Feuille d'exercices de bac sur les piles et files

Exercice 1 : Une méthode simple pour gérer l'ordonnancement des processus est d'exécuter les processus en une seule fois et dans leur ordre d'arrivée.

1) Parmi les propositions suivantes, quelle est la structure de données la plus appropriée pour mettre en œuvre le mode FIFO (First In First Out)?
   a) liste b) dictionnaire c) pile d) file
2) On choisit de stocker les données des processus en attente à l'aide d'une liste Python `lst`. On dispose déjà d'une fonction `retirer(lst)` qui renvoie l'élément `lst[0]` puis le supprime de la liste `lst`. Écrire en Python le code d'une fonction `ajouter(lst, proc)` qui ajoute à la fin de la liste `lst` le nouveau processus en attente `proc`.

On choisit maintenant d'implémenter une file **`file`** à l'aide d'un couple `(p1, p2)` où `p1` et `p2` sont des piles. Ainsi **`file`**`[0]` et **`file`**`[1]` sont respectivement les piles `p1` et `p2`. Pour enfiler un nouvel élément `elt` dans **`file`**, on l'empile dans `p1`. Pour défiler **`file`**, deux cas se présentent.

- La pile `p2` n'est pas vide : on dépile `p2`.
- La pile `p2` est vide : on dépile les éléments de `p1` en les empilant dans `p2` jusqu'à ce que `p1` soit vide, puis on dépile `p2`.

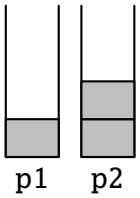
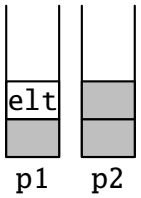
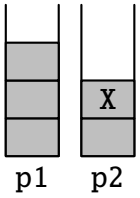
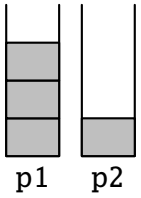
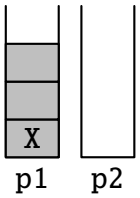
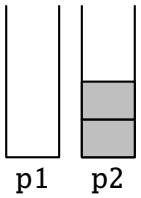
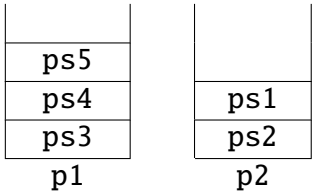
	État de la file avant	État de la file après
`enfiler(file, elt)`		
`defiler(file)` cas où `p2` n'est pas vide		
`defiler(file)` cas où `p2` est vide		

Illustration du fonctionnement des fonctions `enfiler` et `défiler`.

3) On exécute la séquence d'instructions suivante à partir de la situation ci-dessous:

```
enfiler(file, ps6)
defiler(file)
defiler(file)
defiler(file)
enfiler(file, ps7)
```



Représenter le contenu final des deux piles à la suite de ces instructions.

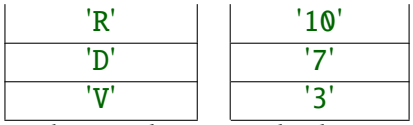
4) On dispose des fonctions :
   - `empiler(p, elt)` qui empile l'élément `elt` dans la pile `p`,
   - `depiler(p)` qui renvoie le sommet de la pile `p` si `p` n'est pas vide et le supprime,
   - `pile_vide(p)` qui renvoie **`True`** si la pile `p` est vide, **`False`** si la pile `p` n'est pas vide.

   a) Écrire en Python une fonction `est_vide(f)` qui prend en argument un couple de piles `f` et qui renvoie **`True`** si la file représentée par `f` est vide, **`False`** sinon.
   b) Écrire en Python une fonction `enfiler(f, elt)` qui prend en arguments un couple de piles `f` et un élément `elt` et qui ajoute `elt` en queue de la file représentée par `f`.
   c) Écrire en Python une fonction `defiler(f)` qui prend en argument un couple de piles `f` et qui renvoie l'élément en tête de la file représentée par `f` en le retirant.

Exercice 2 : On cherche à obtenir un mélange d'une liste comportant un nombre **pair** d'éléments. Dans cet exercice, on notera N le nombre d'éléments de la liste à mélanger.
La méthode de mélange utilisée dans cette partie est inspirée d'un mélange de jeux de cartes :

- On sépare la liste en deux piles :
  - à gauche, la première pile contient les N/2 premiers éléments de la liste ;
  - à droite, la deuxième pile contient les N/2 derniers éléments de la liste.
- On crée une liste vide.
- On prend alors le sommet de la pile de gauche et on le met en début de liste.
- On prend ensuite le sommet de la pile de droite que l'on ajoute à la liste et ainsi de suite jusqu'à ce que les piles soient vides.

