

Solutions des Exercices Avancés Python (14-16)

Ce document contient les solutions complètes des exercices avancés sur les fichiers, les exceptions et la POO.

Solution Exercice 14 : Gestionnaire de Journal

```
"""
```

Gestionnaire de Journal Intime Électronique

Permet de créer, lire et rechercher des entrées de journal

```
"""
```

```
import json
```

```
import os
```

```
from datetime import datetime
```

```
class GestionnaireJournal:
```

```
    def __init__(self, fichier="journal.json"):
```

```
        """
```

Initialise le gestionnaire de journal.

Args:

fichier (str): Nom du fichier JSON pour sauvegarder les entrées

```
        """
```

```
        self.fichier = fichier
```

```
        self.donnees = self.charger_journal()
```

```
    def charger_journal(self):
```

```
        """
```

Charge les entrées depuis le fichier JSON.

Returns:

dict: Dictionnaire contenant les entrées du journal

"""

```
if not os.path.exists(self.fichier):
```

```
    return {"entrees": []}
```

```
try:
```

```
    with open(self.fichier, "r", encoding="utf-8") as f:
```

```
        return json.load(f)
```

```
except json.JSONDecodeError:
```

```
    print("⚠ Fichier corrompu. Création d'un nouveau journal.")
```

```
    return {"entrees": []}
```

```
except Exception as e:
```

```
    print(f"⚠ Erreur lors du chargement : {e}")
```

```
    return {"entrees": []}
```

```
def sauvegarder_journal(self):
```

```
    """Sauvegarde les entrées dans le fichier JSON."""
```

```
    try:
```

```
        with open(self.fichier, "w", encoding="utf-8") as f:
```

```
            json.dump(self.donnees, f, indent=2, ensure_ascii=False)
```

```
        return True
```

```
    except Exception as e:
```

```
        print(f"⚠ Erreur lors de la sauvegarde : {e}")
```

```
        return False
```

```
def ajouter_entree(self, titre, contenu, tags=None):
```

"""

Ajoute une nouvelle entrée au journal.

Args:

titre (str): Titre de l'entrée

contenu (str): Contenu de l'entrée

tags (list): Liste de tags (mots-clés)

Returns:

bool: True si succès

"""

if tags is None:

tags = []

Générer un nouvel ID

if self.donnees["entrees"]:

nouvel_id = max(entree["id"] for entree in self.donnees["entrees"]) + 1

else:

nouvel_id = 1

Créer l'entrée

entree = {

"id": nouvel_id,

"date": datetime.now().isoformat(),

"titre": titre,

"contenu": contenu,

"tags": tags

}

```
self.donnees["entrees"].append(entree)
```

```
if self.sauvegarder_journal():
```

```
    print(f"✓ Entrée #{nouvel_id} ajoutée avec succès")
```

```
    return True
```

```
return False
```

```
def lire_toutes_entrees(self):
```

```
    """Affiche toutes les entrées du journal."""
```

```
    if not self.donnees["entrees"]:
```

```
        print("△ Aucune entrée dans le journal")
```

```
    return
```

```
print("\n" + "="*80)
```

```
print("TOUTES LES ENTRÉES".center(80))
```

```
print("="*80)
```

```
for entree in self.donnees["entrees"]:
```

```
    self._afficher_entree(entree)
```

```
def _afficher_entree(self, entree):
```

```
    """
```

```
    Affiche une entrée de manière formatée.
```

```
    Args:
```

```
        entree (dict): Entrée à afficher
```

```
    """
```

```
date = datetime.fromisoformat(entree["date"])
date_formatee = date.strftime("%d/%m/%Y à %H:%M")
```

```
print(f"\n--- Entrée #{entree['id']} ---")
print(f"Date : {date_formatee}")
print(f"Titre : {entree['titre']}")
print(f"Tags : {' '.join(entree['tags']) if entree['tags'] else 'Aucun'}")
print(f"\n{entree['contenu']}")
print("-"*80)
```

```
def rechercher_par_date(self, date_str):
```

```
    """
```

```
    Recherche des entrées par date.
```

```
    Args:
```

```
        date_str (str): Date au format JJ/MM/AAAA
```

```
    """
```

```
    try:
```

```
        date_recherche = datetime.strptime(date_str, "%d/%m/%Y").date()
```

```
        entrees_trouvees = []
```

```
        for entree in self.donnees["entrees"]:
```

```
            date_entree = datetime.fromisoformat(entree["date"]).date()
```

```
            if date_entree == date_recherche:
```

```
                entrees_trouvees.append(entree)
```

```
        if entrees_trouvees:
```

```
            print(f"\n✓ {len(entrees_trouvees)} entrée(s) trouvée(s) pour le {date_str}")
```

```

        for entree in entrees_trouvees:

            self._afficher_entree(entree)

        else:

            print(f"⚠ Aucune entrée trouvée pour le {date_str}")

    except ValueError:

        print("⚠ Format de date invalide. Utilisez JJ/MM/AAAA")

def rechercher_par_tag(self, tag):
    """
    Recherche des entrées par tag.

    Args:
        tag (str): Tag à rechercher
    """
    tag_lower = tag.lower()
    entrees_trouvees = [
        entree for entree in self.donnees["entrees"]
        if any(t.lower() == tag_lower for t in entree["tags"])
    ]

    if entrees_trouvees:
        print(f"\n✓ {len(entrees_trouvees)} entrée(s) avec le tag '{tag}'")
        for entree in entrees_trouvees:
            self._afficher_entree(entree)
    else:
        print(f"⚠ Aucune entrée avec le tag '{tag}'")

```

```

def rechercher_par_mot_cle(self, mot_cle):
    """
    Recherche des entrées contenant un mot-clé.

    Args:
        mot_cle (str): Mot-clé à rechercher
    """
    mot_cle_lower = mot_cle.lower()
    entrees_trouvees = [
        entree for entree in self.donnees["entrees"]
        if mot_cle_lower in entree["titre"].lower() or
        mot_cle_lower in entree["contenu"].lower()
    ]

    if entrees_trouvees:
        print(f"\n✓ {len(entrees_trouvees)} entrée(s) contenant '{mot_cle}'")
        for entree in entrees_trouvees:
            self._afficher_entree(entree)
    else:
        print(f"⚠ Aucune entrée contenant '{mot_cle}'")

def afficher_statistiques(self):
    """Affiche les statistiques du journal."""
    nb_entrees = len(self.donnees["entrees"])

    if nb_entrees == 0:
        print("⚠ Aucune statistique disponible")
        return

