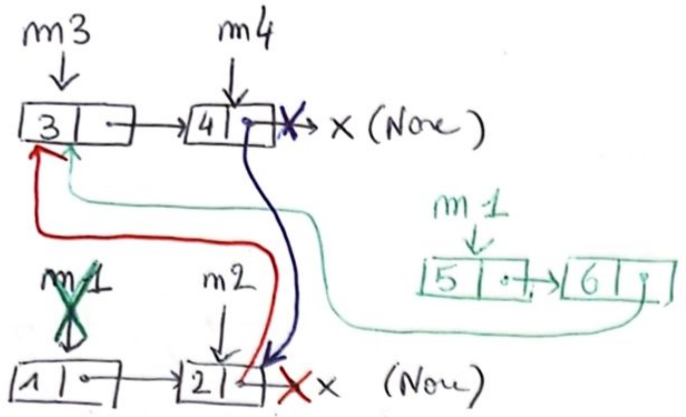
Exercice 1.

1a.



Le code affiche :

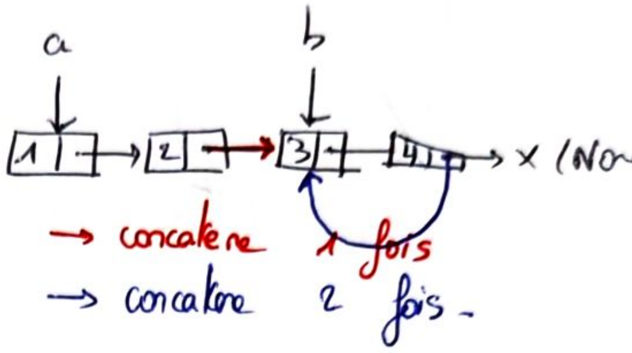
```
1 - 2 - x
2 - x
3 - 4 - x
4 - x .
```

b
```
>>> affiche(m1)
    5 - 6 - 3 - 4 - x
>>> affiche(m2)
    2 - x
```

c.
```
>>> affiche(m1)
    5 - 6 - 3 - 4 - x
>>> affiche(m2)
    2 - 3 - 4 - x
```

d! En bleu sur le schéma.
```
>>> affiche(m1)
    5 - 6 - 3 - 4 - 2 - 3 - 4 - 2 - ...
```
L'exécution ne termine pas.   continue d'afficher 3-4-2-...

2. a)
```
def concatene(m1, m2)
    m_c = m1
    while m_c.suivant is not None :
        m_c = mc.suivant
    m_c.suivant = m2.
```



b) i)
```
>>> affiche(a)
1 - 2 - 3 - 4 - x
```
ii)
```
>>> affiche(a)
1 - 2 - 3 - 4 - 3 - 4 - 3 - 4 - ....
```
iii) L'exécution de concatene (a, b) ne termine pas.

Exercice 2.

d((1, 2, 3)) ↗ (1, 1, 2, 2, 3, 3)

↓ reste = (2, 2, 3, 3)

new = (1, 1, 2, 2, 3, 3) (2, 2, 3, 3)

d((2, 3))

↓ reste = (3, 3)

new = (2, 2, 3, 3) (3, 3)

d((3))

↓ reste = ( )

new = (3, 3)

d(( )) ↗ ( )

Donc
```
>>> affiche(d(l))
1 - 1 - 2 - 2 - 3 - 3 - x
```

2.

$$\begin{tikzcd}
f(\ (1,2,3),\ \ (4,5,6)\ )\quad \text{inter} = (2,5,3,6) \arrow[d] \arrow[r] & (1,4,2,5,3,6) \\
f(\ (2,3),\ \ (5,6)\ )\quad \text{inter} = (3,6) \arrow[d] \arrow[u, bend right, "(2,5,3,6)"'] & \\
f(\ (3),\ \ (6)\ )\quad \text{inter} = (\ ) \arrow[d] \arrow[u, bend right, "(3,6)"'] & \\
f(\ (\ ),\ \ (\ )\ ) \arrow[u, bend right, "(\ )"'] &
\end{tikzcd}$$

Donc >>> affiche(f(l1 , l2))

1 - 4 - 2 - 5 - 3 - 6 - x

Exercice 3. Partie A.

1. a. i) Renvoie (3)

ii) Renvoie (3, 4, 5, 6)

iii) Renvoie (4, 11, 14, 18)

iv) Renvoie (19, 20, 22, 23)

v) Renvoie (5, 12, 16, 42)

b.
```
def inserer_dans_liste_triee (l, e)
    if est_vide (l):
        return singleton(e)
    elif e <= tete(l):
        return ajoute( l, e )
    else:
        return ajoute( inserer_dans_liste_triee (queue (l),e),
                       tete (l))
```

2. a. (8, 1, 9, 7, 3)

→ $\underbrace{8}_{\text{tête}}$ $\underbrace{(1, 9, 7, 3)}_{\text{queue}}$ $\underbrace{(1, 3, 7, 9)}_{\text{queue triée}}$

→ on insère 8 dans la queue triée.

(1, 3, 7, 8, 9)

b.
```
def tri_insertion(l)
    if est_vide(l):
        return l
    else:
        trie_intermediaire = tri_insertion(queue(l))
        affiche(trie_intermediaire)
        resultat = inserer_dans_liste_triee(tri_intermediaire,
                                             tete(l))
        print(f"On y insere {tete(l)}")
        return resultat.
```

c.
```
>>> tri_insertion(l)
x
On y insere 1
1 - x
On y insere 9
1 - 9 - x
On y insere 8
1 - 8 - 9 - x
```

$$\begin{tikzcd}
 & \\
\text{t\_i((8,9,1))} \arrow[d] \arrow[u, "{(1,8,9)}"'] & \text{p10 p16} \\
\text{t\_i((9,1))} \arrow[d] \arrow[u, "{(1,9)}"'] & \text{p10 p16} \\
\text{t\_i((1))} \arrow[d] \arrow[u, "{(1)}"'] & \text{p10 p16} \\
\text{t\_i(())} \arrow[u, "{()}"'] &
\end{tikzcd}$$

Partie B.

1.
```
1) def tri_selection(l)
2)     if est_vide(l):
3)         return l
4)     else:
5)         mini = minimum(l)
6)         l_sans_mini = supprime(l, m)
7)         inter = tri_selection(l_sans_mini)
8)         return ajoute(inter, mini)
```

2. tri_selection est une fonction récursive car elle effectue un appel récursif ligne 7.