— *Les indentations seront tracées à la règle. Tout doute sur cette indentation sera notée en votre défaveur.*
— *Les fonctions* ***max***, ***min***, ***index***, ***sum*** *et la méthode* ***sort*** *ne seront pas utilisées. La méthode* ***append*** *est autorisée.*
— *Pour répondre à une question donnée de ce sujet, on pourra utiliser librement les fonctions écrites dans les questions précédentes, même si on n'a pas réussi à programmer l'une de ces fonctions.*

# 1 Des algorithmes classiques

**Q1.** Coder une fonction **maxi** d'argument une liste **L** de flottants, qui renvoie la valeur du maximum de **L**.

**Q2.** Coder une fonction **maxi_et_rang** d'argument une liste **L** de flottants, qui renvoie la valeur et le rang du maximum de **L**. Si la valeur maximale apparaît plusieurs fois dans la liste, c'est le rang de la dernière rencontrée dans la liste lue de gauche à droite.

**Q3.** Coder une fonction **somme** d'argument une liste **L** qui renvoie la somme des valeurs de cette liste.

**Q4.** On rappelle un code pour trier en place par ordre croissant la liste **L** de flottants selon le principe du tri à bulle :

```
def tri_croissant (L):
    n = len(L)
    for i in range(n-1):
        for j in range(n-i-1):
            if L[j]>L[j+1]:
                L[j],L[j+1] =L[j+1],L[j]
```

Coder une fonction **tri_decroissant** d'argument une liste **L** qui trie en place par ordre décroissant la liste **L** selon le principe du tri à bulle.

# 2 Étude de trafic routier

Ce sujet concerne la conception d'un logiciel d'étude de trafic routier. On modélise le déplacement d'un ensemble de voitures sur des files à sens unique (voir Figures 1 et 2). C'est un schéma simple qui peut permettre de comprendre l'apparition d'embouteillages et de concevoir des solutions pour fluidifier le trafic.

**Notations** : soit $L$ une liste,
— on note $len(L)$ sa longueur ;
— pour $i$ entier, $0 \leqslant i < len(L)$, l'élément de la liste d'indice $i$ est noté $L[i]$ ;
— pour $i$ et $j$ entiers, $0 \leqslant i < j \leqslant len(L)$, $L[i:j]$ est la sous-liste composée des éléments $L[i], ..., L[j-1]$;
— $p * L$, avec $p$ entier, est la liste obtenue en concaténant $p$ copies de $L$. Par exemple, $3 * [0]$ est la liste $[0, 0, 0]$.

**Exemples** :

FIGURE 1 – Représentation d'une file de longueur 11 comprenant 4 voitures, Les voitures sont représentées par une flèche. situées respectivement sur les cases d'indices 0, 2, 3 et 10.

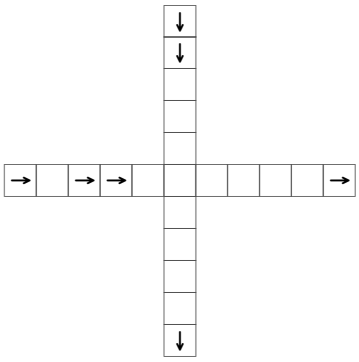


FIGURE 2 – Configuration représentant deux files de circulation à sens unique se croisant en une case.

## 2.1 Préliminaires

Dans un premier temps, on considère le cas d'une seule file, illustré par la Figure 1. Une file de longueur $n$ est représentée par $n$ cases. Une case peut contenir au plus une voiture. Les voitures présentes dans une file circulent toutes dans la même direction (sens des indices croissants, désigné par les flèches sur la Figure 1) et sont indifférenciées.

**Q5.** On se propose de représenter une file de voitures à l'aide d'une liste de booléens. Ainsi, la liste représentant la file de la Figure 1 est [**True, False, True, True, False, False, False, False, False, False, True**]. Donner la liste qui représente la file verticale de la Figure 2

**Q6.** Soit $L$ une liste représentant une file de longueur $n$ et $i$ un entier tel que $0 \leqslant i < n$. Coder une fonction **occupe** d'arguments une liste de booléens **L** et un entier **i** qui renvoie **True** lorsque la case d'indice $i$ de la file est occupée par une voiture et **False** sinon.

**Q7.** Coder une fonction **egales** d'arguments deux listes **L1** et **L2** retournant un booléen permettant de savoir si ces deux listes ont égales (et alors tous leurs éléments de même rang sont égaux). On n'utilisera pas L1==L2 !

