

## Table des matières

1	France J2 2021	2
2	Amérique du nord 2021	3
3	Amérique du Nord J1 2022	5
4	Centre étrangers J1 2022	7
5	Mayotte J1 2022	8
6	France Sept 2022	10
7	Centres-étrangers J1 2023	13

## 1 France J2 2021

## Exercice 1

Cet exercice porte sur les structures de données linéaires.

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

**Q1.** 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

**Q2.** 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`.

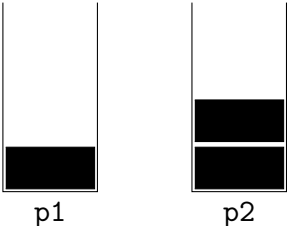
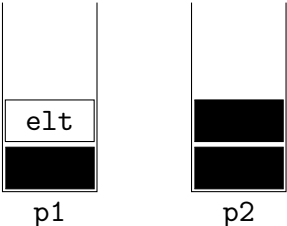
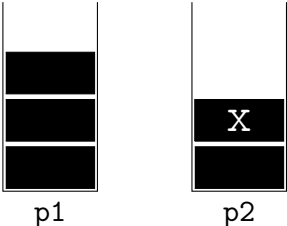
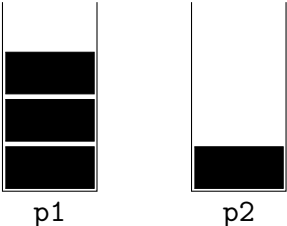
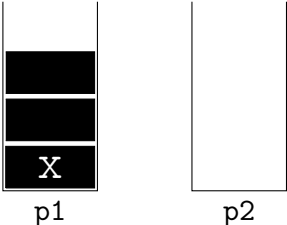
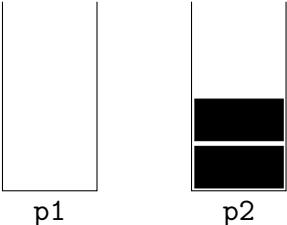
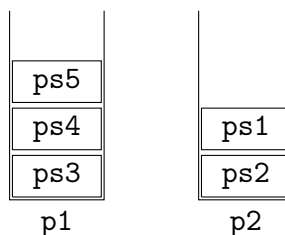
	État de la file avant	État de la file après
<code>enfiler(file, elt)</code>	 <p style="text-align: center;">p1                      p2</p>	 <p style="text-align: center;">p1                      p2</p>
<code>defiler(file)</code> cas où <code>p2</code> n'est pas vide	 <p style="text-align: center;">p1                      p2</p>	 <p style="text-align: center;">p1                      p2</p>
<code>defiler(file)</code> cas où <code>p2</code> est vide	 <p style="text-align: center;">p1                      p2</p>	 <p style="text-align: center;">p1                      p2</p>

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

Q3. On considère la situation représentée ci-dessous.



On exécute la séquence d'instructions suivante :

```

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.

Q4. 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.

## 2 Amérique du nord 2021

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
<b>Opérations :</b> <ul style="list-style-type: none"> <li>• <code>creer_pile_vide : <math>\emptyset \rightarrow \text{Pile}</math></code>  <code>creer_pile_vide()</code> renvoie une pile vide</li> <li>• <code>est_vide : Pile <math>\rightarrow</math> Booléen</code>  <code>est_vide(pile)</code> renvoie True si pile est vide, False sinon</li> <li>• <code>empiler : Pile, Élément <math>\rightarrow \emptyset</math></code>  <code>empiler(pile, element)</code> ajoute element à la pile pile</li> <li>• <code>depiler : Pile <math>\rightarrow</math> Élément</code>  <code>depiler(pile)</code> renvoie l'élément au sommet de la pile pile en le retirant de la pile</li> </ul>

**Structure de données abstraite : File****Utilise :** Élément, Booléen**Opérations :**

- `creer_file_vide :  $\emptyset \rightarrow \text{File}$`   
`creer_file_vide()` renvoie une file vide
- `est_vide : File  $\rightarrow$  Booléen`  
`est_vide(file)` renvoie True si file est vide, False sinon
- `enfiler : File, Élément  $\rightarrow \emptyset$`   
`enfiler(file, element)` ajoute element à la file file
- `defiler : File  $\rightarrow$  Élément`  
`defiler(file)` renvoie l'élément au sommet de la file file en le retirant de la file

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

$$\text{enfilement} \rightarrow \overline{\text{"rouge" "vert" "jaune" "rouge" "jaune"}} \rightarrow \text{défilement}$$

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))
4 
```

