

Encadré Par :

Z. BOUSALEM

Y. MADANI

Présenté Par :

MAFTOUH Omar

Université Sultan Moulay Slimane Faculté Polydisciplinaire Béni Mellal

Département INFORMATIQUE (MIP)

Filière : Science de données et sécurité des systèmes d’information

A.U : 2023-2024

Sujet

**Compte Rendu de TP02 – Python pour la science des données**

**Introduction :**

Ce TP Python, explore différentes notions de programmation en Python. Il présente des exercices variés allant de la manipulation de listes à la création de fonctions pour le traitement des nombres d'Armstrong, la substitution de caractères dans une chaîne, la vérification de l'ordre dans une liste d'entiers, et enfin la réalisation du produit matriciel entre deux matrices. Chaque exercice offre une approche méthodique pour résoudre un problème spécifique, mettant en avant des compétences essentielles en programmation et en calcul matriciel.

**Exercice01 :**

def nombre\_chiffres(x):

    # Initialise un compteur de chiffres à 0

    cp = 0

    # Boucle tant que x est différent de zéro

    while x != 0:

        x //= 10

        cp += 1

    # Retourne le nombre de chiffres dans x

    return cp

def calcul\_nombre\_amstrong(x):

    # Obtient le nombre de chiffres dans x en appelant la fonction nombre\_chiffres

    nbrChiffre = nombre\_chiffres(x)

    # Initialise une liste pour stocker les chiffres de x

    chiffresEntier = []

    # Boucle tant que x est supérieur à zéro

    while x > 0:

        # Obtient le dernier chiffre de x

        chiffre = x % 10

        # Insère le chiffre au début de la liste des chiffres

        chiffresEntier.insert(0, chiffre)

        # Division entière de x par 10 pour enlever le dernier chiffre

        x //= 10

    # Initialise le nombre d'Armstrong à zéro

    nbrAmstrong = 0

    # Boucle sur chaque chiffre dans la liste des chiffres

    for i in range(0, len(chiffresEntier)):

        # Ajoute la puissance de chaque chiffre au nombre de chiffres de x

        nbrAmstrong += chiffresEntier[i] \*\* nbrChiffre

    # Retourne le nombre d'Armstrong calculé

    return int(nbrAmstrong)

def est\_un\_nombre\_amstrong(nombre):

    # Calcule le nombre d'Armstrong pour le nombre donné

    nbrAmstrong = calcul\_nombre\_amstrong(nombre)

    # Vérifie si le nombre est égal à son nombre d'Armstrong

    if nombre == nbrAmstrong:

        return True

    else:

        return False

def afficher\_nombres\_amstrong(List):

    # Initialise une liste pour stocker les nombres d'Armstrong

    ListNbrAmstrong = []

    # Parcourt chaque élément de la liste donnée

    for i in range(0, len(List)):

        # Vérifie si l'élément est un nombre d'Armstrong en appelant la fonction est\_un\_nombre\_amstrong

        if est\_un\_nombre\_amstrong(List[i]):

            # Ajoute l'élément à la liste des nombres d'Armstrong

            ListNbrAmstrong.append(List[i])

    # Retourne la liste des nombres d'Armstrong trouvés dans la liste donnée

    return ListNbrAmstrong

# Liste de nombres à vérifier

L = [153, 1634, 2, 6, 14]

# Affiche la liste des nombres d'Armstrong trouvés dans la liste L

print(f"Liste contenant des nombres d'Armstrong : {afficher\_nombres\_amstrong(L)}")

# Affiche le nombre d'Armstrong pour le nombre 23

print(f"Nombre d'Armstrong pour 23 : {calcul\_nombre\_amstrong(23)}")

# Vérifie si le nombre 14 est un nombre d'Armstrong

print(f"Vérification si 14 est un nombre d'Armstrong : {est\_un\_nombre\_amstrong(14)}")

**Résultat du programme :**

Liste contenant des nombres d'Armstrong : [153, 1634, 2, 6]

Nombre d'Armstrong pour 23 : 13

Vérification si 23 est un nombre d'Armstrong : False

**Exercice 02 :**

L'exercice 2 demande la création d'une fonction en Python, "changer\_car", qui remplace tous les caractères "car1" par "car2" dans une chaîne de caractères "chaine". Les paramètres "debut" et "fin" spécifient une plage optionnelle pour la substitution, mais si absents, la chaîne est traitée entièrement. Notons que l'utilisation de fonctions prédéfinies pour cette opération est interdite.

