Piles et files

Structures de données

Exercice 1 *QCM : piles et files*

Pour chaque question, une seule réponse est correcte.

1. On empile successivement A, B, C, D dans une pile vide. On dépile deux fois. Quelle lettre est au sommet ?

   A. A

   B. B

   C. C

   D. D

2. On enfile A, B, C dans une file, puis on défile une fois. Quelle lettre est renvoyée ?

   A. C (la dernière enfilée)

   B. B (celle du milieu)

   C. A (la première enfilée)

   D. une erreur car la file est pleine

3. Parmi les situations suivantes, laquelle correspond à une pile ?

   A. les clients qui font la queue à la boulangerie

   B. la fonction « annuler » (Ctrl-Z) d'un éditeur de texte

   C. les documents envoyés à une imprimante partagée

   D. les voitures qui passent à un péage dans l'ordre d'arrivée

4. Quelle opération est **interdite** sur une pile ?

   A. consulter l'élément au sommet sans le retirer

   B. empiler un nouvel élément

   C. accéder directement au troisième élément en partant du fond

   D. dépiler l'élément au sommet

5. On implémente une pile avec `list` en utilisant `append` et `pop()`. Complexité ?

   A. `append` en $O(n)$, `pop()` en $O(n)$

   B. `append` en $O(1)$, `pop()` en $O(1)$

   C. `append` en $O(1)$, `pop()` en $O(n)$

   D. `append` en $O(n)$, `pop()` en $O(1)$

Exercice 2 *Exercice guidé : dérouler empiler/dépiler pas à pas*

On considère une pile `p` initialement vide. On exécute :

```
p.empiler(4)
p.empiler(7)
p.empiler(2)
a = p.depiler()
p.empiler(9)
b = p.depiler()
p.empiler(1)
c = p.depiler()
```

1. Compléter le tableau en dessinant l'état de la pile après chaque instruction et en indiquant les valeurs de `a`, `b` et `c`.

Instruction	État (bas → haut)	Variable
`empiler(4)`		
`empiler(7)`		
`empiler(2)`		
`depiler()`		`a =`
`empiler(9)`		
`depiler()`		`b =`
`empiler(1)`		
`depiler()`		`c =`

2. Quels éléments restent dans la pile ? Dans quel ordre ?

Exercice 3 *Compléter une classe Pile à trous*

Compléter les méthodes :

```
class Pile:
    def __init__(self):
        self.contenu = ...        # creer une liste vide

    def est_vide(self):
        return ...                # vrai si la liste est vide

    def empiler(self, element):
        ...                       # ajouter en fin de liste

    def depiler(self):
        if self.est_vide():
            raise ValueError("pile vide")
```

```
        return ...                # retirer et renvoyer le dernier

    def sommet(self):
        if self.est_vide():
            raise ValueError("pile vide")
        return ...                # acceder au dernier element

    def __len__(self):
        return ...                # nombre d'elements
```

Exercice 4 *Implémenter une file*

Écrire une classe `File` avec `list` et les méthodes : `est_vide`, `enfiler`, `defiler`, `__len__`.

Indication : pour retirer le premier élément, utiliser `pop(0)`.

Tester avec :

```
f = File()
f.enfiler("Alice")
f.enfiler("Bob")
f.enfiler("Charlie")
print(f.defiler())   # Alice
print(f.defiler())   # Bob
```

Exercice 5 *Notation polonaise inversée (NPI)*

La **notation polonaise inversée** écrit les calculs sans parenthèses : d'abord les opérandes, puis l'opérateur. L'évaluation utilise une pile : si le jeton est un nombre, on l'empile ; si c'est un opérateur, on dépile deux opérandes, on calcule, on empile le résultat.

1. Évaluer pas à pas `5 3 + 8 2 - *` en complétant le tableau :

Jeton	Action	Pile (bas → haut)
5	empiler 5	5
3		
+		
8		
2		
-		
*		

2. Écrire une fonction `evaluer_npi(expression)` utilisant la classe `Pile`.
3. Tester avec `evaluer_npi("5 3 + 8 2 - *")` → 48.

Exercice 6 *Vérification du parenthésage*

1. Dérouler l'algorithme de vérification sur `"[a + (b * c)]"` en montrant l'état de la pile.
2. Écrire `parenthesage_valide(expression)` qui renvoie `True` si le parenthésage est correct.
3. Tester sur : `"()"`, `"([a + b] * (c - d))"`, `"(a + b]"`, `"(("`.

Exercice 7 *Inversion d'une file avec une pile*

1. Expliquer le principe pour inverser une file en utilisant uniquement une pile.
2. Écrire `inverser_file(f)`.
3. Tester : file `A` ← `B` ← `C` (A en tête) → `C` ← `B` ← `A`.

Exercice 8 *Synthèse : simuler une file d'impression*

1. Créer une classe `FileImpression` avec : `soumettre(nom, pages)`, `imprimer_suivant()`, `documents_en_attente()`, `pages_restantes()`.
2. Simuler : Alice soumet « Rapport » (12 pages), Bob soumet « CV » (2 pages), un document est imprimé, Charlie soumet « Thèse » (150 pages), les deux restants sont imprimés.
3. Afficher le total de pages imprimées.