- (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.

```

1 def taille_file (F):
2     '''File -> Int'''

```

2. Ecrire 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  $F = \overline{\text{"rouge" "vert" "jaune" "rouge" "jaune"}}$  alors `former_pile(F)` va renvoyer la pile P ci-dessous :

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

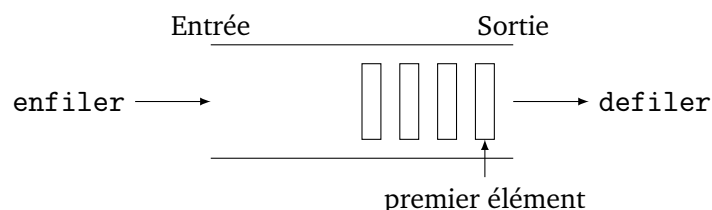
3. Ecrire une fonction `nb_elements` qui prend en paramètres une file `F` et un élément `elt` et qui renvoie le nombre de fois où `elt` est présent dans la file `F`. Après appel de cette fonction la file `F` doit avoir retrouvé son état d'origine.
4. Ecrire une fonction `verifier_contenu` qui prend en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

### 3 Amérique du Nord J1 2022

*Cet exercice porte sur files, les tableaux et les algorithmes associés.*

L'objectif de cet exercice est de travailler sur les températures relevées par une station météorologique. Les données sont enregistrées une fois par jour, à la même heure, et traitées dans l'ordre dans lequel elles arrivent.

On choisit d'utiliser une file. On rappelle qu'une file est une structure de données abstraites fondée sur le principe « premier arrivé, premier sorti ».



On munit la structure de données `File` des quatre fonctions primitives définies ci-dessous :

Structure de données abstraite : <code>File</code>
Utilise : <code>Elément</code> , <code>Booléen</code>
Opérations :
<ul style="list-style-type: none"> <li>• <code>creer_file_vide : ∅ → File</code>  <code>creer_file_vide()</code> renvoie une file vide</li> <li>• <code>est_vide : File → Booléen</code>  <code>est_vide(F)</code> renvoie <code>True</code> si la file <code>F</code> est vide, <code>False</code> sinon</li> <li>• <code>enfiler : File, Elément → ∅</code>  <code>enfiler(F, element)</code> ajoute <code>element</code> en entrée de la file <code>F</code></li> <li>• <code>defiler : File → Elément</code>  <code>defiler(F)</code> renvoie l'élément en sortie de la file <code>F</code> (premier élément) en le retirant de la file <code>F</code></li> </ul>

1. Les températures relevées ont été 15, puis 17, puis 14.

(a) Parmi les quatre propositions suivantes, indiquer celle qui représente correctement cette file :

**Proposition 1 :**

Entrée	Sortie
15    17    14	

Le premier élément est 14

**Proposition 2 :**

Entrée		Sortie
14	17	15

      Le premier élément est 15

**Proposition 3 :**

Entrée		Sortie
15	17	14

      Le premier élément est 15

**Proposition 4 :**

Entrée		Sortie
14	17	15

      Le premier élément est 14

(b) En utilisant les fonctions primitives précédentes, donner les instructions permettant de créer cette file.

2. On appelle longueur d'une file le nombre d'éléments qu'elle contient.

La fonction `longueur_file` prend en paramètre une file `F` et renvoie sa longueur `n`.

