somme(a, b):
    return a + b

result = somme(3, 5)  # result = 8

```

2.5.3 Fonctions anonymes (lambda)

```
carre = lambda x: x**2
print(carre(5))  # 25

```

3 Entrées/Sorties

3.1 La fonction input()

Permet de récupérer une saisie utilisateur :

```
nom = input("Entrez votre nom: ")
age = int(input("Entrez votre ge: "))

```

3.2 La fonction print()

Permet d'afficher du texte :

```
print("Bonjour", nom, "vous avez", age, "ans")
print(f"Bonjour {nom}, vous avez {age} ans")  # f-string

```

4 Exercices

4.1 Exercice 1 : Échange de variables

Écrire un programme qui échange le contenu de deux variables :

```
a = input("Donnez a: ")
b = input("Donnez b: ")
print("Avant:", a, b)
a, b = b, a  # change
print("Aprs:", a, b)

```

4.2 Exercice 2 : Calcul de moyenne

Calculer la moyenne de trois nombres :

```
a = float(input("Note 1: "))
b = float(input("Note 2: "))
c = float(input("Note 3: "))
moy = (a + b + c) / 3
print("Moyenne:", moy)

```

4.3 Exercice 3 : Conversion secondes

Convertir des secondes en heures, minutes, secondes :

```
sec = int(input("Secondes: "))
h = sec // 3600
m = (sec % 3600) // 60
s = (sec % 3600) % 60
print(f"{h}h {m}m {s}s")

```

5 Listes et Tuples

5.1 Listes

5.1.1 Définition

Les variables, telles que nous les avons vues, ne permettent de stocker qu'une seule donnée à la fois. Or, pour de nombreuses données, des variables distinctes seraient beaucoup trop lourdes à gérer. Heureusement, Python propose des structures de données permettant de stocker l'ensemble de ces données dans une variable commune. Ainsi, pour accéder à ces valeurs il suffit de parcourir la variable en utilisant les indices.

Une liste est une structure de données qui permet de stocker plusieurs valeurs de type hétérogène (simple ou composé).

5.1.2 Création

On peut construire une liste de plusieurs manières :

— Par la donnée explicite des éléments, entre crochets, séparés par des virgules :

```
L1 = []         # liste vide (ou bien L1 = list())
L2 = [1, 15, 20, 35] # liste des entiers
L3 = [True, 12, "hello", 12.2]

```

— Par concaténation de listes (à l'aide de +) :

```
L4 = [1, 2] + [3, 4] # L4 sera [1, 2, 3, 4]

```

— Par multiplication d'une liste par un entier :

```
L5 = [0] * 5 # L5 sera [0, 0, 0, 0, 0]

```

— À partir d'une chaîne de caractères (chaque caractère devient un élément) :

```
L6 = list("Python") # L6 sera ['P', 'y', 't', 'h', 'o', 'n']

```

5.1.3 Accès aux éléments

Indexation : Les éléments d'une liste sont indicés à partir de 0.

```
my_list = ['a', 'b', 'c', 'd', 'e']
print(my_list[0])  # Output: 'a'
print(my_list[2])  # Output: 'c'
print(my_list[-1]) # Output: 'e' (dernier lment)
```

Tranches (Slicing) : Permet d'extraire une sous-liste. La syntaxe est `list[start:end:step]`.

```
my_list = ['a', 'b', 'c', 'd', 'e', 'f']
print(my_list[1:4])  # Output: ['b', 'c', 'd']
print(my_list[:3])   # Output: ['a', 'b', 'c']
print(my_list[3:])   # Output: ['d', 'e', 'f']
print(my_list[::2])  # Output: ['a', 'c', 'e'] (tous les deux lments)
print(my_list[::-1]) # Output: ['f', 'e', 'd', 'c', 'b', 'a'] (liste inverse)
```

5.1.4 Modification des listes

Les listes sont mutables, ce qui signifie que leurs éléments peuvent être changés, ajoutés ou supprimés.

— **Modification d'un élément :**

```
my_list = [10, 20, 30]
my_list[1] = 25 # my_list devient [10, 25, 30]

```

— **Ajout d'éléments :**

— `append(x)` : Ajoute `x` à la fin de la liste.

```
L = [1, 2]
L.append(3) # L est maintenant [1, 2, 3]
```

— `extend(iterable)` : Ajoute tous les éléments d'un itérable à la fin de la liste.

