Chapitre 1

Rappels sur la programmation en Python

Objectifs pédagogiques (Durée : 2 semaines)

Ce chapitre introductif vise à consolider les bases fondamentales de la programmation Python et à établir les concepts informatiques essentiels pour la suite du cours. À l'issue de ce chapitre, l'étudiant sera capable de :

— Comprendre l'architecture fondamentale des systèmes informatiques

— Maîtriser l'installation et la configuration d'un environnement de développement Python professionnel

— Manipuler les structures de données de base avec efficacité

— Implémenter des algorithmes simples en utilisant les structures de contrôle appropriées

— Distinguer les différents paradigmes de programmation et leurs applications

1.1 Introduction aux concepts informatiques fondamentaux

1.1.1 Architecture von Neumann et son impact sur la programmation

L'architecture von Neumann, fondement des ordinateurs modernes, repose sur quatre composants principaux illustrés dans la figure 1.1 :

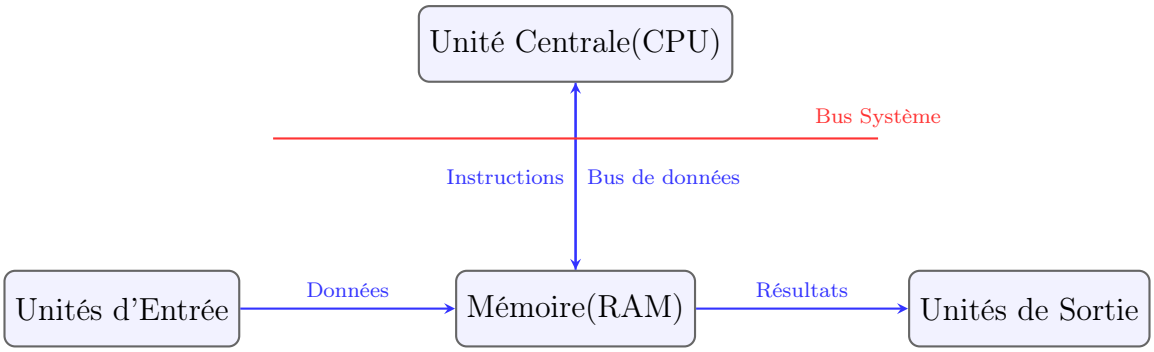

FIGURE 1.1 – Architecture von Neumann - Modèle conceptuel

Cette architecture implique que :

— Le **processeur (CPU)** lit les instructions et les données depuis la mémoire, exécute les opérations, puis renvoie les résultats à la mémoire ou aux unités de sortie.

— La **mémoire vive (RAM)** joue donc le rôle de médiateur entre le processeur et les unités d'entrée/sortie. Elle communique avec le CPU par le bus système.

— Les **périphériques d'entrée/sortie** permettent l'interaction avec l'environnement externe

— Le **bus système** assure la communication entre ces composants via trois sous-systèmes :
  — **Bus de données** : transfert des données entre composants
  — **Bus d'adresses** : sélection des emplacements mémoire
  — **Bus de contrôle** : gestion des signaux de synchronisation

1.1.2 Modèle d'exécution des programmes Python

L'exécution d'un programme Python suit un processus en plusieurs étapes :

1. **Écriture du code source** en langage Python (fichiers .py)
2. **Compilation en bytecode** par l'interpréteur Python (fichiers .pyc)
3. **Exécution dans la Python Virtual Machine (PVM)**
4. **Gestion mémoire automatique** par le garbage collector

