Structures de données : Piles et queues

Ce TD propose une approche simplifiée dans laquelle nous ne tenterons pas d'implémenter une classe objet en python. Nous allons donc représenter nos structures de données [pile puis queue] par un tuple non modifiable en python contenant les attributs. Les méthodes sur l'objet seront des fonctions.
Dans les deux cas on peut remplir nos piles avec n'importe quels objets python. Je vous recommande de choisir des entiers ça n'a aucune importance ici. En revanche l'objet None sera réservé pour indiquer que la structure de données est vide.

Important : les fonctions en python ne permettent pas de mettre à jour automatiquement les structures de données comme le fait naturellement une class objet. A chaque modification de l'objet il faudra le mettre à jour « à la main ». Pour cela vous devez déclarez votre objet de façon global dans le corps de fonction. Ex : global maPile. Les modifications sur l'objet seront alors prises en compte hors de la fonction.

A - Structure de pile

On peut imaginer la pile comme une pile d'assiettes : on accède qu'à l'assiette au sommet de la pile que ce soit pour poser ou prendre une assiette.
Le premier posé est aussi le dernier retiré selon la logique FILO : « **First In Last Out** ».

Cette structure de base est une des plus courantes en informatique [mémoire, récursivité] et sert de base à de nombreux algorithmes.

1 - Créer un objet appelée maPile dans le script global.

La structure de données en mémoire va prendre la forme suivante :
maPile = (0 , [ ]) #objet Pile => (height, pile)

- Il n'y aura qu'une variable **pile** dans ce problème. Toutes les méthodes concernant les listes sont évidemment interdites pour la suite. pile est initialisée comme liste vide [ ].

- La variable **height**, qui maintiendra à jour la hauteur de la pile, ou nombre d'éléments empilés est initialisée à 0.

On dit que height et pile sont des attributs de la structure de données ils doivent être mis jour à chaque modification de la pile.

2 - Vérification que la pile n'est pas vide.

Cette opération est indispensable pour ne pas rechercher des données qui n'existent pas.
Dans le cas contraire des erreurs de type indexError seraient déclenchées.
Créer une fonction **isEmpty**() qui dit si la pile est vide. La fonction **len**() des listes est interdite.

Point technique :

Dans la suite on veut remplir notre structure de données puis éventuellement la vider. Cela suppose de récupérer les données de l'objet et d'enregistrer ces modifications dans le script global. Pour cela vous pourrez introduire en début de corps de fonction le code suivant :

```
def maFonction(variablesEventuelles):
    global maPile              #permet de modifier l'objet maPile depuis le corps de fonction
    height, pile = maPile      #récupération des attributs (car le tuple n'est pas modifiable)

                               *****   votre code ****

    maPile=(height,pile)      #mise à jour de votre structure de données.
```

3 - Méthode push(elem) : on pose une assiette !

Ecrire une fonction **push**(elem) qui ajoute un élément sur la pile mais qui ne renvoie rien.
Il faut mettre à jour la structure de données.

4 - Méthode pop() : on enlève une assiette !

Ecrire une fonction **pop**() qui renvoie l'élément en haut de la pile et le retire de la pile.
Attention : si la pile est vide la méthode pop() ne doit rien renvoyer.
Il faut mettre à jour la structure de données.

5 - Méthode lookUp() : on regarde juste quel est l'élément sur la pile.

Ecrire une fonction **lookUp**() qui renvoie l'élément en haut de la pile mais ne le retire de la pile.
Attention : si la pile est vide la méthode pop() ne doit rien renvoyer.
pas de mise à jour.

6 - Exercice : Trouver le k-ième élément sur la pile

Ecrire une fonction kPop(k) qui sorte le k-ième élement sur la pile et préserve le reste de la pile. Vous n'avez le droit pour cela qu'aux fonctions précédentes.

Rq : L'utilisation des méthodes permet de modifier l'objet sans passer par une déclaration en global.

B - Structure de queue

On peut imaginer la queue comme une file d'attente ou des perles enfilées sur une ficelle :
on entre dans la file par la gauche et on sort par la droite selon logique FIFO : « First In First Out ».
Cette structure de base est aussi l'une des plus courantes en informatique [buffer, router] et sert de base à de nombreux algorithmes.

Nous reprenons donc la même démarche ici, toutefois la gestion en mémoire peut-être très différente car la file se remplit d'un coté et se vide de l'autre : les deux frontières sont « mobiles ».
Cette particularité sera ici masquée par le fait que nous utilisons des listes dynamiques python.
[Cf - cours gestion des tableaux].

1 - Créer un objet appelée maFile dans le script global.

La structure de données en mémoire va prendre la forme suivante :
maFile = (0 , [ ]) #objet queue => (length, file)

- Il n'y aura qu'une variable f**ile** dans ce problème. Toutes les méthodes concernant les listes sont évidemment interdites pour la suite. file est initialisée comme liste vide [ ].

- La variable **length**, qui maintiendra à jour la longueur de la file, ou nombre d'éléments dans la queue est initialisée à 0.

On dit que length et file sont des attributs de la structure de données ils doivent être mis jour à chaque modification de la pile.

2 - Vérification que la pile n'est pas vide.

Cette opération est indispensable pour ne pas rechercher des données qui n'existent pas.
Dans le cas contraire des erreurs de type indexError seraient déclenchées.
Créer une fonction **isEmpty**() qui dit si la file est vide. La fonction **len**() des listes est interdite.

Point technique :

Dans la suite on veut remplir notre structure de données puis éventuellement la vider. Cela suppose de récupérer les données de l'objet et d'enregistrer ces modifications dans le script global. Pour cela vous pourrez introduire en début de corps de fonction le code suivant :

```
def maFonction(variablesEventuelles):
    global maFile              #permet de modifier l'objet maFile depuis le corps de fonction
    length, file = maFile      #récupération des attributs (car le tuple n'est pas modifiable)

                          *****   votre code ****

    maFile=(length, file)     #mise à jour de votre structure de données.
```

3 - Méthode push(elem) : on entre dans la queue par la gauche !

Ecrire une fonction **push**(elem) qui rentre un élément dans la file par la gauche mais qui ne renvoie rien. Il faut mettre à jour la structure de données.

4 - Méthode pull() : on sort de la queue par la droite !

Ecrire une fonction **pull**() qui renvoie l'élément en bout de file à droite et le retire.
Attention : si la file est vide la méthode pull() ne doit rien renvoyer.
Il faut mettre à jour la structure de données.

5 - Exercice : rétrograder de k-place !

Ecrire une fonction **retrograder**(i,k) qui recule le i-ème élément de k places si l'opération est possible et qui renvoie un message d'erreur sinon. Cette fonction ne renvoie rien.
A nouveau, seules les fonctions de la parties B sont autorisées et pas les méthodes de liste : ~~file[i], file[k] = file[k], file[i].~~

Idée : Envisager la queue comme des perles enfilées sur une ficelle : on ne peut les sortir que par la droite et ne les rentrer que par la gauche ce qui ne simplifie pas les choses.
Rq : L'utilisation des méthodes permet de modifier l'objet sans passer par une déclaration en global.

Fin