Par exemple, si on applique cette méthode de mélange à la liste `['V','D','R','3','7','10']`, on obtient pour le partage de la liste en 2 piles :



Pile gauche Pile droite

La nouvelle liste à la fin du mélange sera donc `['R','10','D','7','V','3']`.

1) Que devient la liste `['7','8','9','10','V','D','R','A']` si on lui applique cette méthode de mélange ?

On considère que l'on dispose de la structure de données de type pile, munie des seules instructions suivantes :

- `p = Pile()` : crée une pile vide nommée `p`
- `p.est_vide()` : renvoie **`True`** si la liste est vide, **`False`** sinon
- `p.empiler(e)` : ajoute l'élément `e` dans la pile
- `e = p.depiler()` : retire le dernier élément ajouté dans la pile et le renvoie (et l'affecte à la variable `e`)
- `p2 = p.copier()` : renvoie une copie de la pile `p` sans modifier la pile `p` et l'affecte à une nouvelle pile `p2`

2) Compléter le code de la fonction suivante qui transforme une liste en pile.

```
def liste_vers_pile(L):
    '''prend en paramètre une liste et renvoie une pile'''
    N = len(L)
    p_temp = Pile()
    for i in range(N):
        . . . . . .
    return . . . . . .
```

3) On considère la fonction suivante qui partage une liste en deux piles. Lors de sa mise au point et pour aider au débuggage, des appels à la fonction `affichage_pile` ont été insérés. La fonction `affichage_pile(p)` affiche la pile `p` à l'écran verticalement sous la forme suivante :

dernier élément empilé
...
...
premier élément empilé

```
def partage(L):
    N = len(L)
    p_gauche = Pile()
    p_droite = Pile()
    for i in range(N//2):
        p_gauche.empile(L[i])
    for i in range(N//2, N):
        p_droite.empile(L[i])
    affichage_pile(p_gauche)
    affichage_pile(p_droite)
    return p_gauche, p_droite
```

Quels affichages obtient-on à l'écran lors de l'exécution de l'instruction : `partage([1, 2, 3, 4, 5, 6])` ?

4) a) Dans un cas général et en vous appuyant sur une séquence de schémas, expliquer en quelques lignes comment fusionner deux piles `p_gauche` et `p_droite` pour former une liste `L` en alternant un à un les éléments de la pile `p_gauche` et de la pile `p_droite`.

b) Écrire une fonction `fusion(p1, p2)` qui renvoie une liste construite à partir des deux piles `p1` et `p2`.

5) Compléter la dernière ligne du code de la fonction `affichage_pile` pour qu'elle fonctionne de manière récursive.

```
def affichage_pile(p):
    p_temp = p.copier()
    if p_temp.est_vide():
        print('____')
    else:
        elt = p_temp.depiler()
        print('| ', elt, ' |')
        . . . . .                    # ligne à compléter
```

Exercice 3 : *Cet exercice traite du thème « structures de données », et principalement des piles.*

La classe `Pile` utilisée dans cet exercice est implémentée en utilisant des listes Python et propose quatre éléments d'interface :

- Un constructeur qui permet de créer une pile vide, représentée par `[]` ;
- La méthode `est_vide()` qui renvoie `True` si l'objet est une pile ne contenant aucun élément, et `False` sinon ;
- La méthode `empiler` qui prend un objet quelconque en paramètre et ajoute cet objet au sommet de la pile. Dans la représentation de la pile dans la console, cet objet apparait à droite des autres éléments de la pile ;
- La méthode `depiler` qui renvoie l'objet présent au sommet de la pile et le retire de la pile.

Exemples :

```
>>> ma_pile = Pile()
>>> ma_pile.empiler(2)
>>> ma_pile
[2]
>>> ma_pile.empiler(3)
>>> ma_pile.empiler(50)
```

```
>>> ma_pile
[2, 3, 50]
>>> ma_pile.depiler()
50
>>> ma_pile
[2, 3]
```

```
1  def est_triee(self):
2      if not self.est_vide():
3          e1 = self.depiler()
4          while not self.est_vide():
5              e2 = self.depiler()
6              if e1 ... e2:
7                  return False
8              e1 = ...
9      return True
```

La méthode `est_triee` ci-dessous renvoie **`True`** si, en dépilant tous les éléments, ils sont traités dans l'ordre croissant, et **`False`** sinon.

