Contrôle de connaissances

Nom :..... **Prénom :**.....

Exercice 1 : (10 points)

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

Structure de données abstraite : Pile
Utilise : Élément, Booléen

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

Structure de données abstraite : File
Utilise : Élément, Booléen

Opérations :
• `creer_file_vide` : $\varnothing \longrightarrow File$
`creer_file_vide`() renvoie une file vide
• `est_vide` : $File \longrightarrow Booleen$
`est_vide`() envoie `True` si file est vide, `False` sinon
• `enfiler` : $File, Element \longrightarrow \varnothing$
`enfiler`(`file`, `element`) ajoute `element` à la file `pile`
• `defiler` : $File \longrightarrow element$
`defiler`(`file`) envoie l'élément au sommet de la `file` en le retirant de la `file`

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

enfilement $\longrightarrow$ "rouge" "vert" "jaune" "rouge" "jaune" $\longrightarrow$ defilement

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

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

Correction

La boucle s'arrête quand la file est vide, donc après exécution du programme la file `F` est donc vide, et la pile contient :

P :
"rouge"
"vert"
"jaune"
"rouge"
"jaune"

(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 de cette fonction la file `F` doit avoir retrouvé son état d'origine.

Correction

L'idée est d'utiliser une autre file, de vider la file F dans la file F_2 en comptant le nombre de terme, puis on ré alimente la file F :

```
def taille_file(F):
  taille = 0
  F_2 = creer_file_vide()
  while not( est_vide(F)) :
    taille +=1
    enfiler(F_2, defiler(F))
  while not( est_vide(F_2)) :
    enfiler(F, defiler(F_2))
  return taille
```

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-dessous :

$$\texttt{P} = \begin{array}{|c|}\hline \text{"jaune"} \\ \text{"rouge"} \\ \text{"jaune"} \\ \text{"vert"} \\ \text{"rouge"} \\ \hline \end{array}$$

Correction

L'idée est d'utiliser deux piles, la première pour récupérer les éléments de la file, puis de les retournée dans la deuxième.

```
def taille_file(F):
  P_1 = creer_pile_vide()
  P_2 = creer_pile_vide()
  while not( est_vide(F)) :
    empiler(P_1, defiler(F))
  while not( est_vide(P_1)) :
    empiler(P_2, depiler(P_1))
  return P_2
```

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.

Correction

```
def nb_element (F,elt ) :
    f_temp = creer_file_vide()
    compteur = 0
    while not (est_vide( F)) :
        a = defile(F)
        if a == elt :
            compteur += 1
        enfile ( f_temp,a)
    while not(est_vide(f_temp)) :
        enfile(F, defile(f_temp))
    return compteur
```

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 dans la file `F`, "vert" apparaît au plus `nb_vert` fois dans la file `F` et "jaune" apparaît au plus `nb_jaune` fois dans la file `F`. Elle renvoie `False` sinon. On pourra utiliser les fonctions précédentes.

Correction

```
def verifier_contenu (F , nb_rouge , nb_vert , nb_jaune ) :
    return (nb_element (F,"rouge" ) <= nb_rouge ) and \
           (nb_element (F,"vert" ) <= nb_vert ) and \
           (nb_element (F,"jaune" ) <= nb_jaune )
```

Exercice 2 : (10 points)

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'element e derniere position de la file,
     ne renvoie rien."""
    def defiler(self):
    """Retire le premier element 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)`.
   Écrire le code utilisant une des quatre méthodes ci-dessus permettant d'ajouter l'article à l'objet de classe `Panier` appelé `panier1`.

   Correction

   Les instruction sont donc :

   ```
   >>> panier = Panier()
   >>> panier.enfiler((31002, "cafe noir", 1.50, 50525))
   ```

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)
```

Correction

```
def remplir(self,panier_temp) :
        while not panier_temp.est_vide() :
            article = panier_temp.defiler()
            self.enfiler(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.
Écrire le code de la méthode `prix_total`. Attention, après l'appel de cette méthode, le panier devra toujours contenir ses articles.

Correction

```
def prix_total(self) :
        panier_temp = Panier()
        somme = 0
        while not self.est_vide() :
            article = self.defiler()
            somme += article[2]
            panier_temp.enfiler(article)
        self.remplir(panier_temp)
        return somme
```

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.
Écrire une méthode `duree_courses` de la classe `Panier` qui renvoie cette durée.

Correction

```
def duree_course(self) :
        if self.est_vide() : # pas d'article
            return 0
        else :
            premier = self.defiler()
            if self.est_vide() : #un seul article
                return 0
            else :
                while not self.est_vide() :
                    article = self.defiler()
            # a la fin de la boucle, article est le dernier
                duree = article[3]-premier[3]
                return duree
```