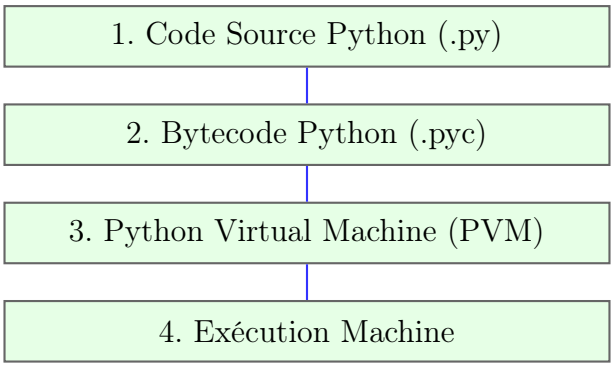

FIGURE 1.2 – Processus d'exécution d'un programme Python

1.1.3 Configuration d'un environnement optimisé

```
# Téléchargement de PyCharm Professional
# 1. Aller sur https://www.jetbrains.com/pycharm/download/
# 2. Télécharger la version Professional pour Windows

# Installation étape par étape :
# - Exécuter le fichier .exe téléchargé
# - Choisir le chemin d'installation (C:\Program Files\Jetbrains\PyCharm )
# - Cocher les options :
#       Create Desktop Shortcut
#       Update PATH variable
#       Update Context Menu
#       Create Associations (.py files)

# Configuration initiale de PyCharm :
# - Au premier lancement, choisir "Do not import settings"
# - Sélectionner le thème (Dark/Light) selon préférence
# - Installer les plugins essentiels :
#       Python
#       Scientific Mode
#       Jupyter
#       Git Integration

# Création d'un nouveau projet :
# - File     New Project
# - Emplacement : C:\Users\VotreNom\PycharmProjects\prog_avancee
# - Interpréteur Python : New environment using Virtualenv
# - Base interpreter : Python 3.11
# - Cocher     Make available to all projects

# Installation des bibliothèques essentielles :
# Méthode 1 : Via Terminal intégré de PyCharm
pip install numpy pandas matplotlib jupyter
pip install scikit-learn seaborn plotly openpyxl
pip install requests beautifulsoup4 tkinter

# Méthode 2 : Via l'interface PyCharm
# - File     Settings     Project     Python Interpreter
# - Cliquer sur '+' et rechercher les bibliothèques
```

Listing 1.1 – Installation de PyCharm sur Windows

```
# Vérification de l'installation
python --version
pip --version

# Création d'un environnement virtuel dédié (optionnel mais recommandé)
python -m venv C:\Users\VotreNom\venv\prog_avancee
# Activation de l'environnement
C:\Users\VotreNom\venv\prog_avancee\Scripts\activate

# Installation groupée des bibliothèques scientifiques
pip install numpy scipy pandas matplotlib seaborn
pip install jupyter notebook jupyterlab plotly
pip install scikit-learn tensorflow torch
pip install requests beautifulsoup4 selenium
pip install openpyxl xlrd pdfkit

# Configuration de Git pour le contrôle de version
git config --global user.name "Votre Nom"
git config --global user.email "votre.email@domain.com"
```

Listing 1.2 – Configuration avancée de l'environnement

1.1.4 Avantages de PyCharm pour le développement professionnel

PyCharm Professional offre des fonctionnalités avancées essentielles pour ce cours :

— **Debugger intégré** : Points d'arrêt, exécution pas à pas, inspection des variables

— **Completion intelligente** : Auto-complétion contextuelle du code Python

— **Gestion des environnements** : Support natif des environnements virtuels

— **Intégration Jupyter** : Édition et exécution de notebooks dans l'IDE

— **Outils de refactoring** : Renommage sécurisé, extraction de méthodes

— **Analyse de code** : Détection des erreurs et suggestions d'amélioration

— **Intégration Git** : Gestion visuelle des branches et commits

— **Support des bases de données** : Outils intégrés pour SQL et NoSQL

— **Testing intégré** : Exécution et débogage des tests unitaires

— **Support Docker** : Développement et déploiement conteneurisé

TABLE 1.1 – Comparaison PyCharm Community vs Professional

Fonctionnalité	**PyCharm Community**	**PyCharm Professional**
Coût	Gratuit	Payant (gratuit pour étudiants)
Support Python	Complet	Complet
Support Scientific	Limitré	Complet (Jupyter, NumPy, etc.)
Support Web	Basique	Avancé (Django, Flask)
Support Bases de données	Non	Complet
Support Remote Development	Non	Complet
Licence étudiante	Non applicable	Gratuite via JetBrains Student Pack

Activation de la licence étudiante

Les étudiants peuvent obtenir une licence gratuite de PyCharm Professional :