```

```
# Tags les plus utilisés
```

```
compteur_tags = {}
```

```
for entree in self.donnees["entrees"]:
```

```
    for tag in entree["tags"]:
```

```
        compteur_tags[tag] = compteur_tags.get(tag, 0) + 1
```

```
tags_tries = sorted(compteur_tags.items(), key=lambda x: x[1], reverse=True)
```

```
# Entrées par mois
```

```
entrees_par_mois = {}
```

```
for entree in self.donnees["entrees"]:
```

```
    date = datetime.fromisoformat(entree["date"])
```

```
    mois = date.strftime("%Y-%m")
```

```
    entrees_par_mois[mois] = entrees_par_mois.get(mois, 0) + 1
```

```
# Affichage
```

```
print("\n" + "="*60)
```

```
print("STATISTIQUES DU JOURNAL".center(60))
```

```
print("="*60)
```

```
print(f"\nNombre total d'entrées : {nb_entrees}")
```

```
if tags_tries:
```

```
    print("\n--- Tags les plus utilisés ---")
```

```
    for tag, compte in tags_tries[:5]:
```

```
        print(f" {tag:20} : {compte} fois")
```



```
print("\n--- Entrées par mois ---")
```

```
for mois, compte in sorted(entrees_par_mois.items(), reverse=True):
```

```
    date_mois = datetime.strptime(mois, "%Y-%m")
```

```
    mois_format = date_mois.strftime("%B %Y")
```

```
    print(f" {mois_format:20} : {compte} entrée(s)")
```

```
print("="*60)
```

```
def exporter_texte(self, fichier_sortie="journal_export.txt"):
```

```
    """
```

Exporte toutes les entrées dans un fichier texte formaté.

Args:

fichier_sortie (str): Nom du fichier de sortie

```
    """
```

```
try:
```

```
    with open(fichier_sortie, "w", encoding="utf-8") as f:
```

```
        f.write("="*80 + "\n")
```

```
        f.write("MON JOURNAL INTIME\n".center(80))
```

```
        f.write(f"Exporté le {datetime.now().strftime('%d/%m/%Y à %H:%M')}\n".center(80))
```

```
        f.write("="*80 + "\n\n")
```

```
    for entree in self.donnees["entrees"]:
```

```
        date = datetime.fromisoformat(entree["date"])
```

```
        date_formatee = date.strftime("%d/%m/%Y à %H:%M")
```

```
        f.write("\n" + "-"*80 + "\n")
```

```
f.write(f"Entrée #{entree['id']} - {date_formatee}\n")
```

```
f.write(f"Titre : {entree['titre']}\n")
```

```
if entree['tags']:
```

```
    f.write(f"Tags : {' '.join(entree['tags'])}\n")
```

```
f.write("-"*80 + "\n\n")
```

```
f.write(entree['contenu'] + "\n")
```

```
f.write("\n" + "="*80 + "\n")
```

```
f.write(f"FIN DU JOURNAL - {len(self.donnees['entrees'])} entrées\n".center(80))
```

```
f.write("="*80 + "\n")
```

```
print(f"✓ Journal exporté vers '{fichier_sortie}')
```

```
return True
```

```
except Exception as e:
```

```
    print(f"⚠ Erreur lors de l'export : {e}")
```

```
    return False
```

```
def afficher_menu():
```

```
    """Affiche le menu principal."""
```

```
    print("\n" + "="*60)
```

```
    print("JOURNAL INTIME ÉLECTRONIQUE".center(60))
```

```
    print("="*60)
```

```
    print("1. Ajouter une entrée")
```

```
    print("2. Lire toutes les entrées")
```

```
    print("3. Rechercher par date")
```

```
    print("4. Rechercher par tag")
```

```
print("5. Rechercher par mot-clé")
print("6. Afficher les statistiques")
print("7. Exporter en fichier texte")
print("8. Quitter")
print("="*60)
```

```
def main():
```

```
    """Fonction principale."""
```

```
    journal = GestionnaireJournal()
```

```
    print("📅 Bienvenue dans votre Journal Intime Électronique")
```

```
    print(f"Date : {datetime.now().strftime('%d/%m/%Y %H:%M')}")
```

```
    while True:
```

```
        afficher_menu()
```

```
        choix = input("\nVotre choix : ").strip()
```

```
        if choix == "1":
```

```
            print("\n--- NOUVELLE ENTRÉE ---")
```

```
            titre = input("Titre : ")
```

```
            print("Contenu (terminez par une ligne vide) :")
```

```
            lignes = []
```

```
            while True:
```

```
                ligne = input()
```

```
                if ligne == "":
```

```
                    break
```

```
            lignes.append(ligne)
```

```
contenu = "\n".join(lignes)
```

```
tags_str = input("Tags (séparés par des virgules) : ")
```

```
tags = [tag.strip() for tag in tags_str.split(",") if tag.strip()]
```

```
journal.ajouter_entree(titre, contenu, tags)
```

```
elif choix == "2":
```

```
    journal.lire_toutes_entrees()
```

```
elif choix == "3":
```

```
    date = input("Entrez la date (JJ/MM/AAAA) : ")
```

```
    journal.rechercher_par_date(date)
```

```
elif choix == "4":
```

```
    tag = input("Entrez le tag à rechercher : ")
```

```
    journal.rechercher_par_tag(tag)
```

```
elif choix == "5":
```

```
    mot_cle = input("Entrez le mot-clé à rechercher : ")
```

```
    journal.rechercher_par_mot_cle(mot_cle)
```

```
elif choix == "6":
```

```
    journal.afficher_statistiques()
```

```
elif choix == "7":
```

```
    nom_fichier = input("Nom du fichier (défaut: journal_export.txt) : ").strip()
```

```
    if not nom_fichier:
```

```

        nom_fichier = "journal_export.txt"
        journal.exporter_texte(nom_fichier)

elif choix == "8":
    print("\n 📅 À bientôt dans votre journal !")
    break

else:
    print("⚠ Choix invalide")

if choix != "8":
    input("\nAppuyez sur Entrée pour continuer...")

if __name__ == "__main__":
    main()