def changer\_car(chaine, car1, car2, debut=None, fin=None):

    # Si debut n'est pas spécifié, on le définit 0

    if debut is None:

        debut = 0

    # Si fin n'est pas spécifié, on le définit le dernier caractère de la chaîne

    if fin is None:

        fin = len(chaine) - 1

    # Initialise une nouvelle chaîne vide pour stocker le résultat du remplacement

    nouvelle\_chaine = ''

    # Parcours de chaque caractère de la chaîne originale

    for i in range(len(chaine)):

        # Vérifie si l'indice actuel est compris entre debut et fin

        if debut <= i <= fin:

            # Si le caractère à l'indice i est égal à car1, le remplace par car2

            if chaine[i] == car1:

                nouvelle\_chaine += car2

            # Si le caractère n'est pas égal à car1, le conserve tel quel

            else:

                nouvelle\_chaine += chaine[i]

        # Si l'indice n'est pas dans la plage spécifiée, conserve le caractère tel quel

        else:

            nouvelle\_chaine += chaine[i]

    # Retourne la nouvelle chaîne avec les remplacements effectués

    return nouvelle\_chaine

# Exemple d'utilisation

chaine = "Le Lorem Ipsum est simplement"

# Appelle la fonction changer\_car avec les arguments spécifiés

nouvelle\_chaine = changer\_car(chaine, 'o', 'a', 3, 10)

# Affiche la nouvelle chaîne obtenue après le remplacement

print(nouvelle\_chaine)

**Résultat du programme :**

Le Larem Ipsum est simplement

**Exercice 03 :**

L'exercice 3 requiert la création d'une fonction en Python, "verifie\_ordre", qui examine une liste d'entiers et retourne True si elle contient les éléments "1 2 3" dans cet ordre, sinon False. Aucune explication sur la façon dont ils sont arrangés, simplement s'ils apparaissent consécutivement dans cet ordre. Par exemple, [1, 2, 3] ou [4, 1, 2, 3, 5] retourneraient True, tandis que [3, 2, 1] ou [1, 3, 2] retourneraient False.

def verifie\_ordre(liste):

    # Initialise une variable pour stocker si l'ordre est vérifié, initialement à False

    order = False

    # Initialise les clés pour les nombres 1, 2 et 3 à 0

    cle01, cle02, cle03 = 0, 0, 0

    # Parcourt chaque élément dans la liste

    for i in range(0, len(liste)):

        # Si l'élément est égal à 1, met à jour la clé pour 1

        if liste[i] == 1:

            cle01 = i

        # Si l'élément est égal à 2, met à jour la clé pour 2

        elif liste[i] == 2:

            cle02 = i

        # Si l'élément est égal à 3, met à jour la clé pour 3

        elif liste[i] == 3:

            cle03 = i

    # Vérifie si l'ordre est vérifié

    if (cle01 < cle02 and cle01 < cle03) or (cle02 > cle01 and cle02 < cle03):

        # Si les conditions sont satisfaites, met à jour la variable order à True

        order = True

    # Retourne l'indicateur d'ordre

    return order

# Définit trois listes pour tester la fonction

L00 = [-1, 6, 7, 1, 2, 3, 5]

L01 = [9, 1, 7, 4, 2, 5, 3]

L02 = [6, 3, 12, 2, -4, 5, 1]

# Appelle la fonction pour tester avec la liste L01 et affiche le résultat

print(verifie\_ordre(L01))  # Output: True

**Exercice 04 :**

L'exercice 4 implique la création d'une fonction en Python, "calcul\_produit\_matriciel(A, B)", qui effectue le produit matriciel entre les matrices A et B. Ce produit matriciel est défini lorsque le nombre de colonnes de la matrice A est égal au nombre de lignes de la matrice B. Pour cela, plusieurs fonctions auxiliaires doivent être mises en œuvre :

* "verifie\_dimensions()": Cette fonction demande à l'utilisateur d'entrer les dimensions des matrices A et B et vérifie si le nombre de colonnes de A est égal au nombre de lignes de B. En cas de non-conformité, elle redemande ces deux valeurs.
* "initialiser\_matrice(R,C)": Fonction pour initialiser et retourner une liste bidimensionnelle représentant une matrice de dimensions R,C.
* "remplir\_matrice(R, C)": Cette fonction remplit une liste bidimensionnelle (matrice) et renvoie une matrice de dimensions R,C. Elle utilise la fonction initialiser\_matrice(R, C).
* "calcul\_produit\_matriciel(A,B,C,m,n,p)": Cette fonction calcule le produit matriciel de A et B, en initialisant une liste C qui servira de matrice résultante. Les dimensions des matrices sont représentées par m, n et p.

