TP n° 5 : les files

1. Manipulations de base des files

1.1. Présentation

Soit une file F composée des éléments suivants : 12, 14, 8, 7, 19 et 22 (la tête de la file est 22, la queue est 12).
Pour chaque exemple ci-dessous on repart de la file d'origine F.

- *defiler(F)*. Renvoie 22 et la file F est maintenant composée des éléments suivants : 12, 14, 8, 7 et 19 (la tête de la file est 19).
- *enfiler(F, 42)*. La file F est maintenant composée des éléments suivants : 42, 12, 14, 8, 7, 19, et 22.
- *tete(F)*. Renvoie 22, la file F n'est pas modifiée.
- Si on applique *defiler(F)* 6 fois de suite, alors file_vide(F) renvoie vrai.
- Si on applique *defiler(F)* une fois, alors *taille(F)* renvoie 5.

1.2. Exercice

Soit une file F composée des éléments suivants : 12, 8, 5, 23 et 30 (la tête de la file est 30).
Décrire l'effet sur la file à chaque étape. On part, cette fois, de l'état de la file de l'état précédent.

1. *defiler(F)*
2. *tete(F)*
3. *enfiler(F, 18)*
4. *taille(F)*
5. *enfiler(F, 42), defiler(F), defiler(F), taille(P)*

2. Implémentation en utilisant des fonctions

2.1. Présentation

Dans python, la méthode la plus simple pour gérer une file, consiste à utiliser un tableau et ses fonctions associées pour stocker les éléments de la file.

Rappel : même si on utilise des tableau pour implémenter le TAD file en Python, il faut avoir conscience que certaines opérations associées aux tableau ne sont pas des opérations de files. Par exemple, faire un accès à n'importe lequel de ses éléments n'est pas une opération de file.

2.2. Exercice

L'objectif est de créer une file en utilisant uniquement un tableau, ainsi que les méthodes *append()* et *pop(0)*. En particulier, la fonction *len()* ne devra pas être utilisée.

Mis à part avec les méthodes *creer_file()*, *enfile()* et *defile()*, la file ne devra pas être modifiée. Il faut parfois la régénérer à la fin si la méthode l'a modifiée.

Ouvrir le programme *TP05_files_fonctions.py*. Remplacer progressivement les instructions *pass* pour implémenter les fonctions suivantes :

- *creer_file()* fonction qui retourne un tableau vide.
- *est_vide(f)* fonction qui retourne *True* si le tableau est vide.
- *enfile(f, a)* fonction qui ajoute l'élément *a* au tableau *f*.
- *defile(f)* fonction qui supprime le premier élément du tableau *f*.
- *tete(f)* fonction qui retourne le premier élément du tableau.
- *queue(f)* fonction qui retourne le dernier élément du tableau.
- *taille(f)* fonction qui renvoie la taille de la file.

Le programme pourra être testé avec les instructions suivantes :

```
if __name__ == '__main__':
    # tester les fonctions
    f = creer_file()
    enfile(f, 3)
    enfile(f, 5)
    enfile(f, 8)
```

```
7       enfile(f, 1)
8       enfile(f, 15)
9       print(f)
10      defile(f)
11      a = defile(f)
12      print(a)
13      print(f)
14      taille(f)
15      queue(f)
16      tete(f)
```

S'il y a le temps, créer une fonction *echange_haut_bas()* qui échange la tête et la queue de la file *f*.

3. Implémentation en POO

Objectif

Créer une classe d'objet File qui simule les fonctionnalités d'une file avec un tableau.

Cahier des charges

Ouvrir le programme *TP04_piles_classe.py*. La classe *Pile* contiendra les méthodes :

- *est_vide()*
- *enfile()* en utilisant la méthode *append()* ;
- *defile()* en utilisant la méthode *pop()* ;
- *tete()* ;
- *queue()* ;
- *taille()* en utilisant la fonction *len()*.

De plus, il faudra gérer la méthode *__str__* pour obetenir ce type d'affichage de la pile.

```
**********
Tête de la file
3
5
8
Queue de la file
**********
```

Test

- Créer une file *f*, puis empiler successivement les valeurs 3 puis 5 et enfin 8.
- Afficher cette file *f*.
- Afficher la taille de cette pile.
- Défiler la premier valeur, puis afficher *f*.
- Recommencer pour les deux valeurs suivantes.

Remarque sur notre implémentation

Notre implémentation répond parfaitement à l'interface qui était demandée. Mais si le « cahier des charges » du TAD File obligeait à ce que les opérations *enfile()* et *defile()* aient lieu en temps constant (en $O(1)$), notre implémentation ne conviendrait pas.

En effet, notre méthode *defile()* agit en temps linéaire ($O(n)$) et non pas en temps constant. L'utilisation de la structure de « liste » de Python (les tableaux dynamiques) provoque, lors de l'instruction s*elf.data.pop(0)*, un redimensionnement de la liste, qui voit disparaître son premier élément. Chaque élément doit être recopié dans la case qui précède, avant de supprimer la dernière case. Ceci nous coûte un temps linéaire.

Capacités exigibles

- Écrire plusieurs implémentations d'une même structure de données.