1. Se rendre sur `https://www.jetbrains.com/student/`
2. Cliquer sur "Apply for free student licenses"
3. Utiliser son email académique pour vérification
4. Créer un compte JetBrains et confirmer l'email
5. Télécharger PyCharm Professional et activer avec le compte

1.2 Programmation de base en Python

1.2.1 Mode interactif et mode script

Python offre deux modes d'exécution principaux :

Mode interactif

Le mode interactif permet d'exécuter du code ligne par ligne dans l'interpréteur Python :

```
# Dans le terminal ou l'invite de commande Python
>>> x = 10
>>> y = 20
>>> print(x + y)
30
>>> nom = input("Entrez votre nom: ")
Entrez votre nom: Alice
>>> print(f"Bonjour {nom}!")
Bonjour Alice!
```

Listing 1.3 – Utilisation du mode interactif

Avantages du mode interactif :

— Test rapide d'instructions simples

— Exploration des fonctions et méthodes

— Débogage interactif

— Apprentissage et expérimentation

Mode script

Le mode script permet d'exécuter des programmes complets sauvegardés dans des fichiers :

```
# Fichier : mon_script.py
# Ceci est un commentaire
print("Début du programme")

# Définition de variables
nom = "Alice"
age = 25
temperature = 23.5

# Opérations de base
resultat = age * 2 + 10
print(f"Résultat du calcul: {resultat}")

# Structure conditionnelle
if age >= 18:
    print(f"{nom} est majeur(e)")
else:
    print(f"{nom} est mineur(e)")

print("Fin du programme")
```

Listing 1.4 – Exemple de script Python

Exécution d'un script :

```
# Dans le terminal
python mon_script.py
```

1.2.2 Variables, types de données et opérateurs

Variables et affectation

```python
# Affectation de variables
nom = "Pierre"          # Chaîne de caractères
age = 30                # Entier (integer)
taille = 1.75           # Nombre à virgule (float)
est_etudiant = True     # Booléen
valeur_nulle = None     # Valeur nulle

# Affichage des variables
print(f"Nom: {nom}, Type: {type(nom)}")
print(f"Age: {age}, Type: {type(age)}")
print(f"Taille: {taille}, Type: {type(taille)}")
print(f"Est étudiant: {est_etudiant}, Type: {type(est_etudiant)}")
```

Listing 1.5 – Déclaration et utilisation des variables

Types de données fondamentaux

TABLE 1.2 – Types de données de base en Python

Type	Description	Exemple	Usage
int	Entier	42, -15, 0	Calculs mathématiques
float	Nombre flottant	3.14, -2.5, 0.0	Calculs scientifiques
str	Chaîne de caractères	"Bonjour", 'Python'	Texte, messages
bool	Booléen	True, False	Conditions, flags
NoneType	Valeur nulle	None	Absence de valeur

Opérateurs en Python

```python
# Opérateurs arithmétiques
a = 10 + 5    # Addition -> 15
b = 10 - 5    # Soustraction -> 5
c = 10 * 5    # Multiplication -> 50
d = 10 / 3    # Division -> 3.333...
e = 10 // 3   # Division entière -> 3
f = 10 % 3    # Modulo -> 1
g = 2 ** 3    # Exponentiation -> 8

# Opérateurs de comparaison
x = 10
print(x == 5)   # Égal à -> False
print(x != 5)   # Différent de -> True
print(x > 5)    # Supérieur à -> True
print(x < 5)    # Inférieur à -> False
print(x >= 10)  # Supérieur ou égal -> True
print(x <= 10)  # Inférieur ou égal -> True

# Opérateurs logiques
```

```python
age = 20
est_majeur = age >= 18
a_un_permis = True

print(est_majeur and a_un_permis)   # ET logique
print(est_majeur or a_un_permis)    # OU logique
print(not est_majeur)               # NON logique
```