```
L = [1, 2]
L.extend([3, 4]) # L est maintenant [1, 2, 3, 4]
```

— `insert(index, x)` : Insère `x` à la position spécifiée.

```
L = [1, 3, 4]
L.insert(1, 2) # L est maintenant [1, 2, 3, 4]
```

— **Suppression d'éléments :**

— `remove(x)` : Supprime la première occurrence de `x`.

```
L = [1, 2, 3, 2]
L.remove(2) # L est maintenant [1, 3, 2]
```

— `pop([index])` : Supprime et retourne l'élément à l'index donné. Si aucun index n'est donné, supprime et retourne le dernier élément.

```
L = [10, 20, 30]
popped_element = L.pop(1) # popped_element est 20, L est [10, 30]
last_element = L.pop()    # last_element est 30, L est [10]
```

— `del list[index]` : Supprime l'élément à l'index donné.

```
L = [1, 2, 3]
del L[1] # L est maintenant [1, 3]
```

— `clear()` : Supprime tous les éléments de la liste.

```
L = [1, 2, 3]
L.clear() # L est maintenant []
```

5.1.5 Opérations courantes et méthodes

— `len(list)` : Retourne le nombre d'éléments de la liste.

— `list.index(x[, start[, end]])` : Retourne l'index de la première occurrence de `x`.

— `list.count(x)` : Retourne le nombre d'occurrences de `x`.

— `list.sort()` : Trie les éléments de la liste sur place (modifie la liste originale).

— `sorted(list)` : Retourne une nouvelle liste triée (ne modifie pas la liste originale).

— `list.reverse()` : Inverse l'ordre des éléments de la liste sur place.

— `list.copy()` : Retourne une copie superficielle de la liste.

— `x in list` : Teste l'appartenance (retourne `True` si `x` est dans la liste, `False` sinon).

```
my_list = [3, 1, 4, 1, 5, 9, 2]
print(len(my_list))        # Output: 7
print(my_list.count(1))    # Output: 2
my_list.sort()
print(my_list)             # Output: [1, 1, 2, 3, 4, 5, 9]
print(2 in my_list)        # Output: True
```

5.1.6 Boucle sur une liste

Vous pouvez parcourir les éléments d'une liste en utilisant une boucle `for`.

```
my_list = ["apple", "banana", "cherry"]
for x in my_list:
print(x)
```

5.2 Tuples

5.2.1 Définition

Un tuple est une collection ordonnée et **immuable** (non modifiable) d'éléments. Une fois créé, les éléments d'un tuple ne peuvent pas être changés, ajoutés ou supprimés.

5.2.2 Création

Les tuples sont créés en plaçant tous les éléments entre parenthèses `()`, séparés par des virgules.

```
my_tuple = ("apple", "banana", "cherry")
empty_tuple = ()
single_element_tuple = (2,) # La virgule est obligatoire pour un tuple  un
  seul lment
```

Attention : Sans la virgule pour un seul élément, Python interpréterait '(2)' comme un entier entre parenthèses.

```
t = (2)
print(type(t)) # Output: <class 'int'>

t = (2,)
print(type(t)) # Output: <class 'tuple'>
```

5.2.3 Accès aux éléments

L'accès aux éléments d'un tuple se fait de la même manière que pour les listes, par indexation et tranchage.

```
t = ("apple", "banana", "cherry")
print(t[0])    # Output: "apple"
print(t[1:3])  # Output: ("banana", "cherry")
```

5.2.4 Immutabilité des Tuples

Contrairement aux listes, vous ne pouvez pas modifier un élément d'un tuple après sa création.

```
t = ("apple", "banana", "cherry")
# t[1] = "kiwi" # Cela provoquerait une erreur: TypeError: 'tuple' object does
   not support item assignment
```

Pour "modifier" un tuple, il faut en créer un nouveau à partir des éléments existants.

```
x = ("apple", "banana", "cherry")
x = x[:1] + ("Kiwi",) + x[2:]
print(x) # Output: ('apple', 'Kiwi', 'cherry')
```

5.2.5 Boucle à travers un tuple

Vous pouvez parcourir les éléments du tuple en utilisant une boucle `for`.

