

## Partie 1 : Les Files (FIFO)

### 1.1 Exercice 1 : Découverte des files ★

Une file est une structure de données FIFO (First In, First Out). On dispose d'une file implémentée avec une liste Python et des fonctions `enfiler(f, element)` et `defiler(f)`.

#### Questions :

1. Quelle est la différence entre une file et une pile ?
2. On effectue les opérations suivantes sur une file vide `f` :

```
1 enfiler(f, 5)
2 enfiler(f, 8)
3 enfiler(f, 3)
4 x = defiler(f)
5 enfiler(f, 12)
```

Quel est le contenu de la file après ces opérations ? Quelle est la valeur de `x` ?

---

### 1.2 Exercice 2 : Utilisation d'une file ★★

On simule une file d'attente à une caisse. Voici l'implémentation fournie :

```
1 def creer_file():
2     return []
3
4 def est_vide(f):
5     return len(f) == 0
6
7 def enfiler(f, element):
8     f.append(element)
9
10 def defiler(f):
11     if est_vide(f):
12         raise IndexError("File vide")
13     return f.pop(0)
```

**Question :** Écrire une fonction `taille_file(f)` qui renvoie le nombre d'éléments dans la file.

---

### 1.3 Exercice 3 : Application - Gestion d'imprimante ★★

Une imprimante reçoit des documents à imprimer. Chaque document est représenté par son nom.

**Question :** Écrire une fonction `traiter_impression(documents)` qui :

- Prend en paramètre une liste de noms de documents

- Utilise une file pour gérer l'ordre d'impression
- Affiche "Impression de : [nom]" pour chaque document dans l'ordre FIFO
- Renvoie le nombre de documents traités

**Exemple :**

```
1 traiter_impression(["rapport.pdf", "photo.jpg", "facture.pdf"])
2 # Affiche :
3 # Impression de : rapport.pdf
4 # Impression de : photo.jpg
5 # Impression de : facture.pdf
6 # Renvoie : 3
```

---

Partie 2

## Partie 2 : Les Listes Chaînées

### 2.1 Exercice 4 : Compréhension ★

Une liste chaînée est composée de maillons. Chaque maillon contient une valeur et une référence vers le maillon suivant.

**Questions :**

1. Quel est l'avantage d'une liste chaînée par rapport à un tableau ?
2. Dessiner une représentation schématique d'une liste chaînée contenant les valeurs 7, 3, 9, 1.

### 2.2 Exercice 5 : Liste chaînée avec dictionnaire simple ★★

On implémente une liste chaînée à l'aide d'un dictionnaire où chaque clé correspond à une valeur et pointe vers la valeur suivante.

**Exemple de liste chaînée contenant 10, 20, 30 :**

```
1 liste = {10: 20, 20: 30, 30: None}
```

Ici, 10 est la première valeur (la tête de liste), 20 suit 10, 30 suit 20, et None indique la fin.

**Questions :**

- a) Écrire une fonction `creer_liste(valeur)` qui crée et renvoie un dictionnaire représentant une liste chaînée contenant uniquement valeur.
- b) Écrire une fonction `ajouter_fin(liste, tete, valeur)` qui ajoute une nouvelle valeur à la fin de la liste chaînée. La fonction prend en paramètres : - `liste` : le dictionnaire représentant la liste chaînée - `tete` : la valeur du premier élément de la liste - `valeur` : la nouvelle valeur à ajouter

La fonction doit parcourir la liste jusqu'au dernier élément (celui qui pointe vers None), puis ajouter la nouvelle valeur.

**Exemple d'utilisation :**

```

1 liste = creer_liste(10)
2 ajouter_fin(liste, 10, 20)
3 ajouter_fin(liste, 10, 30)
4 # liste contient maintenant : {10: 20, 20: 30, 30: None}

```

c) Écrire une fonction `afficher_liste(liste, tete)` qui affiche toutes les valeurs de la liste dans l'ordre, en partant de `tete`.

d) **Question de réflexion** : Quelles sont les limites de cette implémentation ? Que se passe-t-il si on veut stocker deux fois la même valeur dans la liste ?

---

### 2.3 Exercice 6 : Liste chaînée avec dictionnaires imbriqués ★★

On implémente une liste chaînée à l'aide de dictionnaires. Chaque maillon est un dictionnaire avec deux clés :  
 - 'valeur' : la donnée stockée - 'suivant' : le maillon suivant (ou None si c'est le dernier)

Exemple de liste chaînée contenant 5, 8, 3 :

```

1 liste = {
2     'valeur': 5,
3     'suivant': {
4         'valeur': 8,
5         'suivant': {
6             'valeur': 3,
7             'suivant': None
8         }
9     }
10 }

```

**Questions :**

a) Écrire une fonction `creer_maillon(valeur)` qui crée et renvoie un maillon contenant `valeur` et dont le suivant est None.

b) Écrire une fonction `ajouter_fin(liste, valeur)` qui ajoute un nouveau maillon contenant `valeur` à la fin de la liste chaînée. La fonction doit parcourir la liste jusqu'au dernier maillon, puis modifier son champ 'suivant'.

Exemple d'utilisation :

```

1 liste = creer_maillon(10)
2 ajouter_fin(liste, 20)
3 ajouter_fin(liste, 30)
4 # liste contient maintenant : 10 -> 20 -> 30

```

c) Écrire une fonction `afficher_liste(liste)` qui affiche toutes les valeurs de la liste dans l'ordre.

d) Comparer cette implémentation avec celle de l'exercice 5. Quels sont les avantages et inconvénients de chaque approche ?

---

### 2.4 Exercice 7 : Liste chaînée en POO ★★★

On souhaite implémenter une liste chaînée en programmation orientée objet.

**Implémentation fournie :**

```
1 class Maillon:
2     def __init__(self, valeur):
3         self.valeur = valeur
4         self.suivant = None
5
6 class ListeChaine:
7     def __init__(self):
8         self.tete = None
9
10    def est_vide(self):
11        return self.tete is None
12
13    def ajouter_debut(self, valeur):
14        nouveau = Maillon(valeur)
15        nouveau.suivant = self.tete
16        self.tete = nouveau
```

**Questions :**

- a) Compléter la classe ListeChaine avec une méthode `afficher()` qui affiche tous les éléments de la liste.
  - b) Ajouter une méthode `longueur()` qui renvoie le nombre d'éléments dans la liste.
  - c) Ajouter une méthode `rechercher(valeur)` qui renvoie `True` si la valeur est présente dans la liste, `False` sinon.
  - d) Ajouter une méthode `supprimer_debut()` qui supprime le premier élément de la liste et renvoie sa valeur. Si la liste est vide, lever une exception.
- 

**2.5 Exercice 8 : Inversion d'une liste chaînée ★★★**

En utilisant la classe ListeChaine de l'exercice 7, écrire une méthode `inverser()` qui inverse l'ordre des éléments de la liste chaînée.

**Exemple :**

```
1 liste = ListeChaine()
2 liste.ajouter_debut(3)
3 liste.ajouter_debut(8)
4 liste.ajouter_debut(5)
5 # Liste : 5 -> 8 -> 3
6
7 liste.inverser()
8 # Liste : 3 -> 8 -> 5
```

**Contrainte :** Utiliser uniquement les opérations sur les maillons, sans créer de nouvelle liste.

---

**2.6 Exercice 9 : Fusion de deux listes triées ★★★★★**

On dispose de deux listes chaînées triées par ordre croissant. Écrire une fonction `fusionner(liste1, liste2)` qui crée et renvoie une nouvelle liste chaînée contenant tous les éléments des deux listes, toujours triés par ordre croissant.