```

Solution Exercice 15 : Gestion Bancaire Robuste

```

"""

```

Système de Gestion de Compte Bancaire avec Gestion d'Erreurs

```

"""

```

```

import json

```

```

import os

```

```

from datetime import datetime

```

```

# Exceptions personnalisées

```

```

class SoldeInsuffisantException(Exception):

```

```
"""Exception levée quand le solde est insuffisant pour une opération."""
```

```
pass
```

```
class MontantInvalideException(Exception):
```

```
    """Exception levée quand un montant est invalide (négatif ou zéro)."""
```

```
    pass
```

```
class CompteInexistantException(Exception):
```

```
    """Exception levée quand on tente d'accéder à un compte inexistant."""
```

```
    pass
```

```
class CompteBancaire:
```

```
    """Représente un compte bancaire."""
```

```
    def __init__(self, titulaire, numero_compte, solde_initial=0):
```

```
        """
```

```
        Initialise un compte bancaire.
```

```
        Args:
```

```
            titulaire (str): Nom du titulaire
```

```
            numero_compte (str): Numéro unique du compte
```

```
            solde_initial (float): Solde initial (défaut: 0)
```

```
        Raises:
```

```
            MontantInvalideException: Si le solde initial est négatif
```

```
        """
```

```
        if solde_initial < 0:
```

```
raise MontantInvalideException("Le solde initial ne peut pas être négatif")
```

```
self.titulaire = titulaire
```

```
self.numero_compte = numero_compte
```

```
self.__solde = solde_initial
```

```
self.historique = []
```

```
self._journaliser("Création du compte", solde_initial)
```

```
def _journaliser(self, operation, montant):
```

```
    """
```

```
    Enregistre une opération dans l'historique.
```

```
    Args:
```

```
        operation (str): Type d'opération
```

```
        montant (float): Montant de l'opération
```

```
    """
```

```
    entree = {
```

```
        "date": datetime.now().isoformat(),
```

```
        "operation": operation,
```

```
        "montant": montant,
```

```
        "solde_apres": self.__solde
```

```
    }
```

```
    self.historique.append(entree)
```

```
def deposer(self, montant):
```

```
    """
```

```
    Dépose de l'argent sur le compte.
```

Args:

montant (float): Montant à déposer

Returns:

float: Nouveau solde

Raises:

MontantInvalideException: Si le montant est négatif ou zéro

"""

if montant <= 0:

raise MontantInvalideException(

f"Le montant doit être positif (reçu : {montant})"

)

self.__solde += montant

self._journaliser("Dépôt", montant)

print(f"✓ Dépôt de {montant:.2f}€ effectué")

return self.__solde

def retirer(self, montant):

"""

Retire de l'argent du compte.

Args:

montant (float): Montant à retirer

Returns:

float: Nouveau solde

Raises:

MontantInvalideException: Si le montant est négatif ou zéro

SoldeInsuffisantException: Si le solde est insuffisant

"""

if montant <= 0:

```
    raise MontantInvalideException(
        f"Le montant doit être positif (reçu : {montant})"
    )
```

if montant > self.__solde:

```
    raise SoldeInsuffisantException(
        f"Solde insuffisant. Disponible : {self.__solde:.2f}€, "
        f"Demandé : {montant:.2f}€"
    )
```

self.__solde -= montant

self._journaliser("Retrait", -montant)

print(f"✓ Retrait de {montant:.2f}€ effectué")

return self.__solde

def transferer(self, montant, compte_destinataire):

"""

Transfère de l'argent vers un autre compte.

Args:

montant (float): Montant à transférer

compte_destinataire (CompteBancaire): Compte destinataire

Returns:

tuple: (nouveau_solde_source, nouveau_solde_destination)

Raises:

MontantInvalideException: Si le montant est invalide

SoldeInsuffisantException: Si le solde est insuffisant

TypeError: Si le destinataire n'est pas un CompteBancaire

"""

if not isinstance(compte_destinataire, CompteBancaire):

raise TypeError("Le destinataire doit être un CompteBancaire")

if montant <= 0:

raise MontantInvalideException("Le montant doit être positif")

if montant > self.__solde:

raise SoldeInsuffisantException(

f"Solde insuffisant pour le transfert. "

f"Disponible : {self.__solde:.2f}€"

)

Effectuer le transfert

self.__solde -= montant

self._journaliser(

f"Transfert vers {compte_destinataire.numero_compte}",

-montant

)

```
compte_destinataire.deposer(montant)

compte_destinataire._journaliser(
    f"Transfert depuis {self.numero_compte}",
    montant
)
```

```
print(f"✓ Transfert de {montant:.2f}€ effectué vers "
      f"{compte_destinataire.titulaire}")
```

```
return self.__solde, compte_destinataire.obtenir_solde()
```

```
def obtenir_solde(self):
```

```
    """
```

```
    Retourne le solde actuel.
```

```
    Returns:
```

```
        float: Solde actuel
```

```
    """
```

```
    return self.__solde
```

```
def afficher_solde(self):
```

```
    """Affiche le solde de manière sécurisée."""
```

```
    print(f"\n{'='*50}")
```

```
    print(f"Compte : {self.numero_compte}")
```

```
    print(f"Titulaire : {self.titulaire}")
```

```
    print(f"Solde actuel : {self.__solde:.2f}€")
```

```
    print(f"{'='*50}")
```

```

def afficher_historique(self, nb_derniers=10):
    """
    Affiche l'historique des transactions.

    Args:
        nb_derniers (int): Nombre de dernières transactions à afficher
    """
    if not self.historique:
        print("△ Aucune transaction dans l'historique")
        return

    print(f"\n{' '*80}")
    print(f"HISTORIQUE - {self.numero_compte}".center(80))
    print(f"{' '*80}")
    print(f"{'Date':<20} {'Opération':<30} {'Montant':<15} {'Solde':<15}")
    print(f"{' '*80}")

    for entree in self.historique[-nb_derniers:]:
        date = datetime.fromisoformat(entree["date"])
        date_str = date.strftime("%d/%m/%Y %H:%M")
        montant_str = f"{entree['montant']:+.2f}€"
        solde_str = f"{entree['solde_apres']:.2f}€"

        print(f"{date_str:<20} {entree['operation']:<30} "
              f"{montant_str:<15} {solde_str:<15}")

    print(f"{' '*80}")