```
t = ("apple", "banana", "cherry")
for x in t:
print(x)
```

5.2.6 Test d'appartenance

Pour déterminer si un élément spécifié est présent dans un tuple, utilisez le mot clé `in`.

```
t = ("apple", "banana", "cherry")
print('kiwi' in t)   # Output: False
print('banana' in t) # Output: True
```

5.2.7 Longueur du tuple

Pour déterminer le nombre d'éléments d'un tuple, utilisez la méthode `len()`.

```
t = ("apple", "banana", "cherry")
print(len(t)) # Output: 3
```

5.2.8 Opérations sur les Tuples

— `len(U)` : Longueur du tuple `U`.
— `U + V` : Concaténation de deux tuples `U` et `V`.
— `n * U` ou `U * n` : Répète le tuple `U` `n` fois.

6 Chaînes de Caractères et Dictionnaires

6.1 Chaînes de Caractères

6.1.1 Définition

Une chaîne de caractères est une succession de caractères délimités par des guillemets (simples ' ou doubles ").

6.1.2 Création

```
s1 = str() # chane vide
s1 = ""    # chane vide
s2 = str("Hello") # s2 est "Hello"
s2 = "Hello"      # s2 est "Hello"
s3 = str(123) # s3 est "123"
s3 = '123'    # s3 est "123"
```

6.1.3 Caractères spéciaux

Certains caractères ont une signification spéciale lorsqu'ils sont précédés d'un anti-slash (
).

— `\n` : Permet d'insérer des marques de passage à la ligne (nouvelle ligne).

— `\t` : Permet d'insérer des marques de tabulation.

— `\b` : Retour arrière (suppression du caractère avant).

N.B : Même s'ils sont écrits avec deux symboles, les caractères spéciaux comptent pour un seul caractère.

6.1.4 Comparaison de deux chaînes de caractères

Les chaînes peuvent être comparées lexicographiquement.

```
print('green' == 'glow')  # Output: False
print('green' != 'Green') # Output: True (sensible  la casse)
```

6.1.5 Méthodes courantes des chaînes de caractères

Python fournit de nombreuses méthodes pour manipuler les chaînes. Voici quelques exemples :

— `upper()` : Retourne la chaîne en majuscules.

— `lower()` : Retourne la chaîne en minuscules.

— `strip()` : Supprime les espaces blancs (ou caractères spécifiés) du début et de la fin de la chaîne.

— `replace(old, new)` : Remplace toutes les occurrences de `old` par `new`.

— `split(separator)` : Divise la chaîne en une liste de sous-chaînes en utilisant le séparateur spécifié.

— `join(iterable)` : Concatène les éléments d'un itérable (généralement une liste de chaînes) en une seule chaîne, en utilisant la chaîne sur laquelle la méthode est appelée comme séparateur.
— `find(substring)` : Retourne l'index de la première occurrence de la sous-chaîne, ou -1 si non trouvée.
— `count(substring)` : Retourne le nombre d'occurrences de la sous-chaîne.
— `startswith(prefix)` : Vérifie si la chaîne commence par le préfixe.
— `endswith(suffix)` : Vérifie si la chaîne se termine par le suffixe.
— `isdigit()` : Retourne `True` si tous les caractères de la chaîne sont des chiffres et qu'il y a au moins un caractère, `False` sinon.
— `isalpha()` : Retourne `True` si tous les caractères de la chaîne sont alphabétiques, `False` sinon.
— `isalnum()` : Retourne `True` si tous les caractères de la chaîne sont alphanumériques, `False` sinon.

```
text = "  Hello World  "
print(text.strip())        # Output: "Hello World"
print(text.upper())        # Output: "  HELLO WORLD  "
print("hello".replace("l", "X")) # Output: "heXXo"
print("apple,banana,cherry".split(',')) # Output: ['apple', 'banana', 'cherry
  ']
print("-".join(["a", "b", "c"])) # Output: "a-b-c"
```

6.2 Dictionnaires

6.2.1 Définition

Un dictionnaire est une collection de paires clé-valeur, non ordonnée, modifiable et indexée. Chaque clé doit être unique. Les clés peuvent être de n'importe quel type de données immuable (chaînes, nombres, tuples), et les valeurs peuvent être de n'importe quel type de données.

6.2.2 Création

Les dictionnaires sont créés en utilisant des accolades `{}`.