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

**Exemple :** si `F = 

10	10	12	12
----	----	----	----

Recopier et compléter le programme Python suivant qui permet d'implémenter la fonction `longueur_file` (les trois points ... peuvent correspondre à une ou plusieurs lignes de programme).

```

1 def longueur_file(F):
2     """File -> Int"""
3     G = creer_file_vide() # file temporaire
4     n = 0 # initialisation du nombre d'éléments
5     while not(est_vide(F)):
6         ...
7     while not(est_vide(G)): # reconstruction de la file initiale
8         ...
9     return ...

```

3. On s'intéresse à la variation de température d'un jour sur l'autre.

Par exemple, lorsque les températures relevées sont dans l'ordre d'arrivée 15, 17 et 14, les variations sont 2 et -3.

Recopier et compléter le programme Python implémentant la fonction `variations` qui prend en paramètre une file non vide `F` et qui renvoie le tableau `tab` contenant les variations successives, ou un tableau vide si la file `F` ne contient qu'une seule température. Il n'est pas demandé ici que la file `F` retrouve son état d'origine après appel de la fonction `variations`.

Exemple : si `F` est la file qui contient dans l'ordre des relevés les valeurs 15, 17 et 14, alors l'appel `variations(F)` renvoie `[2, -3]`.

Dans le code de la fonction, les trois points ... peuvent correspondre à une ou plusieurs lignes de programme.

```

1 def variations(F):
2     """File -> Tableau"""
3     taille = longueur_file(F)
4     if taille == 1:
5         ...
6     else:

```

```

7     tab = [0 for k in range(taille - 1)]
8     element1 = defiler(F)
9     for i in range(taille - 1):
10         element2 = defiler(F)
11         ...
12     return ...

```

4. Ecrire une fonction `nombre_baisses` qui prend en paramètre un tableau `tab`, non vide, des variations des températures et qui renvoie un p-uplet contenant le nombre de jours où la température a baissé par rapport au jour précédent (soit le nombre de valeurs strictement négatives de `tab`), ainsi que la baisse journalière la plus importante (soit la valeur minimale de `tab`). S'il n'y a aucune baisse (toutes les valeurs de `tab` sont positives), la fonction renvoie le p-uplet (0, 0).

**Exemples :**

- `nombre_baisses([1, -4, 2, -1, 3])` renvoie (2, -4).
- `nombre_baisses([1, 5, 3, 1])` renvoie (0, 0).

## 4 Centre étrangers J1 2022

*Cet exercice porte sur les structures de données 'file et programmation orientée objet en 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.

```

1 class Panier():
2     def __init__(self):
3         """Initialise la file comme une file vide."""
4
5     def est_vide(self):
6         """Renvoie True si la file est vide, False sinon."""
7
8     def enfiler(self, e):
9         """Ajoute l'élément e en dernière position de la file, ne renvoie rien."""
10
11     def defiler(self):
12         """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 chaîne 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 connaître 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.

```

1 def remplir(self, panier_temp):
2     while not panier_temp. .... :
3         article = panier_temp. ....
4         self. ....(article)

```

3. Pour que le client puisse connaître 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 connaître 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.

## 5 Mayotte J1 2022

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. Par exemple :

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.