```

```
def to_dict(self):
```

```
    """
```

```
    Convertit le compte en dictionnaire pour sauvegarde.
```

```
    Returns:
```

```
        dict: Représentation du compte
```

```
    """
```

```
    return {
```

```
        "titulaire": self.titulaire,
```

```
        "numero_compte": self.numero_compte,
```

```
        "solde": self.__solde,
```

```
        "historique": self.historique
```

```
    }
```

```
@classmethod
```

```
def from_dict(cls, data):
```

```
    """
```

```
    Crée un compte depuis un dictionnaire.
```

```
    Args:
```

```
        data (dict): Données du compte
```

```
    Returns:
```

```
        CompteBancaire: Instance du compte
```

```
    """
```

```
    compte = cls(
```

```
        data["titulaire"],
```

```
        data["numero_compte"],
```

```
        data["solde"]
    )
    compte.historique = data.get("historique", [])
    return compte
```

```
class GestionnaireBancaire:
```

```
    """Gère plusieurs comptes bancaires."""
```

```
    def __init__(self, fichier="comptes.json", fichier_log="banque.log"):
```

```
        """
```

```
        Initialise le gestionnaire.
```

```
        Args:
```

```
            fichier (str): Fichier de sauvegarde des comptes
```

```
            fichier_log (str): Fichier de journalisation des erreurs
```

```
        """
```

```
        self.fichier = fichier
```

```
        self.fichier_log = fichier_log
```

```
        self.comptes = {}
```

```
        self.charger_comptes()
```

```
    def _log_erreur(self, message):
```

```
        """
```

```
        Journalise une erreur.
```

```
        Args:
```

```
            message (str): Message d'erreur
```

```
"""
```

```
try:
```

```
    with open(self.fichier_log, "a", encoding="utf-8") as f:
```

```
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

```
        f.write(f"[{timestamp}] {message}\n")
```

```
except Exception as e:
```

```
    print(f"⚠ Impossible d'écrire dans le log : {e}")
```

```
def charger_comptes(self):
```

```
    """Charge les comptes depuis le fichier."""
```

```
    if not os.path.exists(self.fichier):
```

```
        print("❗ Aucun fichier de comptes existant. Démarrage à zéro.")
```

```
        return
```

```
try:
```

```
    with open(self.fichier, "r", encoding="utf-8") as f:
```

```
        donnees = json.load(f)
```

```
    for numero, data in donnees.items():
```

```
        self.comptes[numero] = CompteBancaire.from_dict(data)
```

```
    print(f"✓ {len(self.comptes)} compte(s) chargé(s)")
```

```
except json.JSONDecodeError as e:
```

```
    self._log_erreur(f"Fichier JSON corrompu : {e}")
```

```
    print("⚠ Fichier de comptes corrompu. Démarrage à zéro.")
```

```
except Exception as e:
```

```
self._log_erreur(f"Erreur de chargement : {e}")  
print(f"⚠ Erreur lors du chargement : {e}")
```

```
def sauvegarder_comptes(self):
```

```
    """Sauvegarde tous les comptes."""
```

```
    try:
```

```
        donnees = {  
            numero: compte.to_dict()  
            for numero, compte in self.comptes.items()  
        }
```

```
        with open(self.fichier, "w", encoding="utf-8") as f:
```

```
            json.dump(donnees, f, indent=2, ensure_ascii=False)
```

```
        return True
```

```
    except Exception as e:
```

```
        self._log_erreur(f"Erreur de sauvegarde : {e}")
```

```
        print(f"⚠ Erreur lors de la sauvegarde : {e}")
```

```
        return False
```

```
def creer_compte(self, titulaire, numero_compte, solde_initial=0):
```

```
    """
```

```
    Crée un nouveau compte.
```

```
    Args:
```

```
        titulaire (str): Nom du titulaire
```

```
        numero_compte (str): Numéro du compte
```


solde_initial (float): Solde initial

Returns:

CompteBancaire: Le compte créé ou None

"""

try:

if numero_compte in self.comptes:

print(f"⚠ Le compte {numero_compte} existe déjà")

return None

compte = CompteBancaire(titulaire, numero_compte, solde_initial)

self.comptes[numero_compte] = compte

self.sauvegarder_comptes()

print(f"✓ Compte {numero_compte} créé pour {titulaire}")

return compte

except MontantInvalideException as e:

print(f"⚠ Erreur : {e}")

self._log_erreur(f"Création compte échouée : {e}")

return None

except Exception as e:

print(f"⚠ Erreur inattendue : {e}")

self._log_erreur(f"Erreur création compte : {e}")

return None

def obtenir_compte(self, numero_compte):

"""

Obtient un compte par son numéro.

Args:

numero_compte (str): Numéro du compte

Returns:

CompteBancaire: Le compte

Raises:

CompteInexistantException: Si le compte n'existe pas

"""