1) Recopier sur la copie les lignes 6 et 8 en complétant les points de suspension.

On crée dans la console la pile `A` représentée par `[1, 2, 3, 4]`.

2) a) Donner la valeur renvoyée par l'appel `A.est_triee()`.
   b) Donner le contenu de la pile `A` après l'exécution de cette instruction.

```
1  def depile_max(self):
2      assert not self.est_vide(), "Pile vide"
3      q = Pile()
4      maxi = self.depiler()
5      while not self.est_vide():
6          elt = self.depiler()
7          if maxi < elt:
8              q.empiler(maxi)
9              maxi = ...
10         else :
11             ...
12     while not q.est_vide():
13         self.empiler(q.depiler())
14     return maxi
```

On souhaite maintenant écrire le code d'une méthode `depile_max` d'une pile non vide ne contenant que des nombres entiers et renvoyant le plus grand élément de cette pile en le retirant de la pile.
Après l'exécution de `p.depile_max()`, le nombre d'éléments de la pile `p` diminue donc de 1.

3) Recopier sur la copie les lignes 9 et 11 en complétant les points de suspension.

On crée la pile `B` représentée par `[9, -7, 8, 12, 4]` et on effectue l'appel `B.depile_max()`.

4) a) Donner le contenu des piles `B` et `q` à la fin de chaque itération de la boucle **`while`** de la ligne 5.
   b) Donner le contenu des piles `B` et `q` avant l'exécution de la ligne 14.
   c) Donner un exemple de pile qui montre que l'ordre des éléments restants n'est pas préservé après l'exécution de `depile_max`.

On donne le code de la méthode `traiter` :

```
1  def traiter(self):
2      q = Pile()
3      while not self.est_vide():
4          q.empiler(self.depile_max())
5      while not q.est_vide():
6          self.empiler(q.depiler())
```

5) a) Donner les contenus successifs des piles `B` et `q`
   - avant la ligne 3,
   - avant la ligne 5,
   - à la fin de l'exécution de la fonction `traiter` lorsque la fonction `traiter` est appelée avec la pile `B` contenant `[1, 6, 4, 3, 7, 2]`.

   b) Expliquer le traitement effectué par cette méthode.

Exercice 4 : Dans cet exercice, on considère une pile d'entiers positifs. On suppose que les quatre fonctions suivantes ont été programmées préalablement en langage Python :

- `empiler(P, e)` : ajoute l'élément `e` sur la pile `P` ;
- `depiler(P)` : enlève le sommet de la pile `P` et renvoie la valeur de ce sommet ;
- `est_vide(P)` : renvoie `True` si la pile est vide et `False` sinon ;
- `creer_pile()` : renvoie une pile vide.

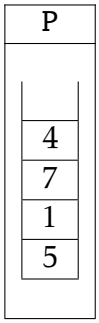
Dans cet exercice, seule l'utilisation de ces quatre fonctions sur la structure de données pile est autorisée.

1) Compléter le schéma ci-dessous en exécutant les appels de fonctions donnés. On écrira ce que renvoie la fonction utilisée dans chaque cas, et on indiquera `None` si la fonction ne renvoie aucune valeur.

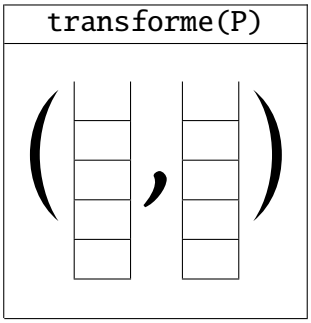
	Étape 0 Pile d'origine `P`	Étape 1 `empiler(P, 9)`	Étape 2 `depiler(P)`	Étape 3 `est_vide(P)`
	4 7 1 5			
Valeur renvoyée				

2) On propose la fonction ci-dessous, qui prend en argument une pile `P` et renvoie un couple de piles :

```
def transforme(P) :
    Q = creer_pile()
    while not est_vide(P) :
        v = depiler(P)
        empiler(Q, v)
    return (P, Q)
```



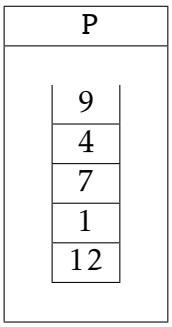
L'exécution de `transforme(P)` renvoie



Compléter sur votre copie le schéma ci-dessus.