```

1 def mystere(f):
2     p = creer_pile_vide()
3     while not est_file_vide(f):
4         empiler(p, defiler(f))
5     while not est_pile_vide(p):
6         enfiler(f, depiler(p))
7     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* suivante dont le paramètre est une file :

```

1 def knuth(f):
2     p=creer_pile_vide()
3     N=taille_file(f)
4     for i in range(N):
5         if est_pile_vide(p):
6             empiler(p, defiler(f))
7         else:
8             e = defiler(f)
9             if e >= sommet(p):
10                empiler(p, e)
11            else:
12                while not est_pile_vide(p) and e < sommet(p):
13                    enfiler(f, depiler(p))
14                empiler(p, e)
15     while not est_pile_vide(p):
16         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	<table><tr><td>2</td><td>1</td><td>3</td></tr></table>	2	1	3	<table><tr><td>2</td><td>1</td></tr></table>	2	1						
2	1	3											
2	1												
p	<table><tr><td></td></tr></table>		<table><tr><td>3</td></tr></table>	3									
3													

(b) Que fait cet algorithme ?

## 6 France Sept 2022

**Rappel :** une file est une structure de données abstraite fondée sur le principe « premier arrivé, premier servi ».

1. Laquelle de ces deux situations est associée à une structure de file ?

**Situation 1 :** « Je cuisine des crêpes. Dès qu'une crêpe est faite, je la place sur un plat. Chaque nouvelle crêpe est placée sur la crêpe précédente. Quand je vais manger une de ces crêpes, je commencerai par la crêpe située en haut de mon plat. »

**Situation 2 :** « Je dispose d'une imprimante placée en réseau dans ma salle de classe équipée d'ordinateurs, tous en réseau. Tous les élèves présents ont accès à cette imprimante, via le réseau. A la fin de la séance, les élèves envoient leur production à l'impression. Les documents sont imprimés dans l'ordre d'arrivée. »

On modélise la gestion de l'attente à une caisse de supermarché. Les clients sont associés à une file. Les personnes prioritaires passeront devant les autres clients sans attendre. Nous ne tenons pas compte dans cet exercice de graduation dans les « priorités ». Nous ne tenons pas compte également de personnes arrivant ensemble en caisse : il y aura toujours un des deux clients avant l'autre. On appelle « première personne » dans la queue la première personne qui est juste derrière le client en train de payer ses articles en caisse. En d'autres termes, le client qui règle ses articles ne compte plus, puisqu'il n'attend plus dans la queue.

Voici les règles appliquées :

- la première personne arrivée se place dans la queue ;
- le contrôleur « relations clients » du supermarché vérifie les priorités « clients » ;
- si une personne dispose d'un accès prioritaire, elle passe en position 1 et, de ce fait, tout le reste des clients dans l'attente rétrograde d'une place ;
- si deux personnes sont prioritaires, la deuxième arrivée se placera derrière la première arrivée « prioritaire » et ainsi de suite avec tout nouveau prioritaire.

**Exemple :** à un instant  $t$ , la file est dans l'état ci-dessous :

5	4	3	2	1
Client4	Client3	Prioritaire	Client2	Client1

La réorganisation grâce au contrôleur « relations clients » se met en place : les clients Client1 et Client2 font un pas de côté, de même pour les personnes derrière le client Prioritaire en respectant leur ordre d'arrivée :

Client4	Client3		Client2	Client1
		Prioritaire		

Le client Prioritaire s'avance et se retrouve en position 1. Puis la file finale se réorganise :

Client4	Client3	Client2	Client1	Prioritaire
---------	---------	---------	---------	-------------

Nous utiliserons uniquement les quatre fonctions primitives suivantes pour la suite des questions :

**Structure de données abstraite : File****Utilise :** Élément, Booléen**Opérations :**

- **créer\_file\_vide** :  $\emptyset \rightarrow \text{File}$   
    **créer\_file\_vide()** renvoie une file vide
- **est\_vide** :  $\text{File} \rightarrow \text{Booléen}$   
    **est\_vide(F)** renvoie True si la file F est vide, False sinon
- **enfiler** :  $\text{File}, \text{Élément} \rightarrow \emptyset$   
    **enfiler(F, element)** ajoute element en entrée de la file F
- **defiler** :  $\text{File} \rightarrow \text{Élément}$   
    **defiler(F)** renvoie l'élément en tête de la file F (premier élément) en le retirant de la file F

2. On considère que le contenu de la file F est le suivant :

Queue			Tête	
Client4	Prioritaire	Client3	Client2	Client1

(a) On considère la file V définie à l'aide du code ci-dessous :

```

1 V = creer_file_vide()
2 val = defiler(F)
3 while not est_vide(F) and val != 'Prioritaire':
4     enfiler(V, val)
5     val = defiler(F)

```

Quels seront les contenus des files V et F et de la variable val à l'issue de ces instructions ?

(b) On considère la fonction longueur\_file ci-dessous. Le but de cette fonction est de renvoyer le nombre d'éléments d'une file donnée en paramètre. A la fin du programme, cette file doit avoir retrouvé son état d'origine. Compléter cette fonction.