Listing 1.6 – Opérateurs mathématiques et logiques

1.2.3 Structures conditionnelles et boucles

Structures conditionnelles

```python
# Structure if simple
temperature = 25
if temperature > 30:
    print("Il fait chaud")

# Structure if-else
age = 17
if age >= 18:
    print("Accès autorisé")
else:
    print("Accès refusé")

# Structure if-elif-else
note = 14
if note >= 16:
    print("Très bien")
elif note >= 14:
    print("Bien")
elif note >= 12:
    print("Assez bien")
else:
    print("Insuffisant")

# Conditions multiples
age = 25
salaire = 30000
if age >= 18 and salaire > 20000:
    print("Éligible au prêt")
```

Listing 1.7 – Structures conditionnelles if/elif/else

Boucles for

```python
# Boucle for avec range
print("Compte de 0 à 4:")
for i in range(5):
    print(i)

print("Compte de 2 à 8 par pas de 2:")
```

```python
for i in range(2, 10, 2):
    print(i)

# Boucle for avec liste
fruits = ["pomme", "banane", "orange"]
print("Liste des fruits:")
for fruit in fruits:
    print(f"- {fruit}")

# Boucle for avec énumération
for index, fruit in enumerate(fruits):
    print(f"Fruit {index + 1}: {fruit}")
```

Listing 1.8 – Boucles for avec range et collections

Boucles while

```python
# Boucle while simple
compteur = 0
while compteur < 5:
    print(f"Compteur: {compteur}")
    compteur += 1  # Incrémentation importante !

# Boucle while avec condition complexe
reponse = ""
while reponse.lower() != "quit":
    reponse = input("Tapez 'quit' pour quitter: ")
    print(f"Vous avez tapé: {reponse}")

# Boucle while avec break
while True:
    nombre = int(input("Entrez un nombre (0 pour quitter): "))
    if nombre == 0:
        break
    print(f"Le carré de {nombre} est {nombre ** 2}")
```

Listing 1.9 – Boucles while avec condition

1.3 Fonctions et éléments essentiels

1.3.1 Fonctions prédéfinies et création de fonctions

Fonctions prédéfinies courantes

```python
# Fonctions de conversion de type
nombre_texte = "123"
nombre = int(nombre_texte)      # Conversion en entier
decimal = float("3.14")         # Conversion en flottant
texte = str(42)                 # Conversion en chaîne

# Fonctions mathématiques
```

```python
valeurs = [5, 2, 8, 1, 9]
maximum = max(valeurs)          # Maximum -> 9
minimum = min(valeurs)          # Minimum -> 1
somme = sum(valeurs)            # Somme -> 25

# Fonctions sur les séquences
longueur = len("Python")        # Longueur -> 6
liste_range = list(range(5))    # Conversion en liste

# Fonctions d'entrée/sortie
nom = input("Entrez votre nom: ")
print(f"Bonjour {nom}!")
```

Listing 1.10 – Fonctions intégrées de Python

Création de fonctions personnalisées

```python
# Fonction simple sans paramètre
def dire_bonjour():
    print("Bonjour !")

# Fonction avec paramètre
def saluer_personne(nom):
    print(f"Bonjour {nom} !")

# Fonction avec paramètre optionnel
def presenter(nom, age=0):
    if age > 0:
        print(f"Je m'appelle {nom} et j'ai {age} ans.")
    else:
        print(f"Je m'appelle {nom}.")

# Fonction avec valeur de retour
def calculer_carre(x):
    return x ** 2

# Fonction avec multiple valeurs de retour
def operations_base(a, b):
    somme = a + b
    difference = a - b
    produit = a * b
    return somme, difference, produit

# Utilisation des fonctions
dire_bonjour()
saluer_personne("Alice")
presenter("Bob")
presenter("Charlie", 25)
resultat = calculer_carre(5)
print(f"5 au carré: {resultat}")

s, d, p = operations_base(10, 3)
print(f"Somme: {s}, Différence: {d}, Produit: {p}")
```