## 2.2 Déplacement de voitures dans la file

On identifie désormais une file de voitures à une liste. On considère les schémas de la Figure 3 représentant des exemples de files. Une étape de simulation pour une file consiste à déplacer les voitures de la file, à tour de rôle, en commençant par la voiture la plus à droite, d'après les règles suivantes :

— une voiture se trouvant sur la case la plus à droite de la file sort de la file ;

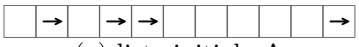
— une voiture peut avancer d'une case vers la droite si elle arrive sur une case inoccupée ;

— une case libérée par une voiture devient inoccupée ;

— la case la plus à gauche peut devenir occupée ou non, selon le cas considéré.

On suppose avoir codé la fonction **avancer** d'arguments une liste de départ, 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.

Par exemple, l'application de cette fonction à la liste illustrée par la Figure 3(a) permet d'obtenir soit la liste illustrée par la Figure 3(b) lorsque l'on considère qu'aucune voiture nouvelle n'est introduite, soit la liste illustrée par la Figure 3(c) lorsque l'on considère qu'une voiture nouvelle est introduite.



(a) liste initiale **A**

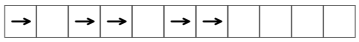
(b) **B=avancer(A, False)**

(c) **C=avancer(A, True)**

FIGURE 3 – Étape de simulation

**Q8.** Étant donnée $A$ la liste définie à la question 5, que renvoie **avancer(avancer(A, False),True)** ?

**Q9.** On considère $L$ une liste et $m$ l'indice d'une case de cette liste ($0 \leqslant m < len(L)$). On s'intéresse à une étape partielle où seules les voitures situées sur la case d'indice $m$ ou à droite de cette case peuvent avancer normalement, les autres voitures ne se déplaçant pas.

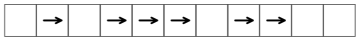
Exemple : pour $m = 5$, la file 

devient 

Coder une fonction **avancer_fin** d'arguments une liste de booléens **L** et un entier **m** qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier **L**.

**Q10.** Soient **L** une liste, **b** un booléen et **m** l'indice d'une case inoccupée de cette liste. On considère une étape partielle où seules les voitures situées à gauche de la case d'indice **m** se déplacent, les autres voitures ne se déplacent pas. Le booléen **b** indique si une nouvelle voiture est introduite sur la case la plus à gauche.

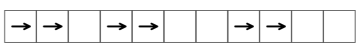
Exemple : pour $m = 5$, la file 

devient  lorsqu'aucune nouvelle voiture n'est introduite.

Coder une fonction **avancer_fin** d'arguments une liste de booléens **L**, un booléen **b** et un entier **m** qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier **L**.

**Q11.** On considère une liste **L** dont la case d'indice **m** strictement positif est temporairement inaccessible et bloque l'avancée des voitures. Une voiture située immédiatement à gauche de la case d'indice **m** ne peut pas avancer. Les voitures situées sur les cases plus à gauche peuvent avancer, à moins d'être bloquées par une case occupée, les autres voitures ne se déplacent pas. Un booléen **b** indique si une nouvelle voiture est introduite lorsque cela est possible.

Exemple : pour $m = 5$, la file 

devient  lorsqu'aucune nouvelle voiture n'est introduite.

Coder une fonction **avancer_debut_bloque** d'arguments une liste de booléens **L**, un booléen **b** et un entier **m** qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste.

## 2.3 Une étape de simulation à deux files

On considère dorénavant deux files $L1$ et $L2$ de même longueur impaire se croisant en leur milieu ; on note $m$ l'indice de la case du milieu. La file $L1$ est toujours prioritaire sur la file $L2$. Les voitures ne peuvent pas quitter leur file et la case de croisement ne peut être occupée que par une seule voiture. Les voitures de la file $L2$ ne peuvent accéder au croisement que si une voiture de la file $L1$ ne s'apprête pas à y accéder. Une étape de simulation à deux files se déroule en deux temps :

— Dans un premier temps, on déplace toutes les voitures situées sur le croisement ou après.

— Dans un second temps, les voitures situées avant le croisement sont déplacées en respectant la priorité.

Exemple : partant d'une configuration donnée par la Figure 4(a)), les configurations successives sont données par les Figures 4(b), 4(c), 4(d), 4(e) et 4(f) en considérant qu'aucune nouvelle voiture n'est introduite.

