Exercice 1. Structures de données

Quelle structure de données choisir pour chacune des tâches suivantes ?

1. Répertoire téléphonique.
   ☐ pile ☐ file ☐ dictionnaire
2. Historique d'une navigateur internet.
   ☐ pile ☐ file ☐ dictionnaire
3. Envoi des données au serveur d'impression.
   ☐ pile ☐ file ☐ dictionnaire

Exercice 2 . Deux files !

Dans cet exercice, on peut réaliser les trois opérations suivantes :

- `CreerFileVide()` : retourne une file vide.
- `Enfiler(F, e)` : ajoute *e* à la fin de la file *F*.
- `Defiler(F)` : l'élément situé en tête de file *F* est retiré et renvoyé.

Déterminer le contenu des files F1 et F2 après exécution des instructions ci-contre.

```
f1 = CreerFileVide()
Enfiler(f1, 'L')
Enfiler(f1, 'A')
Enfiler(f1, 'R')
Enfiler(f1, 'J')
Enfiler(f1, 'E')
f2 = CreerFileVide()
Enfiler(f2, 'V')
Enfiler(f2, 'I')
Enfiler(f2, 'A')
Enfiler(f2, 'N')
Enfiler(f2, Defiler(f2))
Enfiler(f2, Defiler(f2))
Enfiler(f1, Defiler(f2))
Enfiler(f1, Defiler(f2))
for i in range(3):
    Enfiler(f2, Defiler(f1))
```

Exercice 3 . La classe

Nous avons vu en cours que si le type File n'existe pas en Python, nous pouvons le créer grâce à la programmation orientée objet.

```
class File:
    def __init__(self):
        self.taille = 0
        self.contenu = []
    def enfile(self, e):
        self.taille = self.taille + 1
        self.contenu.append(e)
    def defile(self):
        if self.taille > 0:
            self.taille = self.taille - 1
            return self.contenu.pop(0)
```

1. Ajouter une méthode `__str__` permettant de renvoyer une chaîne de caractères constituée des éléments de la file dans l'ordre, premier élément en tête, séparés par le caractère `chr(8592)`.

```
>>> fil = File()
>>> str(fil)
''
>>> fil.enfile(1)
>>> str(fil)
'1'
>>> fil.enfile(2)
>>> str(fil)
'1←2'
>>>
```

2. Ajouter une méthode `__len__` permettant de renvoyer la longueur de la file.

```
>>> str(fil)
'A←B←C←D←E'
>>> len(fil)
5
```

3. Créer une méthode `estVide` qui renvoie True si la file est vide et False dans le cas contraire.

```
>>> L=File()
>>> L.estVide()
True
>>> L.enfile(1)
>>> L.estVide()
False
```

4. Compléter la classe File d'une méthode `lireTete` qui renvoie le premier élément de la file sans le supprimer.

```
>>> str(F1)
'A←B←C←D←E'
>>> F1.lireTete()
'A'
>>> str(F1)
'A←B←C←D←E'
>>>
```

Exercice 4. Les objets

Dans cet exercice, on ne doit utiliser que les méthodes de la classe File de l'exercice précédent et aucune autre méthode des listes Python (~~append~~, ~~pop~~, ~~len~~…)

1. Écrire une fonction `alea(n)` qui renvoie une file de n entiers aléatoires compris entre 0 et 100.

```
>>> F1=alea(10)
>>> str(F1)
'47←9←93←92←58←90←1←10←17←22'
```

2. Créer une fonction `gardePairs(F)` qui utilise la classe implémentée dans l'exercice 3 et qui reçoit file de n entiers et ne conserve que les nombres pairs. Cette fonction ne doit pas créer de nouvelle file.

```
>>> str(F1)
'47←9←93←92←58←90←1←10←17←22'
>>> gardePairs(F1)
>>> str(F1)
'92←58←90←10←22'
```

3. Écrire une fonction `dupliquer(F)` qui renvoie une nouvelle file identique à F. La file F ne doit pas être modifiée après exécution de la fonction.

```
>>> str(F1)
'N←S←I'
>>> F2=dupliquer(F1)
>>> str(F2)
'N←S←I'
```

4. Après avoir créé la file F1, on saisit les instructions ci-contre dans la console.

   - Quel est l'intérêt de la fonction `dupliquer(F)` ?
   - Pourquoi ne pas se contenter de saisir F2 = F1 ?
   - Quel est le nom de l'effet observé ?

```
>>> str(F1)
'1←2←3'
>>> F2=dupliquer(F1)
>>> F3=F1
>>> F1.defile()
1
>>> str(F1)
'2←3'
>>> str(F2)
'1←2←3'
>>> str(F3)
'2←3'
>>>
```

Exercice 5. Implémenter une file avec une liste chaînée

On veut écrire une classe pour gérer une file à l'aide d'une liste chaînée.