```
my_dict = {"brand": "Ford", "model": "Mustang", "year": 1964}
empty_dict = {}
```

6.2.3 Accès aux valeurs

Les valeurs sont accédées en utilisant les clés entre crochets `[]`.

```
my_dict = {"brand": "Ford", "model": "Mustang"}
print(my_dict["model"]) # Output: "Mustang"
```

Alternativement, la méthode `get()` peut être utilisée, qui ne lève pas d'erreur si la clé n'existe pas, mais retourne `None` ou une valeur par défaut.

```
print(my_dict.get("year", "Not found")) # Output: "Not found"
```

6.2.4 Ajout, modification et suppression d'éléments

— **Ajout/Modification :** Si la clé existe, la valeur est modifiée ; sinon, une nouvelle paire clé-valeur est ajoutée.

```
my_dict = {"brand": "Ford", "model": "Mustang"}
my_dict["year"] = 2020 # Ajoute "year": 2020
my_dict["model"] = "F-150" # Modifie la valeur de "model"
print(my_dict)
# Output: {'brand': 'Ford', 'model': 'F-150', 'year': 2020}

```

— **Suppression :**

— `del dict[key]` : Supprime la paire clé-valeur spécifiée.

— `dict.pop(key)` : Supprime la paire clé-valeur et retourne la valeur associée.

— `dict.clear()` : Supprime tous les éléments du dictionnaire.

6.2.5 Boucle sur un dictionnaire

— Parcourir les clés :

```
for x in my_dict: # ou for x in my_dict.keys():
print(x) # Imprime les cls: brand, model, year

```

— Parcourir les valeurs :

```
for x in my_dict.values():
print(x) # Imprime les valeurs: Ford, F-150, 2020

```

— Parcourir les paires clé-valeur :

```
for key, value in my_dict.items():
print(f"{key}: {value}")

```

6.2.6 Méthodes courantes des dictionnaires

— `D.keys()` : Retourne une vue des clés du dictionnaire `D`.

— `D.values()` : Retourne une vue des valeurs stockées dans le dictionnaire `D`.

— `D.items()` : Retourne une vue des paires clé-valeur sous forme de tuples.

— `D.update(D2)` : Met à jour le dictionnaire `D` avec les paires clé-valeur du dictionnaire `D2`. Si une clé de `D2` existe déjà dans `D`, sa valeur est mise à jour.

6.2.7 Exercice sur les Dictionnaires

Exercice 2 : On considère un dictionnaire `service` déclaré d'une manière globale comme suit et qui représente l'âge et les jours de travail de chaque employé :

```
service = {
  'Ahmed': [30, ["Lu", "Ma", "Me", "Je"]],
  'Rachid': [25, ["Lu", "Me", "Di"]],
  'Omar': [55, ["Lu", "Me", "Ve"]],
  'Sara': [30, ["Ma", "Je", "Sa"]],
  'Jamal': [40, ["Lu", "Ma", "Me", "Ve"]]
}
```

1. Écrire une fonction `employes_par_age(age)` qui renvoie la liste des noms des employés âgés de plus ou égal à l'âge passé en paramètre.

```
def employes_par_age(age):
M = []
for x in service: # x est la cl (nom de l'employ)
if service[x][0] >= age: # service[x][0] est l'ge de l'employ
M.append(x)
return M

print(employes_par_age(30)) # Test
print(employes_par_age(50)) # Test

```

2. Écrire une fonction `employes_par_jour(jour)` qui renvoie la liste des noms des employés qui travaillent le jour passé en paramètre.

```
def employes_par_jour(jour):
L = []
for employe, details in service.items():
jours_travail = details[1] # dtails[1] est la liste des jours de
travail
if jour in jours_travail:
L.append(employe)
return L

print(employes_par_jour("Lu")) # Test
print(employes_par_jour("Di")) # Test

```

7 Fichiers sous Python

7.1 Introduction

Dans tous les langages de programmation, on utilise la notion de fichier, qui désigne un ensemble d'informations enregistrées sur un support (clé USB, disque dur, etc.). Dans la majorité des langages de programmation, on distingue deux types de fichiers :

— **Les fichiers textes :** Les informations sont sous un format texte (.txt, .docs, ...) qui est lisible par n'importe quel éditeur de texte.

— **Les fichiers binaires :** Les informations ne sont lisibles que par le programme qui les a conçues (.pdf, .jpg, ...).

7.2 Le module `os`

Le module `os` en Python fournit des fonctions d'interaction avec le système d'exploitation. Il permet d'effectuer des opérations courantes liées au système d'exploitation. Il est indépendant par rapport au système d'exploitation de la machine, ce qui signifie que ce module peut fonctionner sur n'importe quel système d'exploitation.

7.2.1 Opérations sur les fichiers et dossiers

— **Récupérer le répertoire de travail actuel :**