```
if numero_compte not in self.comptes:
```

```
    raise CompteInexistantException(
```

```
        f"Le compte {numero_compte} n'existe pas"
```

```
    )
```

```
return self.comptes[numero_compte]
```

```
def lister_comptes(self):
```

```
    """Liste tous les comptes."""
```

```
    if not self.comptes:
```

```
        print("⚠ Aucun compte dans la base")
```

```
    return
```

```
print(f"\n{' '*70}")
```

```
print("LISTE DES COMPTES".center(70))
```

```
print(f"{' '*70}")
```

```
print(f"{'Numéro':<15}{'Titulaire':<25}{'Solde':<20}")
```

```
print("-"*70)
```

```
for numero, compte in self.comptes.items():
```

```
    solde = compte.obtenir_solde()
```

```
    print(f"{numero:<15}{compte.titulaire:<25}{solde:>18.2f}€")
```

```
print("="*70)
```

```
def afficher_menu():
```

```
    """Affiche le menu principal."""
```

```
    print("\n" + "="*60)
```

```
    print("SYSTÈME BANCAIRE".center(60))
```

```
    print("="*60)
```

```
    print("1. Créer un compte")
```

```
    print("2. Déposer de l'argent")
```

```
    print("3. Retirer de l'argent")
```

```
    print("4. Transférer de l'argent")
```

```
    print("5. Afficher le solde")
```

```
    print("6. Afficher l'historique")
```

```
    print("7. Lister tous les comptes")
```

```
    print("8. Quitter")
```

```
    print("="*60)
```

```
def main():
```

```
    """Fonction principale."""
```

```
    gestionnaire = GestionnaireBancaire()
```

```
print("🏦 Bienvenue dans le Système de Gestion Bancaire")
```

```
while True:
```

```
    afficher_menu()
```

```
    choix = input("\nVotre choix : ").strip()
```

```
    try:
```

```
        if choix == "1":
```

```
            # Créer un compte
```

```
            titulaire = input("Nom du titulaire : ")
```

```
            numero = input("Numéro de compte : ")
```

```
            solde_str = input("Solde initial (0 par défaut) : ").strip()
```

```
            solde = float(solde_str) if solde_str else 0
```

```
            gestionnaire.creer_compte(titulaire, numero, solde)
```

```
        elif choix == "2":
```

```
            # Déposer
```

```
            numero = input("Numéro de compte : ")
```

```
            compte = gestionnaire.obtenir_compte(numero)
```

```
            montant = float(input("Montant à déposer : "))
```

```
            compte.deposer(montant)
```

```
            compte.afficher_solde()
```

```
            gestionnaire.sauvegarder_comptes()
```

```
        elif choix == "3":
```

```
# Retirer
```

```
numero = input("Numéro de compte : ")
```

```
compte = gestionnaire.obtenir_compte(numero)
```

```
montant = float(input("Montant à retirer : "))
```

```
compte.retirer(montant)
```

```
compte.afficher_solde()
```

```
gestionnaire.sauvegarder_comptes()
```

```
elif choix == "4":
```

```
# Transférer
```

```
numero_source = input("Compte source : ")
```

```
numero_dest = input("Compte destination : ")
```

```
montant = float(input("Montant à transférer : "))
```

```
compte_source = gestionnaire.obtenir_compte(numero_source)
```

```
compte_dest = gestionnaire.obtenir_compte(numero_dest)
```

```
compte_source.transférer(montant, compte_dest)
```

```
gestionnaire.sauvegarder_comptes()
```

```
elif choix == "5":
```

```
# Afficher solde
```

```
numero = input("Numéro de compte : ")
```

```
compte = gestionnaire.obtenir_compte(numero)
```

```
compte.afficher_solde()
```

```
elif choix == "6":
```

```

# Afficher historique

numero = input("Numéro de compte : ")
compte = gestionnaire.obtenir_compte(numero)
nb = input("Nombre de transactions (10 par défaut) : ").strip()
nb = int(nb) if nb else 10
compte.afficher_historique(nb)

elif choix == "7":
    # Lister comptes
    gestionnaire.lister_comptes()

elif choix == "8":
    print("\n 🏠 Au revoir et merci de votre confiance !")
    break

else:
    print("⚠️ Choix invalide")

except ComptelInexistantException as e:
    print(f"\n⚠️ {e}")

except SoldeInsuffisantException as e:
    print(f"\n⚠️ {e}")

except MontantInvalideException as e:
    print(f"\n⚠️ {e}")

except ValueError:

```

```

print("\n⚠ Veuillez entrer un montant valide")

except Exception as e:

    print(f"\n⚠ Erreur inattendue : {e}")

    gestionnaire._log_erreur(f"Erreur inattendue : {e}")

if choix != "8":

    input("\nAppuyez sur Entrée pour continuer...")

if __name__ == "__main__":

    main()

```

Solution Exercice 16 : Système de Bibliothèque

(Cette solution est très longue. Je vais créer la structure principale avec les classes essentielles)

```

"""

```

Système de Gestion de Bibliothèque - POO Complète

```

"""

```

```

import json

```

```

import os

```

```

from datetime import datetime

```

```

# Exceptions personnalisées

```

```

class LivreNonDisponibleException(Exception):

```

```

    """Exception levée quand un livre n'est pas disponible."""

```

```

    pass

```

```
class MembreNonInscritException(Exception):

    """Exception levée quand un membre n'est pas inscrit."""

    pass


class Livre:

    """Représente un livre dans la bibliothèque."""

    def __init__(self, titre, auteur, isbn, annee):

        """

        Initialise un livre.

        Args:

            titre (str): Titre du livre

            auteur (str): Auteur du livre

            isbn (str): Numéro ISBN unique

            annee (int): Année de publication

        """

        self.titre = titre

        self.auteur = auteur

        self.isbn = isbn

        self.annee = annee

        self.disponible = True

    def afficher_info(self):

        """Affiche les informations du livre."""

        statut = "✓ Disponible" if self.disponible else "X Emprunté"
```



```
print(f"\n--- {self.titre} ---")
print(f"Auteur : {self.auteur}")
print(f"ISBN : {self.isbn}")
print(f"Année : {self.annee}")
print(f"Statut : {statut}")
```

