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

```
def maxi(L):
    m=L[0]
    for k in range(1,len(L)):
        if L[k]>m:
            m=L[k]
    return m
```

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**.

```
def maxi_et_rang(L):
    m,r=L[0],0
    for k in range(1,len(L)):
        if L[k]>m:
            m,r=L[k],k
        elif L[k]==m:
            r=k
    return m,r
```

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

```
def somme(L):
    s =0
    for element in L:
        s+=element
    return s
```

Q4. Coder une fonction **tri_decroissant** d'argument une liste **L** qui trie par ordre décroissant la liste **L** selon le principe du tri à bulle.

```
def tri_decroissant (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]
```

Q5. Donner la liste qui représente la file verticale de la Figure **??**

```
[True]*2+[False]*7+[True]
```

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.

```
def occupe(L,i):
    return L[i]
```

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 !

```python
def egales(L1,L2):
    for k in range(len(L1)):
        if L1[k]!=L2[k]:
            return False
    return True
```

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

```python
[True, False, True, False, True, True, False, False, False, False, False]
```

Q9. 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**.

```python
avancer_fin =lambda L, m : L[:m] +
avancer (L[m:], False)
```

```python
def avancer_fin(L,m):
    nouvelleL=L[:]
    for k in range(len(L)-1,m,-1):
        nouvelleL[k]=nouvelleL[k-1]
    nouvelleL[m]=False
    return nouvelleL
```

```python
def avancer_fin (L, m):
    if m ==len (L) - 1:
        nouvelleL= L [:]
        nouvelleL [m] =False
        return nouvelleL
    else :
        nouvelleL =avancer_fin (L, m +1)
        nouvelleL [m+1],nouvelleL[m] =
nouvelleL [m], False
        return nouvelleL
```

Q10. Coder une fonction **avancer_debut** 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**.

```python
avancer_debut=lambda L, b, m :avancer(L[:m +
1], b)+L[m +1:]
```

```python
def avancer_debut(L,m,b):
    nouvelleL=L[:]
    for k in range(m,0,-1):
        nouvelleL[k]=nouvelleL[k-1]
    nouvelleL[0]=b
    return nouvelleL
```

```python
def avancer_debut (L, b, m):
    if m ==1:
        nouvelleL=L[:]
        nouvelleL [1] =nouvelleL [0]
        nouvelleL [0] =b
        return nouvelleL
    else :
        nouvelleL [m] =nouvelleL [m - 1]
        nouvelleL =avancer_debut (L, b, m - 1)
        return nouvelleL
```

Q11. 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.

```python
def avancer_debut_bloque(L,m,b):
    nouvelleL=L[:]
    for k in range(m-2,-1,-1):
        if nouvelleL[k]==False :
            return avancer_debut(nouvelleL,k,b)
    return nouvelleL
```

Q12. On considère les listes **D = [False, True, False, True, False]**, **E = [False, True, True, False, False]**. Que doit renvoyer l'appel **avancer_files(D, False, E, False)** ? Vous pouvez dessiner sur votre copie !

```
[[False, False, True, False, True], [False, True, False, True, False]]
```

Q13. Coder la fonction **avancer_files(L1, b1, L2, b2)**

```
def avancer_files(L1,b1,L2,b2):
    R1,R2=L1[:],L2[:]
    m=len(L1)//2
    R1=avancer(R1,b1)
    R2=avancer_fin(R2,m)
    if R1[m]==True:
        R2=avancer_debut_bloque(R2,m,b2)
    else:
        R2=avancer_debut(R2,m,b2)
    return [R1,R2]
```