def verifie\_dimensions():

    # Fonction pour vérifier si les dimensions des matrices sont compatibles pour le produit matriciel.

    while True:

        m = int(input("Entrez le nombre de lignes de la matrice A : "))

        n = int(input("Entrez le nombre de colonnes de la matrice A et le nombre de lignes de la matrice B : "))

        p = int(input("Entrez le nombre de colonnes de la matrice B : "))

        # Vérifie si le nombre de colonnes de A est égal au nombre de lignes de B

        if n == p:

            # Si les dimensions sont compatibles, retourne les valeurs m, n, p

            return m, n, p

        else:

            # Si les dimensions ne sont pas compatibles, demande à l'utilisateur de réessayer

            print("Les dimensions des matrices ne sont pas compatibles. Veuillez réessayer.")

def initialiser\_matrice(R, C):

    # Initialise une liste vide pour stocker les lignes de la matrice

    matrice = []

    # Boucle sur le nombre de lignes R

    for \_ in range(R):

        # Initialise une liste vide pour stocker les éléments de chaque ligne

        ligne = []

        # Boucle sur le nombre de colonnes C

        for \_ in range(C):

            # Ajoute un zéro à la ligne pour chaque colonne

            ligne.append(0)

        # Ajoute la ligne complète à la matrice

        matrice.append(ligne)

    return matrice

def remplir\_matrice(R, C):

    # Remplit une liste bidimensionnelle pour représenter une matrice avec les entrées de l'utilisateur.

    matrice = initialiser\_matrice(R, C)

    print("Entrez les éléments de la matrice :")

    for i in range(R):

        for j in range(C):

            matrice[i][j] = float(input(f"Entrez l'élément [{i + 1},{j + 1}] : "))

    return matrice

def calcul\_produit\_matriciel(A, B, m, n, p):

    # Calcule le produit matriciel de A et B.

    # Initialise la matrice résultante C avec des zéros

    C = initialiser\_matrice(m, p)

    # Effectue le produit matriciel

    for i in range(m):

        for j in range(p):

            for k in range(n):

                C[i][j] += A[i][k] \* B[k][j]

    return C

# Main code

# Demande à l'utilisateur de saisir les dimensions des matrices et les vérifie

m, n, p = verifie\_dimensions()

# Remplit les matrices A et B avec les entrées de l'utilisateur

print("Pour la matrice A :")

A = remplir\_matrice(m, n)

print("Pour la matrice B :")

B = remplir\_matrice(n, p)

# Calcule le produit matriciel de A et B

produit\_matriciel = calcul\_produit\_matriciel(A, B, m, n, p)

# Affiche la matrice résultante

print("Le produit matriciel de A et B est :")

for row in produit\_matriciel:

    print(row)

**Résultat du programme :**

Entrez le nombre de lignes de la matrice A : 2

Entrez le nombre de colonnes de la matrice A et le nombre de lignes de la matrice B : 1

Entrez le nombre de colonnes de la matrice B : 1

Pour la matrice A :

Entrez les éléments de la matrice :

Entrez l'élément [1,1] : 1

Entrez l'élément [2,1] : 2

Pour la matrice B :

Entrez les éléments de la matrice :

Entrez l'élément [1,1] : 1

Le produit matriciel de A et B est :

[1.0]

[2.0]

**Conclusion :**

En conclusion, ce TP Python pour la Science des données a permis d'explorer divers concepts et techniques de programmation en Python, allant de la manipulation de listes à la création de fonctions pour le traitement des nombres d'Armstrong, la substitution de caractères dans une chaîne, la vérification de l'ordre dans une liste d'entiers, et enfin la réalisation du produit matriciel entre deux matrices. Chaque exercice a offert une opportunité d'apprentissage pratique, mettant en avant des compétences essentielles en programmation et en calcul matriciel. Ces exercices ont permis de renforcer la compréhension des structures de données et des algorithmes fondamentaux en Python, tout en illustrant l'importance de la méthodologie et de la logique dans la résolution de problèmes informatiques.