N.S.I. Term. – Exercices F10-1
Piles – Files (suite)

Exercice 1. Sujet 10 2022 - 22-NSIJ1LR1 – Exercice 1

Cet exercice porte sur les structures de données (listes, piles et files).

On cherche ici à mettre en place des algorithmes qui permettent de modifier l'ordre des informations contenues dans une file. On considère pour cela les structures de données abstraites de Pile et File définies par leurs fonctions primitives suivantes :

Pile :

- `creer_pile_vide()` renvoie une pile vide ;
- `est_pile_vide(p)` renvoie `True` si la pile `p` est vide, `False` sinon ;
- `empiler(p, element)` ajoute `element` au sommet de la pile `p` ;
- `depiler(p)` renvoie l'élément se situant au sommet de la pile `p` en le retirant de la pile `p` ;
- `sommet(p)` renvoie l'élément se situant au sommet de la pile `p` sans le retirer de la pile `p`.

File :

- `creer_file_vide()` renvoie une file vide ;
- `est_file_vide(f)` renvoie `True` si la file `f` est vide, `False` sinon ;
- `enfiler(f, element)` ajoute `element` dans la file `f` ;
- `defiler(f)` renvoie l'élément à la tête de la file `f` en le retirant de la file `f`.

On considère de plus que l'on dispose d'une fonction permettant de connaître le nombre d'éléments d'une file :

- `taille_file(f)` renvoie le nombre d'éléments de la file `f`.

On représentera les files par des éléments en ligne, l'élément de droite étant la tête de la file et l'élément de gauche étant la queue de la file. On représentera les piles en colonnes, le sommet de la pile étant le haut de la colonne.

La file suivante est appelée `f` :

4	3	8	2	1

La pile suivante est appelée `p` :

5
8
6
2

1. Les quatre questions suivantes sont indépendantes. Pour chaque question, on repartira de la pile `p` et de la file `f` initiales (présentées ci-dessus)
   a. Représenter la file `f` après l'exécution du code suivant.
   ```
   enfiler(f, defiler(f))
   ```
   b. Représenter la pile `p` après l'exécution du code suivant.
   ```
   empiler(p, depiler(p))
   ```
   c. Représenter la pile `p` et la file `f` après l'exécution du code suivant.
   ```
   for i in range(2):
       enfiler(f, depiler(p))
   ```

d. Représenter la pile `p` et la file `f` après l'exécution du code suivant.

```
for i in range(2):
    empiler(p, defiler(f))
```

2. On donne ici une fonction `mystere` qui prend une file en argument, qui modifie cette file, mais qui ne renvoie rien.

```
def mystere(f):
    p = creer_pile_vide()
    while not est_file_vide(f):
        empiler(p, defiler(f))
    while not est_pile_vide(p):
        enfiler(f, depiler(p))
    return p
```

Préciser l'état de la variable `f` après chaque boucle de la fonction `mystere` appliquée à la file | 1 | 2 | 3 | 4 | Indiquer le contenu de la pile renvoyée par la fonction.

3. On considère la fonction `knuth(f)` suivante dont le paramètre est une file :

```
def knuth(f):
    p=creer_pile_vide()
    N=taille_file(f)
    for i in range(N):
        if est_pile_vide(p):
            empiler(p, defiler(f))
        else:
            e = defiler(f)
            if e >= sommet(p):
                empiler(p, e)
            else:
                while not est_pile_vide(p) and e < sommet(p):
                    enfiler(f, depiler(p))
                empiler(p, e)
    while not est_pile_vide(p):
        enfiler(f, depiler(p))
```

a. Recopier et compléter le tableau ci-dessous qui détaille le fonctionnement de cet algorithme étape par étape pour la file `2,1,3`. Une étape correspond à une modification de la pile ou de la file. Le nombre de colonnes peut bien sûr être modifié.

f	2,1,3	2,1									
p	⌊ ⌋	⌊3⌋									

b. Que fait cet algorithme ?

Exercice 2. Sujet 06 2022 - 22-NSIJ1G11 – Exercice 2

Cet exercice porte sur les structures de données (files et la programmation objet en langage python)

Un supermarché met en place un système de passage automatique en caisse.
Un client scanne les articles à l'aide d'un scanner de code-barres au fur et à mesure qu'il les ajoute dans son panier.
Les articles s'enregistrent alors dans une structure de données.

La structure de données utilisée est une file définie par la classe `Panier`, avec les primitives habituelles sur la structure de file.
Pour faciliter la lecture, le code de la classe `Panier` n'est pas écrit.

```
class Panier():
  def __init__(self):
      """Initialise la file comme une file vide."""

  def est_vide(self):
      """Renvoie True si la file est vide, False sinon."""

  def enfiler(self, e):
      """Ajoute l'élément e en dernière position de la file,
    ne renvoie rien."""

  def defiler(self):
      """Retire le premier élément de la file et le renvoie."""
```

Le panier d'un client sera représenté par une file contenant les articles scannés.
Les articles sont représentés par des tuples `(code_barre, designation, prix, horaire_scan)` où

- `code_barre` est un nombre entier identifiant l'article ;
- `designation` est une chaine de caractères qui pourra être affichée sur le ticket de caisse ;
- `prix` est un nombre décimal donnant le prix d'une unité de cet article ;
- `horaire_scan` est un nombre entier de secondes permettant de connaitre l'heure où l'article a été scanné.

1. On souhaite ajouter un article dont le tuple est le suivant `(31002, "café noir", 1.50, 50525)`.
   Ecrire le code utilisant une des quatre méthodes ci-dessus permettant d'ajouter l'article à l'objet de classe `Panier` appelé `panier1`.

2. On souhaite définir une **méthode** `remplir(panier_temp)` dans la classe `Panier` permettant de remplir la file avec tout le contenu d'un autre panier `panier_temp` qui est un objet de type `Panier`.

Recopier et compléter le code de la méthode `remplir` en remplaçant chaque `.....` par la primitive de file qui convient.

```
def remplir(self, panier_temp):
    while not panier_temp.....  :
        article = panier_temp.....
        self.....(article)
```

3. Pour que le client puisse connaitre à tout moment le montant de son panier, on souhaite ajouter une **méthode** `prix_total()` à la classe `Panier` qui renvoie la somme des prix de tous les articles présents dans le panier.
Ecrire le code de la méthode `prix_total`. **Attention, après l'appel de cette méthode, le panier devra toujours contenir ses articles**.

4. Le magasin souhaite connaitre pour chaque client la durée des achats. Cette durée sera obtenue en faisant la différence entre le champ `horaire_scan` du dernier article scanné et le champ `horaire_scan` du premier article scanné dans le panier du client. Un panier vide renverra une durée égale à zéro. On pourra accepter que le panier soit vide après l'appel de cette méthode.
Ecrire une **méthode** `duree_courses` de la classe `Panier` qui renvoie cette durée.