Listing 1.11 – Définition et utilisation de fonctions

1.3.2 Modules standards (math, random)

Module math

```python
import math

# Constantes mathématiques
print(f"Pi: {math.pi}")
print(f"Exponentielle: {math.e}")

# Fonctions trigonométriques
angle_rad = math.radians(45)  # Conversion degrés -> radians
sinus = math.sin(angle_rad)
cosinus = math.cos(angle_rad)
print(f"Sin(45  ): {sinus:.2f}")

# Fonctions exponentielles et logarithmiques
exponentielle = math.exp(2)       # e^2
logarithme = math.log(100, 10)    # log10(100)
racine_carree = math.sqrt(25)     #   25

# Arrondis et valeurs absolues
arrondi_sup = math.ceil(3.2)      # Arrondi supérieur -> 4
arrondi_inf = math.floor(3.8)     # Arrondi inférieur -> 3
valeur_absolue = math.fabs(-5)    # Valeur absolue -> 5.0

print(f"Racine carrée de 25: {racine_carree}")
```

Listing 1.12 – Utilisation du module math

Module random

```python
import random

# Génération de nombres aléatoires
nombre_aleatoire = random.random()         # [0.0, 1.0)
entier_aleatoire = random.randint(1, 6)    # Entier entre 1 et 6 inclus
flottant_aleatoire = random.uniform(1.5, 5.5)  # Flottant dans l'
    intervalle

print(f"Nombre aléatoire: {nombre_aleatoire}")
print(f"Dé à 6 faces: {entier_aleatoire}")

# Opérations sur les séquences
liste = [1, 2, 3, 4, 5]
element_aleatoire = random.choice(liste)       # Choix aléatoire
liste_melangee = random.sample(liste, 3)       # Échantillon sans remise
random.shuffle(liste)                          # Mélange en place

print(f"Élément aléatoire: {element_aleatoire}")
print(f"Échantillon de 3: {liste_melangee}")
print(f"Liste mélangée: {liste}")
```

Listing 1.13 – Utilisation du module random

1.3.3 Chaînes de caractères et manipulation de texte

```python
# Création de chaînes
chaine1 = "Bonjour"
chaine2 = 'Python'
chaine3 = """Ceci est
une chaîne
multi-lignes"""

# Opérations de base
longueur = len(chaine1)                    # Longueur
concatenation = chaine1 + " " + chaine2    # Concaténation
repetition = chaine1 * 3                   # Répétition

# Méthodes des chaînes
texte = "  Hello World!  "
minuscules = texte.lower()                 # En minuscules
majuscules = texte.upper()                 # En majuscules
sans_espaces = texte.strip()               # Supprime espaces
remplacement = texte.replace("World", "Python")  # Remplacement

# Formatage de chaînes
nom = "Alice"
age = 25
message1 = "Je m'appelle {} et j'ai {} ans.".format(nom, age)
message2 = f"Je m'appelle {nom} et j'ai {age} ans."  # f-string

print(message1)
print(message2)

# Découpage (slicing)
chaine = "Python Programming"
premiers_caracteres = chaine[0:6]          # "Python"
derniers_caracteres = chaine[-11:]         # "Programming"
inverse = chaine[::-1]                     # Inverse la chaîne
```

Listing 1.14 – Manipulation des chaînes de caractères

1.4 Manipulation des fichiers et structures de données

1.4.1 Manipulation de fichiers

```python
# Écriture dans un fichier
with open("mon_fichier.txt", "w", encoding="utf-8") as fichier:
    fichier.write("Ligne 1\n")
    fichier.write("Ligne 2\n")
    fichier.write("Ligne 3\n")

# Lecture complète d'un fichier
with open("mon_fichier.txt", "r", encoding="utf-8") as fichier:
    contenu = fichier.read()
    print("Contenu complet:")
    print(contenu)

```

```python
# Lecture ligne par ligne
with open("mon_fichier.txt", "r", encoding="utf-8") as fichier:
    print("Ligne par ligne:")
    for ligne in fichier:
        print(ligne.strip())  # strip() enlève les sauts de ligne

# Ajout à un fichier existant
with open("mon_fichier.txt", "a", encoding="utf-8") as fichier:
    fichier.write("Nouvelle ligne ajoutée\n")
```

