Session 3 : Manipulation Avancée des Structures de Données

Objectifs

- Comprendre et implémenter des listes chaînées et des piles.
- Utiliser des structures avancées comme `Counter` et `defaultdict`.
- Manipuler efficacement des ensembles (`set`) et leurs opérations.
- Appliquer ces concepts à travers des exercices pratiques.

---

1. Listes chaînées et Piles

Listes chaînées

Une liste chaînée est une structure de données linéaire composée de nœuds. Chaque nœud contient une valeur et une référence au nœud suivant.

Implémentation en Python

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        temp = self.head
        while temp.next:
            temp = temp.next
        temp.next = new_node

    def display(self):
        temp = self.head
        while temp:
            print(temp.data, end=' -> ')
            temp = temp.next
        print('None')

# Exemple d'utilisation
```

```
ll = LinkedList()
ll.append(1)
ll.append(2)
ll.append(3)
ll.display()
```

Piles (Stacks)

Une pile est une structure LIFO (Last-In, First-Out). L'opération principale est `push` (ajout) et `pop` (retrait).

Implémentation en Python

```
class Stack:
    def __init__(self):
        self.stack = []

    def push(self, item):
        self.stack.append(item)

    def pop(self):
        if not self.is_empty():
            return self.stack.pop()
        return None

    def peek(self):
        return self.stack[-1] if self.stack else None

    def is_empty(self):
        return len(self.stack) == 0

# Exemple d'utilisation
s = Stack()
s.push(5)
s.push(10)
print(s.pop())  # 10
```

---

2. Dictionnaires avancés et collections

Counter

Le module `collections.Counter` permet de compter efficacement les occurrences d'éléments.

```
from collections import Counter

text = "abracadabra"
c = Counter(text)
print(c)  # {'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1}
```

defaultdict

`defaultdict` est une version améliorée de `dict` qui permet d'éviter les erreurs de clé inexistante.

```
from collections import defaultdict

dd = defaultdict(int)
dd["a"] += 1
print(dd["a"])  # 1
print(dd["b"])  # 0 (au lieu d'une erreur)
```

---

3. Sets et opérations sur les ensembles

Les ensembles (`set`) sont des collections non ordonnées d'éléments uniques.

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
print(A | B)  # Union {1, 2, 3, 4, 5, 6}
print(A & B)  # Intersection {3, 4}
print(A - B)  # Différence {1, 2}
print(A ^ B)  # Différence symétrique {1, 2, 5, 6}
```

---

4. Exercices de mise en pratique

Exercice 1 : Implémentation d'une liste chaînée

Implémentez une méthode `remove(value)` pour supprimer un élément d'une liste chaînée.

Exercice 2 : Analyse de texte avec `Counter`

Écrivez un programme qui prend un texte en entrée et affiche les 3 mots les plus fréquents.

Exercice 3 : Gestion d'une pile

Créez une classe `UndoStack` qui permet d'annuler les dernières actions effectuées.