```

1 def longueur_file(F):
2     V = creer_file_vide()
3     n = 0
4     while not est_vide(F):
5         n = .....
6         val = defiler(F)
7         enfiler(V, val)
8     while not est_vide(V):
9         .....
10        .....
11    return n

```

(c) Ecrire une fonction compter\_prio qui prend en paramètre une file F. Cette fonction renvoie le nombre de personnes prioritaires dans la file d'attente à l'instant  $t$ . La file F doit être identique à celle du départ en fin d'exécution de la fonction.

## 7 Centres-étrangers J1 2023

*Cet exercice porte sur les structures de Files*

**Simon** est un jeu de société électronique de forme circulaire comportant quatre grosses touches de couleurs différentes : rouge, vert, bleu et jaune.

Le jeu joue une séquence de couleurs que le joueur doit mémoriser et répéter ensuite. S'il réussit, une couleur parmi les 4 est ajoutée à la fin de la séquence. La nouvelle séquence est jouée depuis le début et le jeu continue. Dès que le joueur se trompe, la séquence est vidée et réinitialisée avec une couleur et une nouvelle partie commence.

Exemple de séquence jouée : rouge → bleu → rouge → jaune → bleu



Dans cet exercice nous essaierons de reproduire ce jeu.

Les quatre couleurs sont stockées dans un tuple nommé couleurs :

couleurs = ("bleu", "rouge", "jaune", "vert")

Pour stocker la séquence à afficher nous utiliserons une structure de file que l'on nommera sequence tout au long de l'exercice.

La file est une structure linéaire de type FIFO (*First In First Out*). Nous utiliserons durant cet exercice les fonctions suivantes :

Structure de données abstraite : File	
créer_file_vide()	: renvoie une file vide
est_vide(f)	: renvoie <b>True</b> si f est vide, <b>False</b> sinon
enfiler(f, element)	: ajoute element en queue de f
defiler(f)	: retire l'élément en tête de f et le renvoie
taille(f)	: renvoie le nombre d'éléments de f

En fin de chaque séquence, le Simon tire au hasard une couleur parmi les 4 proposées. On utilisera la fonction randint(a,b) de la bibliothèque random qui permet d'obtenir un nombre entier compris entre a inclus et b inclus pour le tirage aléatoire.

Exemple : randint(1,5) peut renvoyer 1, 2, 3, 4 ou 5.

**Q1.** Recopier et compléter, sur votre copie, les « ... » des lignes 3 et 4 de la fonction ajout(f) qui permet de tirer au hasard une couleur et de l'ajouter à une séquence.

La fonction ajout prend en paramètre la séquence f ; elle renvoie la séquence f modifiée (qui intègre la couleur ajoutée au format chaîne de caractères).

```

1 def ajout(f):
2     couleurs = ("bleu", "rouge", "jaune", "vert")
3     indice = randint(..., ...)
4     enfiler(..., ...)
5     return f

```

En cas d'erreur du joueur durant sa réponse, la partie reprend au début ; il faut donc vider la file sequence pour recommencer à zéro en appelant vider(sequence) qui permet de rendre la file sequence vide sans la renvoyer.

**Q2.** Écrire la fonction vider qui prend en paramètre une séquence f et la vide sans la renvoyer.

Le Simon doit afficher successivement les différentes couleurs de la séquence.

Ce rôle est confié à la fonction affich\_seq(sequence), qui prend en paramètre la file de couleurs sequence, définie par l'algorithme suivant :

- ▷ on ajoute une nouvelle couleur à sequence ;
- ▷ on affiche les couleurs de la séquence, une par une, avec une pause de 0,5 s entre chaque affichage.

Une fonction affichage(couleur) (dont la rédaction n'est pas demandée dans cet exercice) permettra l'affichage de la couleur souhaitée avec couleur de type chaîne de caractères correspondant à une des 4 couleurs.

La temporisation de 0,5 s sera effectuée avec la commande time.sleep(0.5).

Après l'exécution de la fonction affich\_seq, la file sequence ne devra pas être vidée de ses éléments.

**Q3.** Recopier et compléter, sur la copie, les « ... » des lignes 4 à 10 de la fonction `affich_seq(sequence)` ci-dessous :

