Partie 1

Exercice 1 : Gestion d'une file

On considère une structure de données file que l'on représentera par des éléments en ligne, l'élément à droite étant la tête de la file et l'élément à gauche étant la queue de la file.

On appellera `f1` la file suivante :

'bac'	'nsi'	'2024'	'file'

On suppose que les quatre fonctions suivantes ont été programmées préalablement en langage Python

- `creer_file()` : renvoie une file vide ;
- `est_vide(f)` : renvoie `Truesi` la file `f` est vide et `False` sinon ;
- `enfiler(f, e)` : ajoute l'élément `e` à la queue de la file `f` ;
- `defiler(f)` : renvoie l'élément situé à la tête de la file `f` et le retire de la file.

1. Les trois questions suivantes sont indépendantes.

a. Donner le résultat renvoyé après l'appel de la fonction `est_vide(f1)`

b. Représenter la file `f1` après l'exécution du code `defiler(f1)`

c. Représenter la file `f2` après l'exécution du code suivant :

```
1 f2 = creer_file()
2 liste = ['castor', 'python', 'poule']
3 for elt in liste:
4     enfiler(f2, elt)
```

2. Recopier et compléter les lignes 4, 6, 7 et 9 de la fonction `longueur` qui prend en paramètre une file `f` et qui renvoie le nombre d'éléments qu'elle contient. Après un appel à la fonction, la file `f` doit retrouver son état d'origine.

```
1  def longueur(f):
2      resultat = 0
3      g = creer_file()
4      while ... :
5          elt = defiler(f)
6          resultat = ...
7          enfiler(... , ...)
8      while not(est_vide(g)):
9          enfiler(... , defiler(g))
10     return resultat
```

3. Un site impose à ses clients des critères sur leur mot de passe. Pour cela il utilise la fonction `est_valide` qui prend en paramètre une chaine de caractères `mot` et qui retourne `True` si `mot` correspond aux critères et `False` sinon.

```
def est_valide(mot):
    if len(mot) < 8:
        return False
    for c in mot:
        if c in ['!', '#', '@', ';', ':']:
            return True
        return False
```

Parmi les mots de passe suivants, recopier celui qui sera validé par cette fonction.

A - 'best@'

B - 'paptap23'

C - '2!@59fgds'

4. L'exemple suivant montre l'évolution d'une file `f3` après l'exécution de l'instruction `ajouter_mot(f3, 'super')` :

- état initial de `f3` avant :

'bac'	'nsi'	'2024'

- état de `f3` après `ajouter_mot(f3, 'super')`

'super'	'bac'	'nsi'	'2024'

Écrire le code de cette fonction `ajouter_mot` qui prend en paramètres une file `f` (qui a au plus 3 éléments) et une chaine de caractères valide `mdp`. Cette fonction met à jour la file de stockage `f` des mots de passe en y ajoutant `mdp` et en défilant, si nécessaire, pour avoir au maximum trois éléments dans cette file.

On pourra utiliser la fonction `longueur` de la question 6.

5. Pour intensifier sa sécurité, le site stocke les trois derniers mots de passe dans une file et interdit au client lorsqu'il change son mot de passe d'utiliser l'un des mots de passe stockés dans cette file. Recopier et compléter les lignes 7, 8, 10 et 11 de la fonction `mot_file` :

- qui prend en paramètres une file `f` et `mdp` de type chaine de caractères ;
- qui renvoie `True` si le mot de passe est un élément de la file `f` et `False` sinon. Après un appel à cette fonction, la file f doit retrouver son état d'origine.

```
def mot_file(f, mdp):
    g = creer_file()
    present = False
    while not(est_vide(f)):
        elt = defiler(f)
        enfiler(g, elt)
        if ...:
            present = ...
    while not(est_vide(g)):
        enfiler(..., ...)
    return ...
```

6. Écrire une fonction `modification` qui prend en paramètres une file `f` et une chaine de caractères `nv_mdp`. Si le mot de passe `nv_mdp` répond bien aux deux exigences des questions 4 et 5, alors elle modifie la file des mots de passe stockés et renvoie `True`. Dans le cas contraire, elle renvoie `False`.

On pourra utiliser les fonctions `mot_file`, `est_valide` et `ajouter_mot`

Partie 2

Exercice 2 : Addition binaire

2.1 Partie 1 : programmation fonctionnelle

La fonction suivante, `ajoute1` va ajouter 1 au nombre binaire placé en argument. Cette fonction prend un paramètre, `nombre`, de type `string`. Les bits du nombre binaire sont mis dans la chaine de caractère.

Exemple d'utilisation :

```
>>> ajoute1("11010111")
"11011000"
>>> ajoute1("11111111")
"00000000"
```

Script de la fonction

```
def ajoute1(nombre):
    nombre2 = ""
    retenue = 1
    for c in renverse(...         :
        c = int(c) + retenue
        if c <= ... :
            retenue = ...
            nombre2 = nombre2 + str(c)
        else:
            retenue = ...
            c = ...
            nombre2 = nombre2 + str(c)
    return renverse(...
```

Le script de la fonction `renverse` n'est pas fourni. Celle-ci retourne les caractères en sens inverse :

```
>>> renverse('LEON')
'NOEL'
```

Le principe de la fonction `ajoute1` est le suivant : on parcourt les bits un à un. On ajoute à chaque bit la retenue. Si l'addition binaire donne 2, le bit additionné vaut zero, et on reporte la retenue pour le bit suivant.

1. Compléter le script du programme.
2. Pourquoi faut-il renverser la chaine nombre pour réaliser l'addition binaire ?
3. Pourquoi l'addition binaire sur la chaine "11111111" donne t-elle zero ?
4. Convertir le nombre binaire 11010111 en décimal. Votre reponse doit montrer le calcul réalisé.

2.2 Partie 2 : Programmation orientée objet

On souhaite transformer ce programme en programmation type objet. On utilise une classe appelée `Binaire`, qui aura pour unique attribut `nombre`, de type `str` et pour méthode de classe : `ajoute1`

5. Ecrire le script de la classe `Binaire`, avec les méthodes `__init__` et `ajoute1`
6. Mettre dans `ajoute1` un test d'assertion qui retourne une erreur lorsque le resultats de l'addition binaire donne "00000000". Ajouter un message d'erreur explicatif qui s'affiche dans la sortie courante : assert test, "message"
7. Ecrire les instructions qui instancient l'objet `b1` comme égal à 11010111, puis l'instruction qui ajoute 1 à ce nombre binaire.