```
def emprunter(self):
```

```
    """
```

```
    Emprunte le livre.
```

```
    Raises:
```

```
        LivreNonDisponibleException: Si le livre n'est pas disponible
```

```
    """
```

```
    if not self.disponible:
```

```
        raise LivreNonDisponibleException(
```

```
            f"Le livre '{self.titre}' n'est pas disponible"
```

```
        )
```

```
    self.disponible = False
```

```
def retourner(self):
```

```
    """Retourne le livre à la bibliothèque."""
```

```
    self.disponible = True
```

```
def __str__(self):
```

```
    """Représentation en chaîne pour l'utilisateur."""
```

```
    return f"{self.titre} par {self.auteur}"
```

```
def __repr__(self):
```

```
"""Représentation pour les développeurs."""  
return f"Livre(titre='{self.titre}', isbn='{self.isbn}')
```

```
def to_dict(self):  
    """Convertit en dictionnaire pour sauvegarde."""  
    return {  
        "type": "Livre",  
        "titre": self.titre,  
        "auteur": self.auteur,  
        "isbn": self.isbn,  
        "annee": self.annee,  
        "disponible": self.disponible  
    }
```

```
@classmethod
```

```
def from_dict(cls, data):  
    """Crée un livre depuis un dictionnaire."""  
    livre = cls(data["titre"], data["auteur"], data["isbn"], data["annee"])  
    livre.disponible = data.get("disponible", True)  
    return livre
```

```
class LivreNumerique(Livre):
```

```
    """Représente un livre numérique."""
```

```
def __init__(self, titre, auteur, isbn, annee, taille_fichier_mo):
```

```
    """
```

```
    Initialise un livre numérique.
```

Args:

titre (str): Titre

auteur (str): Auteur

isbn (str): ISBN

annee (int): Année

taille_fichier_mo (float): Taille du fichier en Mo

"""

```
super().__init__(titre, auteur, isbn, annee)
```

```
self.taille_fichier_mo = taille_fichier_mo
```

```
def afficher_info(self):
```

```
    """Affiche les informations (redéfinition)."""
```

```
    super().afficher_info()
```

```
    print(f"Type : Livre numérique")
```

```
    print(f"Taille : {self.taille_fichier_mo} Mo")
```

```
def telecharger(self):
```

```
    """Simule le téléchargement du livre."""
```

```
    if not self.disponible:
```

```
        raise LivreNonDisponibleException("Livre non disponible")
```

```
    print(f" 📄 Téléchargement de '{self.titre}' ({self.taille_fichier_mo} Mo)...")
```

```
    print("✓ Téléchargement terminé !")
```

```
def to_dict(self):
```

```
    """Convertit en dictionnaire."""
```

```
    data = super().to_dict()
```

```
    data["type"] = "LivreNumerique"
```

```
data["taille_fichier_mo"] = self.taille_fichier_mo  
return data
```

```
@classmethod  
def from_dict(cls, data):  
    """Crée depuis un dictionnaire."""  
    livre = cls(  
        data["titre"], data["auteur"], data["isbn"],  
        data["annee"], data["taille_fichier_mo"]  
    )  
    livre.disponible = data.get("disponible", True)  
    return livre
```

```
class Membre:
```

```
    """Représente un membre de la bibliothèque."""
```

```
    def __init__(self, nom, numero_membre):
```

```
        """
```

```
        Initialise un membre.
```

```
        Args:
```

```
            nom (str): Nom du membre
```

```
            numero_membre (str): Numéro unique du membre
```

```
        """
```

```
        self.nom = nom
```

```
        self.numero_membre = numero_membre
```

```
        self.livres_empruntes = []
```

```

def emprunter_livre(self, livre):
    """
    Emprunte un livre.

    Args:
        livre (Livre): Livre à emprunter

    Raises:
        LivreNonDisponibleException: Si livre non disponible
    """
    livre.emprunter()
    self.livres_empruntees.append({
        "livre": livre,
        "date_emprunt": datetime.now().isoformat()
    })
    print(f"✓ {self.nom} a emprunté {livre}")

def retourner_livre(self, livre):
    """
    Retourne un livre.

    Args:
        livre (Livre): Livre à retourner
    """
    # Trouver le livre dans les emprunts
    for emprunt in self.livres_empruntees:
        if emprunt["livre"].isbn == livre.isbn:

```

```
livre.retourner()

self.livres_empruntes.remove(emprunt)

print(f"✓ {self.nom} a retourné {livre}")

return
```

```
print(f"△ {self.nom} n'a pas emprunté ce livre")
```

```
def afficher_livres(self):
```

```
    """Affiche les livres empruntés."""
```

```
    if not self.livres_empruntes:
```

```
        print(f"\n{self.nom} n'a emprunté aucun livre")
```

```
    return
```

```
print(f"\n=== Livres empruntés par {self.nom} ===")
```

```
for emprunt in self.livres_empruntes:
```

```
    livre = emprunt["livre"]
```

```
    date = datetime.fromisoformat(emprunt["date_emprunt"])
```

```
    date_str = date.strftime("%d/%m/%Y")
```

```
    print(f" • {livre} (depuis le {date_str})")
```

```
def to_dict(self):
```

```
    """Convertit en dictionnaire."""
```

```
    return {
```

```
        "nom": self.nom,
```

```
        "numero_membre": self.numero_membre,
```

```
        "livres_empruntes": [
```

```
            {
```

```
                "isbn": emprunt["livre"].isbn,
```

```

        "date_emprunt": emprunt["date_emprunt"]
    }
    for emprunt in self.livres_empruntes
]
}

```

```
@classmethod
```

```
def from_dict(cls, data, bibliotheque):
```

```
    """Crée depuis un dictionnaire."""
```

```
    membre = cls(data["nom"], data["numero_membre"])
```

```
    # Reconstituer les emprunts
```

```
    for emprunt_data in data.get("livres_empruntes", []):
```

```
        livre = bibliotheque.rechercher_livre(emprunt_data["isbn"], "isbn")
```