Le code est à télécharger depuis Moodle : exo5.py

On dispose d'une classe `Maillon` permettant la création d'un maillon de la chaîne, celui-ci étant constitué d'une valeur et d'une référence au maillon suivant de la chaîne :

```
class Maillon :
    def __init__(self,v) :
        self.valeur = v
        self.suivant = None
```

Compléter la classe `FileM` suivante où l'attribut `dernier_file` contient le maillon correspondant à l'élément arrivé en dernier dans la file :

```
class FileM :
    def __init__(self) :
        self.dernier_file = None
    def enfile(self,element) :
        nouveau_maillon = Maillon(...)
        nouveau_maillon.suivant = ...
        self.dernier_file = ...
    def est_vide(self) :
        return self.dernier_file == None
    def affiche(self) :
        maillon = self.dernier_file
        while maillon != ... :
            print(maillon.valeur)
            maillon = ...
    def defile(self) :
        if not self.est_vide() :
            if self.dernier_file.suivant == None :
                resultat = self.dernier_file.valeur
                self.dernier_file = None
                return resultat
            maillon = ...
            while maillon.suivant.suivant != None :
                maillon = maillon.suivant
            resultat = ...
            maillon.suivant = None
            return resultat
         return None
```

On pourra tester le fonctionnement de la classe en utilisant les commandes ci-contre dans la console Python.

```
>>> F = FileM()
>>> F.est_vide()
True
>>> F.enfile(2)
>>> F.affiche()
2
>>> F.est_vide()
False
>>> F.enfile(5)
>>> F.enfile(7)
>>> F.affiche()
7
5
2
>>> F.defile()
2
>>> F.defile()
5
>>> F.affiche()
7
```

Exercice 6. Files bornées en POO

Nous allons implémenter dans cet exercice la classe `FileB` en considérant des files bornées de capacité *n*.

La classe `FileB` a trois attributs :

- elem : une liste initialement constituée de *n* « None ».
- capa : le nombre de places *n* dans la file (capacité).
- nb : le nombre d'éléments dans la file bornée, initialement nul.

```
class FileB:
    def __init__(self, n):
        self.elem = [None] * n
        self.capa = n
        self.nb = 0
```

1. Ajouter la méthode `est_vide(self)` qui renvoie `True` si la file est vide, `False` sinon.
2. Ajouter la méthode `est_pleine(self)` qui renvoie `True` si la file est pleine, `False` sinon.
3. Compléter la classe avec la méthode `enfiler(self, e)` qui ajoute l'élément e à la fin de la file et ne renvoie rien.
4. Ajouter la méthode `defiler(self)` qui retire l'élément en tête de la file si elle n'est pas vide et le renvoie.
5. Terminer avec la méthode `__str__(self)` qui renvoie les éléments de la file, du premier au dernier, séparés par des « < ».

On pourra tester le fonctionnement de la classe en utilisant les commandes ci-contre dans la console Python.

```
>>> FB = FileB(3)
>>> str(FB)
''
>>> FB.est_vide()
True
>>> FB.est_pleine()
False
>>> FB.enfiler('N')
>>> FB.enfiler('S')
>>> FB.enfiler('I')
>>> FB.enfiler('Z')
>>> FB.est_vide()
False
>>> FB.est_pleine()
True
>>> str(FB)
'N < S < I'
>>> FB.defiler()
'I'
>>> str(FB)
'N < S'
>>>
```

Exercice 7. Files circulaires en POO (Facultatif)

Reprendre l'exercice précédent en implémentant la classe `FileC` pour des files à circulaires.

La classe FileC a quatre attributs :

- elem : une liste initialement vide.
- capa : le nombre de places dans la file (capacité).
- iDeb : l'index du premier élément de la file bornée, initialement nul.
- iFin : l'index du dernier élément de la file bornée, initialement nul.