```
    import os
    current_directory = os.getcwd()
    print(f"Rpertoire de travail actuel: {current_directory}")

```

— **Changer le répertoire de travail :**

```
    import os
    # Assurez-vous que le dossier de destination existe, sinon une
    FileNotFoundError sera leve.
    new_directory = "/path/to/your/new/directory" # Remplacez par un chemin
     valide
    try:
    os.chdir(new_directory)
    print(f"Rpertoire de travail chang pour: {os.getcwd()}")
    except FileNotFoundError:
    print(f"Erreur: Le rpertoire '{new_directory}' n'existe pas.")

```

7.3 Ouverture et fermeture des fichiers

Pour travailler avec un fichier, la première étape est de l'ouvrir en utilisant la fonction intégrée `open()`.

```
# Ouverture d'un fichier en mode lecture ('r')
file_read = open("mon_fichier.txt", "r")

```

```
# Ouverture d'un fichier en mode criture ('w') - cre le fichier s'il n'existe pas, crase son contenu s'il existe
file_write = open("nouveau_fichier.txt", "w")

# Ouverture d'un fichier en mode ajout ('a') - cre le fichier s'il n'existe pas, ajoute  la fin s'il existe
file_append = open("log.txt", "a")

# N'oubliez pas de fermer le fichier une fois que vous avez termin de l'utiliser
file_read.close()
file_write.close()
file_append.close()
```

7.3.1 Modes d'ouverture

— `'r'` : Mode lecture (par défaut). Le fichier doit exister.

— `'w'` : Mode écriture. Crée un nouveau fichier s'il n'existe pas. Si le fichier existe, son contenu est tronqué (effacé).

— `'a'` : Mode ajout. Crée un nouveau fichier s'il n'existe pas. Si le fichier existe, les nouvelles données sont écrites à la fin.

— `'x'` : Mode création exclusive. Crée un nouveau fichier. Si le fichier existe déjà, l'opération échoue (lève une 'FileExistsError').

— `'+'` : Ouvrir un fichier pour la mise à jour (lecture et écriture). Peut être combiné avec d'autres modes (ex : `'r+'`, `'w+'`).

— `'b'` : Mode binaire. Utilisé pour les fichiers non textuels (images, vidéos). Doit être combiné avec un autre mode (ex : `'rb'`, `'wb'`).

— `'t'` : Mode texte (par défaut). Utilisé pour les fichiers textuels. (ex : `'rt'`, `'wt'`).

7.4 Écriture dans un fichier

— `file.write(string)` : Écrit la chaîne donnée dans le fichier. Ne retourne pas le nombre de caractères écrits.

```
    f = open("exemple_ecriture.txt", "w")
    f.write("Bonjour DWFS1,\n")
    f.write("ceci n'est pas\n")
    f.write("facile !!!\n")
    f.close()

```

— `file.writelines(list_of_strings)` : Écrit une liste de chaînes dans le fichier. Chaque chaîne doit contenir les caractères de fin de ligne si désiré.

```
    lines = ["Premire ligne\n", "Deuxime ligne\n", "Troisime ligne\n"]
    f = open("exemple_writelines.txt", "w")
    f.writelines(lines)
    f.close()

```

7.5 Lecture depuis un fichier

— `file.read([size])` : Lit tout le contenu du fichier comme une seule chaîne. Si `size` est spécifié, lit au maximum `size` octets/caractères.

```
    f = open("exemple_ecriture.txt", "r")
    content = f.read()
    print(content)
    f.close()

```

— `file.readline()` : Lit une seule ligne du fichier. Inclut le caractère de fin de ligne (\n).

```
    f = open("exemple_ecriture.txt", "r")
    line1 = f.readline()
    line2 = f.readline()
    print(line1, end='') # Utilisation de end='' pour viter un double saut de ligne
    print(line2, end='')
    f.close()

```

— `file.readlines()` : Retourne une liste contenant toutes les lignes du fichier. Chaque élément de la liste est une ligne du fichier, incluant le caractère de fin de ligne.