```
        if livre:
```

```
            membre.livres_empruntes.append({
```

```
                "livre": livre,
```

```
                "date_emprunt": emprunt_data["date_emprunt"]
```

```
            })
```

```
    return membre
```

```
class Bibliotheque:
```

```
    """Gère une bibliothèque complète."""
```

```
    def __init__(self, nom):
```

```
        """
```

Initialise la bibliothèque.

Args:

nom (str): Nom de la bibliothèque

"""

self.nom = nom

self.catalogue = []

self.membres = []

def ajouter_livre(self, livre):

"""

Ajoute un livre au catalogue.

Args:

livre (Livre): Livre à ajouter

"""

Vérifier si ISBN existe déjà

if any(l.isbn == livre.isbn for l in self.catalogue):

print(f"⚠ Un livre avec l'ISBN {livre.isbn} existe déjà")

return

self.catalogue.append(livre)

print(f"✓ Livre '{livre.titre}' ajouté au catalogue")

def retirer_livre(self, isbn):

"""

Retire un livre du catalogue.

Args:

isbn (str): ISBN du livre à retirer

"""

for livre in self.catalogue:

if livre.isbn == isbn:

if not livre.disponible:

print("⚠ Impossible de retirer un livre emprunté")

return

self.catalogue.remove(livre)

print(f"✓ Livre retiré du catalogue")

return

print(f"⚠ Aucun livre trouvé avec l'ISBN {isbn}")

def inscrire_membre(self, membre):

"""

Inscrit un nouveau membre.

Args:

membre (Membre): Membre à inscrire

"""

if any(m.numero_membre == membre.numero_membre for m in self.membres):

print(f"⚠ Un membre avec le numéro {membre.numero_membre} existe déjà")

return

self.membres.append(membre)

print(f"✓ {membre.nom} inscrit avec le numéro {membre.numero_membre}")

```
def rechercher_livre(self, critere, type_recherche="titre"):
```

```
    """
```

```
    Recherche un livre.
```

```
    Args:
```

```
        critere (str): Critère de recherche
```

```
        type_recherche (str): Type ('titre', 'auteur', 'isbn')
```

```
    Returns:
```

```
        Livre ou list: Livre(s) trouvé(s)
```

```
    """
```

```
    critere_lower = critere.lower()
```

```
    resultats = []
```

```
    for livre in self.catalogue:
```

```
        if type_recherche == "titre":
```

```
            if critere_lower in livre.titre.lower():
```

```
                resultats.append(livre)
```

```
        elif type_recherche == "auteur":
```

```
            if critere_lower in livre.auteur.lower():
```

```
                resultats.append(livre)
```

```
        elif type_recherche == "isbn":
```

```
            if livre.isbn == critere:
```

```
                return livre # ISBN est unique
```

```
    return resultats if type_recherche != "isbn" else None
```

```
def livres_disponibles(self):
```

```
"""Retourne la liste des livres disponibles."""
```

```
return [livre for livre in self.catalogue if livre.disponible]
```

```
def statistiques(self):
```

```
    """Affiche les statistiques de la bibliothèque."""
```

```
    nb_livres = len(self.catalogue)
```

```
    nb_disponibles = len(self.livres_disponibles())
```

```
    nb_empruntes = nb_livres - nb_disponibles
```

```
    nb_membres = len(self.membres)
```

```
    print(f"\n{'='*60}")
```

```
    print(f"STATISTIQUES - {self.nom}".center(60))
```

```
    print(f"{'='*60}")
```

```
    print(f"Livres au catalogue    : {nb_livres}")
```

```
    print(f" • Disponibles          : {nb_disponibles}")
```

```
    print(f" • Empruntés             : {nb_empruntes}")
```

```
    print(f"Membres inscrits       : {nb_membres}")
```

```
    print(f"{'='*60}")
```

```
def sauvegarder(self, fichier="bibliotheque.json"):
```

```
    """Sauvegarde la bibliothèque."""
```

```
    try:
```

```
        donnees = {
```

```
            "nom": self.nom,
```

```
            "catalogue": [livre.to_dict() for livre in self.catalogue],
```

```
            "membres": [membre.to_dict() for membre in self.membres]
```

```
        }
```

```
with open(fichier, "w", encoding="utf-8") as f:
    json.dump(donnees, f, indent=2, ensure_ascii=False)
```

```
print(f"✓ Bibliothèque sauvegardée dans '{fichier}'")
return True
```

```
except Exception as e:
    print(f"⚠ Erreur lors de la sauvegarde : {e}")
    return False
```

```
@classmethod
```

```
def charger(cls, fichier="bibliotheque.json"):
    """Charge une bibliothèque depuis un fichier."""
    if not os.path.exists(fichier):
        print(f"⚠ Fichier '{fichier}' introuvable")
        return None
```

```
try:
    with open(fichier, "r", encoding="utf-8") as f:
        donnees = json.load(f)

    bibliotheque = cls(donnees["nom"])

    # Charger le catalogue
    for livre_data in donnees.get("catalogue", []):
        if livre_data["type"] == "LivreNumerique":
            livre = LivreNumerique.from_dict(livre_data)
        else:
```

```

        livre = Livre.from_dict(livre_data)

        bibliotheque.catalogue.append(livre)

# Charger les membres
for membre_data in donnees.get("membres", []):
    membre = Membre.from_dict(membre_data, bibliotheque)
    bibliotheque.membres.append(membre)

print(f"✓ Bibliothèque '{bibliotheque.nom}' chargée")
print(f" {len(bibliotheque.catalogue)} livres, {len(bibliotheque.membres)} membres")

return bibliotheque

except Exception as e:
    print(f"⚠ Erreur lors du chargement : {e}")
    return None

def afficher_menu():
    """Affiche le menu principal."""
    print("\n" + "="*60)
    print("SYSTÈME DE BIBLIOTHÈQUE".center(60))
    print("="*60)
    print("1. Ajouter un livre physique")
    print("2. Ajouter un livre numérique")
    print("3. Inscrire un membre")
    print("4. Emprunter un livre")

```

```
print("5. Retourner un livre")
print("6. Rechercher des livres")
print("7. Afficher les livres disponibles")
print("8. Afficher les statistiques")
print("9. Sauvegarder la bibliothèque")
print("10. Charger une bibliothèque")
print("11. Quitter")
print("="*60)
```