3) Écrire une fonction `maximum` en langage Python recevant une pile `P` non vide comme argument et qui renvoie la valeur maximale de cette pile. On ne s'interdit pas qu'après exécution de la fonction, la pile soit vide.

4) On souhaite connaître le nombre d'éléments d'une pile à l'aide de la fonction `taille(P)`. Après l'appel de cette fonction, la pile est dans son état initial.
   a) Proposer une stratégie écrite en langage naturel et/ou expliquée à l'aide de schémas, qui permette de mettre en place une telle fonction.
   b) Donner le code Python de cette fonction `taille(P)` (on pourra utiliser les cinq fonctions déjà programmées).



`taille(P)` renverra l'entier 5

Exercice 5 : Les interfaces de structures de données abstraites `Pile` et `File` sont proposées ci-dessous. On utilisera uniquement les fonctions ci-dessous :

Structure de données abstraite : `Pile`
Utilise : `Elément`, `Booléen`

Opérations :
• `creer_pile_vide()` : $\emptyset \longrightarrow$ `Pile`
`creer_pile_vide()` renvoie une pile vide
• `est_vide()` : `Pile` $\longrightarrow$ `Booléen`
`est_vide(pile)` renvoie **`True`** si `pile` est vide, **`False`** sinon
• `empiler(pile, element)` : `Pile`,`Elément` $\longrightarrow \emptyset$
`empiler(pile, element)` ajoute `element` à la pile `pile`
• `depiler(pile)` : `Pile` $\longrightarrow$ `Elément`
`depiler(pile)` renvoie l'élément au sommet de la `pile` en le retirant de la `pile`

Structure de données abstraite : `File`
Utilise : `Elément`, `Booléen`
Opérations : • `creer_file_vide()` : $\emptyset \longrightarrow$ `File` `creer_file_vide()` renvoie une file vide • `est_vide()` : `File` $\longrightarrow$ `Booléen` `est_vide(file)` renvoie **`True`** si **`file`** est vide, **`False`** sinon • `enfiler(file, element)` : `File`,`Elément` $\longrightarrow \emptyset$ `enfiler(file, element)` ajoute `element` dans la file **`file`** • `defiler(file)` : `File` $\longrightarrow$ `Elément` `defiler(file)` renvoie l'élément en tête de la **`file`** en le retirant de la **`file`**

1) a) On considère la file `F` suivante :

enfilement $\longrightarrow$ `"rouge" "vert" "jaune" "rouge" "jaune"` $\longrightarrow$ défilement

Quel sera le contenu de la pile `P` et de la file `F` après l'exécution du programme Python suivant ?

```
P = creer_pile_vide()
while not(est_vide(F)):
    empiler(P, defiler(F))
```

b) Créer une fonction `taille_file` qui prend en paramètre une file `F` et qui renvoie le nombre d'éléments qu'elle contient. Après appel à cette fonction, la file `F` doit avoir retrouvé son état d'origine.

2) Écrire une fonction `former_pile` qui prend en paramètre une file `F` et qui renvoie une pile `P` contenant les mêmes éléments que la file.

Le premier élément sorti de la file devra se trouver au sommet de la pile, le deuxième élément sorti de la file devra se trouver juste en-dessous du sommet, etc.
Exemple : Si `F =` `"rouge" "vert" "jaune" "rouge" "jaune"`, alors l'appel `former_pile(F)` va renvoyer la pile `P` ci-contre :

`P =`
`"jaune"`
`"rouge"`
`"jaune"`
`"vert"`
`"rouge"`

3) Écrire une fonction `nb_elements` qui prend en paramètres une file `F` et un élément `elt` et qui renvoie le nombre de fois où `elt` est présent dans la file `F`.
Après appel de cette fonction, la file `F` doit avoir retrouvé son état d'origine.

4) Écrire une fonction `verifier_contenu` qui prend en paramètres une file `F` et trois entiers : `nb_rouge`, `nb_vert` et `nb_jaune`. Cette fonction renvoie le booléen **`True`** si `"rouge"` apparaît au plus `nb_rouge` fois, `"vert"` apparaît au plus `nb_vert` fois et `"jaune"` apparaît au plus `nb_jaune` fois dans `F`. Elle renvoie **`False`** sinon. On pourra utiliser les fonctions précédentes.