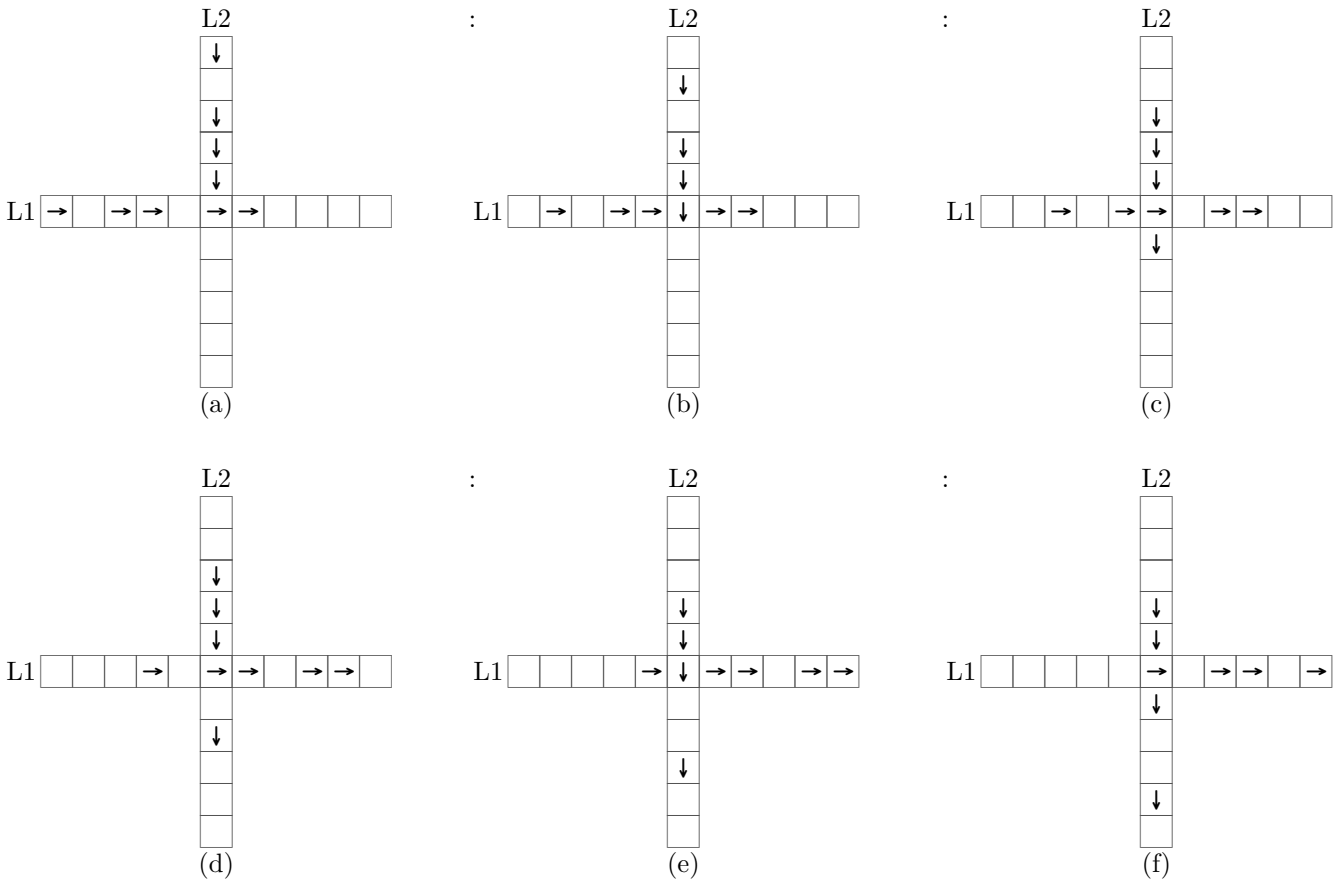



FIGURE 4 – Étapes de simulation à deux files

**Q12.** Coder une fonction **avancer_files(L1, b1, L2, b2)** qui renvoie le résultat d'une étape de simulation sous la forme d'une liste de deux éléments notée **[R1, R2]**, sans changer les listes **L1** et **L2**. Les booléens **b1** et **b2** indiquent respectivement si une nouvelle voiture est introduite dans les files **L1** et **L2**. Les listes **R1** et **R2** correspondent aux listes après déplacement.

**Q13.** On considère les listes **D = [False, True, False, True, False]**, **E = [False, True, True, False, False]**. Que renvoie l'appel **avancer_files(D, False, E, False)** ?