```
def main():
```

```
    """Fonction principale."""
```

```
    bibliotheque = Bibliotheque("Bibliothèque Municipale")
```

```
    print("📖 Système de Gestion de Bibliothèque")
```

```
    print(f"Bibliothèque : {bibliotheque.nom}\n")
```

```
    while True:
```

```
        afficher_menu()
```

```
        choix = input("\nVotre choix : ").strip()
```

```
        try:
```

```
            if choix == "1":
```

```
                # Ajouter livre physique
```

```
                titre = input("Titre : ")
```

```
                auteur = input("Auteur : ")
```

```
                isbn = input("ISBN : ")
```

```
                annee = int(input("Année : "))
```

```
livre = Livre(titre, auteur, isbn, annee)
```

```
bibliotheque.ajouter_livre(livre)
```

```
elif choix == "2":
```

```
    # Ajouter livre numérique
```

```
    titre = input("Titre : ")
```

```
    auteur = input("Auteur : ")
```

```
    isbn = input("ISBN : ")
```

```
    annee = int(input("Année : "))
```

```
    taille = float(input("Taille (Mo) : "))
```

```
livre = LivreNumerique(titre, auteur, isbn, annee, taille)
```

```
bibliotheque.ajouter_livre(livre)
```

```
elif choix == "3":
```

```
    # Inscrire membre
```

```
    nom = input("Nom du membre : ")
```

```
    numero = input("Numéro de membre : ")
```

```
membre = Membre(nom, numero)
```

```
bibliotheque.inscrire_membre(membre)
```

```
elif choix == "4":
```

```
    # Emprunter
```

```
    numero_membre = input("Numéro de membre : ")
```

```
    isbn = input("ISBN du livre : ")
```

```
membre = next((m for m in bibliotheque.membres
                if m.numero_membre == numero_membre), None)
if not membre:
    raise MembreNonInscritException(
        f"Membre {numero_membre} non inscrit"
    )
```

```
livre = bibliotheque.rechercher_livre(isbn, "isbn")
if not livre:
    print("⚠ Livre non trouvé")
else:
    membre.emprunter_livre(livre)
```

```
elif choix == "5":
    # Retourner
    numero_membre = input("Numéro de membre : ")
    isbn = input("ISBN du livre : ")
```

```
membre = next((m for m in bibliotheque.membres
                if m.numero_membre == numero_membre), None)
if not membre:
    raise MembreNonInscritException(
        f"Membre {numero_membre} non inscrit"
    )
```

```
livre = bibliotheque.rechercher_livre(isbn, "isbn")
if not livre:
    print("⚠ Livre non trouvé")
```


else:

 membre.retourner_livre(livre)

elif choix == "6":

 # Rechercher

 print("\nType de recherche :")

 print("1. Par titre")

 print("2. Par auteur")

 print("3. Par ISBN")

 type_rech = input("Choix : ")

 if type_rech == "1":

 critere = input("Titre (ou partie) : ")

 resultats = bibliotheque.rechercher_livre(critere, "titre")

 elif type_rech == "2":

 critere = input("Auteur (ou partie) : ")

 resultats = bibliotheque.rechercher_livre(critere, "auteur")

 elif type_rech == "3":

 critere = input("ISBN : ")

 resultat = bibliotheque.rechercher_livre(critere, "isbn")

 resultats = [resultat] if resultat else []

 else:

 print("Choix invalide")

 continue

if resultats:

 print(f"\n✓ {len(resultats)} livre(s) trouvé(s) :")

 for livre in resultats:

```

        livre.afficher_info()

    else:

        print("⚠️ Aucun livre trouvé")


elif choix == "7":

    # Livres disponibles

    disponibles = bibliotheque.livres_disponibles()

    if disponibles:

        print(f"\n✓ {len(disponibles)} livre(s) disponible(s) :")

        for livre in disponibles:

            livre.afficher_info()

    else:

        print("⚠️ Aucun livre disponible")


elif choix == "8":

    # Statistiques

    bibliotheque.statistiques()


elif choix == "9":

    # Sauvegarder

    nom_fichier = input("Nom du fichier (défaut: bibliotheque.json) : ").strip()

    if not nom_fichier:

        nom_fichier = "bibliotheque.json"

    bibliotheque.sauvegarder(nom_fichier)


elif choix == "10":

    # Charger

    nom_fichier = input("Nom du fichier à charger : ").strip()

```

```
bib_chargee = Bibliotheque.charger(nom_fichier)
```

```
if bib_chargee:
```

```
    bibliotheque = bib_chargee
```

```
elif choix == "11":
```

```
    print("\n 📖 Merci et à bientôt !")
```

```
    break
```

```
else:
```

```
    print("⚠️ Choix invalide")
```

```
except LivreNonDisponibleException as e:
```

```
    print(f"\n⚠️ {e}")
```

```
except MembreNonInscritException as e:
```

```
    print(f"\n⚠️ {e}")
```

```
except ValueError as e:
```

```
    print(f"\n⚠️ Valeur invalide : {e}")
```

```
except Exception as e:
```

```
    print(f"\n⚠️ Erreur inattendue : {e}")
```

```
if choix != "11":
```

```
    input("\nAppuyez sur Entrée pour continuer...")
```

```
if __name__ == "__main__":
```

main()

Conclusion

Félicitations ! Vous avez maintenant accès à toutes les solutions des exercices du cours Python.

Ces solutions montrent :

- **Exercice 14** : Manipulation de fichiers JSON, gestion d'erreurs, structures de données
- **Exercice 15** : POO avec exceptions personnalisées, encapsulation, sauvegarde de données
- **Exercice 16** : POO avancée avec héritage, polymorphisme, gestion complète d'un système

Conseils pour utiliser ces solutions

1. **Essayez d'abord par vous-même** : Ne regardez la solution qu'après avoir tenté l'exercice
2. **Comprenez chaque ligne** : Ne copiez pas sans comprendre
3. **Modifiez et expérimentez** : Ajoutez des fonctionnalités, changez le comportement
4. **Testez avec différentes données** : Assurez-vous de comprendre tous les cas
5. **Écrivez votre propre version** : Utilisez ces solutions comme inspiration, pas comme copie

Bon apprentissage ! 