```
    f = open("exemple_ecriture.txt", "r")
    all_lines = f.readlines()
    print(all_lines)
    # Output: ['Bonjour DWFS1,\n', 'ceci nest pas\n', 'facile !!!\n']
    f.close()

```

7.6 Boucle sur un fichier

Un fichier texte ouvert en lecture possède également les propriétés d'un itérateur : on pourra alors parcourir les lignes du fichier par une boucle. C'est souvent la méthode la plus efficace pour lire un fichier ligne par ligne.

```
with open("exemple_ecriture.txt", "r") as f:
for line in f:
print(line, end='') # print() ajoute un '\n' par dfaut, donc end='' est utile
```

7.7 Ouverture du fichier avec `with`

De façon pragmatique, l'instruction `with` permet d'écrire un code sans utiliser explicitement l'instruction `close()`. Le fichier est automatiquement fermé, même si une erreur survient. C'est la méthode recommandée en Python.

Les deux bouts de code suivant sont équivalents :

Sans `with` :

```
f = open("data.txt", "w")
try:
f.write("Some data")
finally:
f.close()
```

Avec `with` (recommandé) :

```
with open("data.txt", "w") as f:
f.write("Some data")
# Le fichier est automatiquement ferm ici
```

7.8 Exercices sur les fichiers

Exercice 1 : Diviseurs

1. Écrire un programme qui permet de stocker dans un fichier nommé "diviseurs.txt" les diviseurs séparés par '#' d'un nombre N lu au clavier.

```
    N = int(input("Entrez un nombre entier N: "))
    diviseurs = []
    for i in range(1, N + 1):
    if N % i == 0:
    diviseurs.append(str(i)) # Convertir en chane pour la jointure

    diviseurs_str = "#".join(diviseurs)

    with open("diviseurs.txt", "w") as f:
    f.write(diviseurs_str)

    print(f"Les diviseurs de {N} ont t  stocks  dans diviseurs.txt")

```

2. Écrire un programme qui permet de lire ces nombres à partir du fichier "diviseurs.txt" et afficher leur somme.

```
    total_somme = 0
    try:
    with open("diviseurs.txt", "r") as f:
    content = f.read()
    diviseurs_str_list = content.split('#')

    for s_num in diviseurs_str_list:
    try:
    total_somme += int(s_num)
    except ValueError:
    print(f"Avertissement: '{s_num}' n'est pas un nombre valide et sera
    ignor.")

    print(f"La somme des diviseurs est: {total_somme}")
```

```
    except FileNotFoundError:
    print("Le fichier 'diviseurs.txt' n'a pas t trouv. Veuillez d'abord
    excuter le programme de stockage.")

```

Exercice 2 : Notes des étudiants On désire stocker les notes des étudiants d'une classe dans un fichier. Écrire un programme dans lequel l'utilisateur peut entrer le nom, le prénom et la note de chaque étudiant. Le programme doit demander à l'utilisateur s'il veut ajouter un autre étudiant. Chaque étudiant sera stocké sur une nouvelle ligne dans le fichier "notes.txt" avec le format "Nom Prénom : Note".

```
def ajouter_notes_etudiants():
with open("notes.txt", "a") as f:
while True:
nom = input("Entrez le nom de l'tudiant: ")
prenom = input("Entrez le prnom de l'tudiant: ")

while True:
try:
note = float(input(f"Entrez la note de {prenom} {nom}: "))
if 0 <= note <= 20:
break
else:
print("La note doit tre entre 0 et 20.")
except ValueError:
print("Note invalide. Veuillez entrer un nombre.")

f.write(f"{nom} {prenom}: {note}\n")
print(f"Note de {prenom} {nom} ({note}) ajoute.")

continuer = input("Voulez-vous ajouter un autre tudiant (oui/non)? ").lower()
if continuer != 'oui':
break
print("Enregistrement des notes termin.")

# Appeler la fonction pour lancer le programme
ajouter_notes_etudiants()

# Exemple de lecture du fichier aprs criture (pour vrification)
print("\nContenu du fichier notes.txt:")
try:
with open("notes.txt", "r") as f:
print(f.read())
except FileNotFoundError:
print("Le fichier notes.txt n'existe pas encore.")
```