```

1 def affich_seq(sequence):
2     stock = creer_file_vide()
3     ajout(sequence)
4     while not est_vide(sequence) :
5         c = ...
6         ...
7         time.sleep(0.5)
8         ...
9     while ... :
10        ...

```

**Q4.** Cette question est indépendante des précédentes : bien qu'elle fasse appel aux fonctions construites précédemment, elle peut être résolue même si le candidat n'a pas réussi toutes les questions précédentes.

Nous allons ici créer une fonction `tour_de_jeu(sequence)` qui gère le déroulement d'un tour quelconque de jeu côté joueur. La fonction `tour_de_jeu` prend en paramètre la file de couleurs `sequence`, qui contient un certain nombre de couleurs.

- ▷ Le jeu électronique Simon commence par ajouter une couleur à la séquence et affiche l'intégralité de la séquence.
- ▷ Le joueur doit reproduire la séquence dans le même ordre. Il choisit une couleur via la fonction `saisie_joueur()`.
- ▷ On vérifie si cette couleur est conforme à celle de la séquence.
- ▷ S'il s'agit de la bonne couleur, on poursuit sinon on vide `sequence`.
- ▷ Si le joueur arrive au bout de la séquence, il valide le tour de jeu et le jeu se poursuit avec un nouveau tour de jeu, sinon le joueur a perdu et le jeu s'arrête.

La fonction `tour_de_jeu` s'arrête donc si le joueur a trouvé toutes les bonnes couleurs de `sequence` dans l'ordre, ou bien dès que le joueur se trompe.

Après l'exécution de la fonction `tour_de_jeu`, la file `sequence` ne devra pas être vidée de ses éléments en cas de victoire.

**(a)** Afin d'obtenir la fonction `tour_de_jeu(sequence)` correspondant au comportement décrit ci-dessus, recopier le script ci-dessous et :

- ▷ Compléter le « ... »
- ▷ Choisir parmi les propositions de syntaxes suivantes lesquelles correspondent aux ZONES A, B, C, D, E et F figurant dans le script et les y remplacer (il ne faut donc en choisir que six parmi les onze) :

```

vider(sequence)
defiler(sequence)
enfiler(sequence, c_joueur)
enfiler(stock, c_seq)
enfiler(sequence, defiler(stock))
enfiler(stock, defiler(sequence))
affich_seq(sequence)
while not est_vide(sequence):
while not est_vide(stock):
if not est_vide(sequence):
if not est_vide(stock):

```

```
1 def tour_de_jeu(sequence):  
2     # ZONE A  
3     stock = creer_file_vide()  
4     while not est_vide(sequence) :  
5         c_joueur = saisie_joueur()  
6         c_seq = # ZONE B  
7         if c_joueur ... c_seq :  
8             # ZONE C  
9         else:  
10            # ZONE D  
11    # ZONE E  
12    # ZONE F
```

- (b) Proposer une modification pour que la fonction se répète si le joueur trouve toutes les couleurs de la séquence (dans ce cas, une nouvelle couleur est ajoutée) ou s'il se trompe (dans ce cas, la séquence est vidée et se voit ajouter une nouvelle couleur). On pourra ajouter des instructions qui ne sont pas proposées dans la question (a).