Correction MPSI 2

Python

1. Préliminaires

Q1 – Il s'agit de booléens (`bool`), la réponse est donnée plus loin dans le sujet...

Q2 – `A = [True, False, True, True, False, False, False, False, False, False,True]`

Q3 – Une fonction `occupe(file:list, i:int)` qui renvoie `True` lorsque la case d'indice i de la file est occupée par une voiture et `False` sinon.

```
def occupe(file:list, i:int):
    return file[i]
```

Q4 – On compte 2^n files possibles car 2 possibilités pour n cases.

Q5 – Une fonction `egal(file1, file2)` retournant un booléen permettant de savoir si deux files représentées par les listes `file1` et `file2` sont égales.

```
def egal(file1, file2):
    n1, n2 = len(file1), len(file2)
    if n1 != n2:
        return False
    for i in range(n1):
        if file1[i] != file2[i]:
            return False
    return True
```

Q6 – Dans le meilleur des cas le parcours s'arrête à la comparaison de chacun des premiers termes, la complexité est alors en temps constant $\mathcal{O}(1)$.
Dans le pire des cas, il faut parcourir toute la liste, la complexité est linéaire $\mathcal{O}(n)$.

Q7 – Une fonction `copie(file:list)` qui renvoie une copie de la liste donnée en argument.

Python 1 Par compréhension

```
def copie(file):
    return [voiture for voiture in file]
```

Python 2 Avec append

```
def copie(file):
    new = []
    for voiture in file:
        new.append(voiture)
    return new
```

Q8 – Une fonction `comptage(file)` renvoyant le nombre voitures dans la file.

```
def comptage(file):
    compteur = 0
    for voiture in file:
        if voiture:
            compteur += 1
    return compteur
```

La complexité de ce code est linéaire.

2. Déplacement de voitures dans la file

Q9 – Une fonction `avancer(file:list, gauche:bool)` prenant en paramètres une liste de départ codant la file de voitures, un booléen indiquant si la case la plus à gauche doit devenir occupée lors de l'étape de simulation, et renvoyant la liste obtenue par une étape de simulation.

```
def avancer(file:list, gauche:bool):
    n = len(file)
    for i in range(n-1, 0, -1):
        file[i] = file[i-1]
    file[0] = gauche
    return file
```

Q10 – L'instruction suivante : `avancer(avancer(A, False), True)` renvoie :
`[True, False, True, False, True, True, False, False, False, False, False]`

Q11 – Une fonction `avancer_fin(file:list, indice:int)` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier la file donnée en argument.

```
def avancer_fin(file:list, m):
    new = copie(file)
    n = len(file)
    for i in range(n-1, m, -1):
        new[i] = new[i-1]
    new[m] = False # dans tous les cas, la case sera vide
    return new
```

Q12 – Une fonction `avancer_debut(file:list, gauche:bool, indice:int)` :

```
def avancer_debut(file:list, gauche:bool, m:int):
    new = copie(file)  # case m libre
    n = len(file)
    for i in range(m, 0, -1):
        new[i] = new[i-1]
    new[0] = gauche
    return new
```

Q13 – Une fonction `avancer_debut_bloque(file, gauche, indice)` :

```
def avancer_debut_bloque(file, gauche, m):
    new = copie(file)
    n = len(file)
    for i in range(m-2, -1, -1):
```

```
        if not occupe(new,i+1):
            new[i+1] = new[i] # on déplace
            new[i] = False # on supprime
    new[0] = gauche or new[0] # si true pas remplacé
    return new
```

3. Simulation à deux files

Q14 – Une fonction `avancer_files(file1, gauche1:bool, file2, haut2:bool)` qui renvoie le résultat d'une étape de simulation.

```
def avancer_files(file1, gauche1, file2, haut2):
    m = len(file1) // 2
    result2 = avancer_fin(file2, m)
    result1 = avancer(file1, gauche1) # pas de contrainte
    if occupe(result1, m):
        result2 = avancer_debut_bloque(result2, haut2, m)
    else:
        result2 = avancer_debut(result2, haut2, m)
    return [result1, result2]
```

Q15 – On considère les listes `file1 = [False, True, False, True, False]` et `file2 = [False, True, True, False, False]`. L'appel `avancer_files(file1, False, file2, False)` renvoie :
`[[False, False, True, False, True], [False, True, False, True, False]]`

Q16 – On importe la fonction `random` du module `random` selon :

```
from random import random
```

Q17 – Une fonction `voiture()` qui renvoie `True` ou `False` de façon équiprobable.

```
def voiture():
    return random() < .5
```

Q18 – Une fonction `simulation(file1, file2, duree:int)` qui renvoie le résultat d'une simulation après un nombre `duree` d'étape.

```
def simulation(file1, file2, duree):
    for _ in range(duree):
        file1, file2 = avancer_files(file1, voiture(), file2,
            voiture())
    return [file1, file2]
```

Q19 – Une fonction `couleur(file:list, color:str)` renvoyant la liste des couleurs pour une file donnée.

```
def couleur(file , color):
    list_color = []
    for voiture in file:
        if voiture:
            list_color.append(color)
        else:
            list_color.append('w')
    return list_color
```

ou en version *One-liner*

```
def couleur(file, color):
    return [color if voiture else 'w' for voiture in file]
```