Listing 1.15 – Lecture et écriture de fichiers

1.4.2 Listes, tuples et dictionnaires

Listes

```python
# Création de listes
nombres = [1, 2, 3, 4, 5]
fruits = ["pomme", "banane", "orange"]
mixte = [1, "texte", 3.14, True]

# Accès aux éléments
premier = nombres[0]        # 1
dernier = nombres[-1]       # 5
sous_liste = nombres[1:4]   # [2, 3, 4]

# Modification
nombres.append(6)           # Ajout en fin [1,2,3,4,5,6]
nombres.insert(2, 99)       # Insertion à l'index 2
nombres.remove(3)           # Supprime la première occurrence de 3
element_supprime = nombres.pop()  # Supprime et retourne le dernier

# Opérations
longueur = len(nombres)
tri = sorted(nombres)       # Nouvelle liste triée
nombres.sort()              # Tri en place
inverse = nombres.reverse() # Inverse l'ordre

print(f"Liste: {nombres}")
print(f"Longueur: {longueur}")
```

Listing 1.16 – Manipulation des listes

Tuples

```python
# Création de tuples
coordonnees = (48.8566, 2.3522)        # Coordonnées géographiques
couleur_rgb = (255, 0, 0)              # Couleur rouge
personne = ("Alice", 25, "Paris")      # Information personne

# Les tuples sont immuables
print(f"Coordonnées: {coordonnees}")
print(f"Latitude: {coordonnees[0]}")
```

```python
print(f"Longitude: {coordonnees[1]}")

# Déballage (unpacking)
nom, age, ville = personne
print(f"Nom: {nom}, Age: {age}, Ville: {ville}")

# Utilisation comme clés de dictionnaire (car immuables)
localisations = {
    (48.8566, 2.3522): "Paris",
    (40.7128, -74.0060): "New York"
}
```

Listing 1.17 – Utilisation des tuples

Dictionnaires

```python
# Création de dictionnaires
etudiant = {
    "nom": "Alice",
    "age": 20,
    "cours": ["Math", "Python"],
    "note": 15.5
}

# Accès aux valeurs
nom = etudiant["nom"]                    # Accès direct
age = etudiant.get("age")                # Accès sécurisé
ville = etudiant.get("ville", "Inconnu") # Valeur par défaut

# Modification
etudiant["age"] = 21                     # Modification
etudiant["ville"] = "Paris"              # Ajout
note_supprimee = etudiant.pop("note")    # Suppression

# Parcours
print("Clés:")
for cle in etudiant.keys():
    print(f"- {cle}")

print("Valeurs:")
for valeur in etudiant.values():
    print(f"- {valeur}")

print("Éléments:")
for cle, valeur in etudiant.items():
    print(f"{cle}: {valeur}")
```

Listing 1.18 – Manipulation des dictionnaires

1.5 Exercices pratiques

1.5.1 Exercices d'apprentissage de Python

1. **Calculatrice simple** : Écrire un programme qui demande deux nombres et une opération (+, -, *, /) à l'utilisateur, puis affiche le résultat.

2. **Gestionnaire de contacts** : Créer un programme qui permet d'ajouter, afficher et rechercher des contacts (nom, téléphone, email) en utilisant un dictionnaire.

3. **Analyse de texte** : Écrire une fonction qui prend un texte en paramètre et retourne le nombre de mots, de caractères et la fréquence de chaque mot.

4. **Jeu de devinette** : Programme où l'ordinateur choisit un nombre aléatoire entre 1 et 100, et l'utilisateur doit le deviner en recevant des indications "plus grand" ou "plus petit".

5. **Gestion des notes** : Créer un système qui permet de saisir les notes d'étudiants, calculer les moyennes et afficher les étudiants ayant la meilleure et la pire note.

1.5.2 Exercices d'utilisation des bibliothèques

1. **Calculs scientifiques avec math** : Écrire un programme qui résout une équation du second degré en utilisant le module math.

2. **Simulation avec random** : Créer un simulateur de lancers de dés qui calcule les statistiques sur 1000 lancers.

3. **Manipulation de données avec listes** : Implémenter des fonctions pour trier, filtrer et transformer des listes de nombres.

4. **Gestion de fichiers** : Écrire un programme qui lit un fichier CSV, traite les données et écrit les résultats dans un nouveau fichier.

