

## DM\_Pile\_File

*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.

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`.

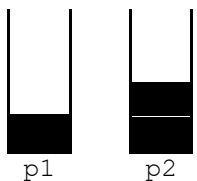
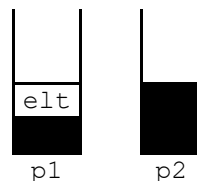
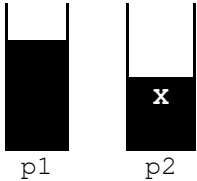
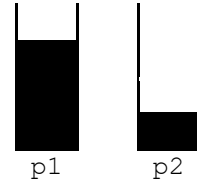
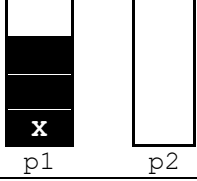
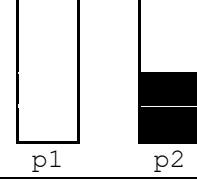
On dispose déjà d'une fonction `retirer(lst)` qui renvoie l'élément `lst[0]` puis le supprime de la liste `lst`. Écrire en Python le code d'une fonction `ajouter(lst, proc)` qui ajoute à la fin de la liste `lst` le nouveau processus en attente `proc`.

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
<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>

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

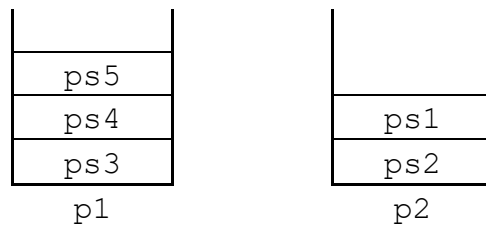
```

Entrée [ ]:
#Question 2 :
def ajouter(lst, proc) :
    lst[-1].append(proc)

Entrée [ ]:
def ajouter(lst, proc) :
    lst[1].append(proc)

```

3. 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)

```

Visual feedback boxes during execution:

- After `enfiler(file, ps6)`: `p1 : ps6 p2 : ps5`
- After `defiler(file)`: `p1 : ps6 p2 : ps5` (no change)
- After `defiler(file)`: `p1 : ps6 p2 : ps5` (no change)
- After `defiler(file)`: `p1 : ps6 p2 : ps5` (no change)
- After `enfiler(file, ps7)`: `p1 : p2 : ps6 ps5 ps4`

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.