5. **Utilisation de NumPy et Pandas** : Manipuler des tableaux de données numériques avec NumPy et effectuer des analyses basiques avec Pandas.

1.5.3 Exercices intégrateurs

```
class Bibliotheque:
    def __init__(self):
        self.livres = {}  # Dictionnaire: ISBN -> informations livre
        self.emprunts = []  # Liste des emprunts en cours
```

```
5
6       def ajouter_livre(self, isbn, titre, auteur, annee):
7           """Ajoute un livre à la bibliothèque"""
8           if isbn in self.livres:
9               print(f"Le livre avec ISBN {isbn} existe déjà")
10              return False
11
12          self.livres[isbn] = {
13              'titre': titre,
14              'auteur': auteur,
15              'annee': annee,
16              'disponible': True
17          }
18          print(f"Livre '{titre}' ajouté avec succès")
19          return True
20
21      def emprunter_livre(self, isbn, emprunteur):
22          """Permet d'emprunter un livre"""
23          if isbn not in self.livres:
24              print("Livre non trouvé")
25              return False
26
27          livre = self.livres[isbn]
28          if not livre['disponible']:
29              print("Livre déjà emprunté")
30              return False
31
32          livre['disponible'] = False
33          self.emprunts.append({
34              'isbn': isbn,
35              'emprunteur': emprunteur,
36              'date_emprunt': '2024-01-01'  # En pratique, utiliser
    datetime
37          })
38          print(f"Livre '{livre['titre']}' emprunté par {emprunteur}")
39          return True
40
41      def rechercher_livre(self, critere, valeur):
42          """Recherche un livre par titre ou auteur"""
43          resultats = []
44          for isbn, livre in self.livres.items():
45              if valeur.lower() in livre[critere].lower():
46                  resultats.append((isbn, livre))
47          return resultats
48
49      def statistiques(self):
50          """Affiche les statistiques de la bibliothèque"""
51          total_livres = len(self.livres)
52          livres_disponibles = sum(1 for livre in self.livres.values()
53                                   if livre['disponible'])
54          livres_empruntes = total_livres - livres_disponibles
55
56          print(f"Statistiques de la bibliothèque:")
57          print(f"Total livres: {total_livres}")
58          print(f"Livres disponibles: {livres_disponibles}")
59          print(f"Livres empruntés: {livres_empruntes}")
60
61  # Utilisation du système
```

```
if __name__ == "__main__":
    biblio = Bibliotheque()

    # Ajout de livres
    biblio.ajouter_livre("123", "Python pour débutants", "A. Dupont",
    2023)
    biblio.ajouter_livre("456", "Algorithmes avancés", "M. Martin",
    2022)

    # Recherche et emprunt
    resultats = biblio.rechercher_livre("auteur", "dupont")
    if resultats:
        biblio.emprunter_livre("123", "Alice")

    biblio.statistiques()
```

Listing 1.19 – Exercice intégrateur : Système de bibliothèque

1.6 Conclusion et synthèse du chapitre

1.6.1 Points clés à retenir

— **Environnement de développement** : PyCharm offre un environnement professionnel pour le développement Python

— **Programmation de base** : Maîtrise des variables, structures de contrôle et fonctions

— **Structures de données** : Compréhension des listes, tuples et dictionnaires et de leurs cas d'usage

— **Manipulation de fichiers** : Capacité à lire et écrire des fichiers texte

— **Modules standards** : Utilisation efficace des modules math et random

1.6.2 Perspectives pour la suite

Les concepts maîtrisés dans ce chapitre constituent le fondement nécessaire pour aborder les sujets avancés des chapitres suivants :

— Programmation orientée objet et design patterns

— Automatisation et manipulation de fichiers avancée

— Analyse de données et visualisation

— Introduction au machine learning

1.6.3 Évaluation des compétences

Les exercices proposés permettent d'évaluer :

— La compréhension des concepts fondamentaux de Python

— La capacité à résoudre des problèmes simples avec les structures appropriées

— L'utilisation correcte des fonctions et modules standards

— La mise en œuvre de solutions structurées et maintenables