Proposition de correction

Exercice 1

Q1.a

```
class Concurrent :
    def __init__(self, pseudo, temps, penalite) :
            self.pseudo = pseudo
            self.temps = temps
            self.penalite = penalite
            self.temps_tot = temps + penalite
```

Q1.b

99.67

Q1.c

c1.temps_tot

Q2.a

```
resultats.queue().queue().tete()
```

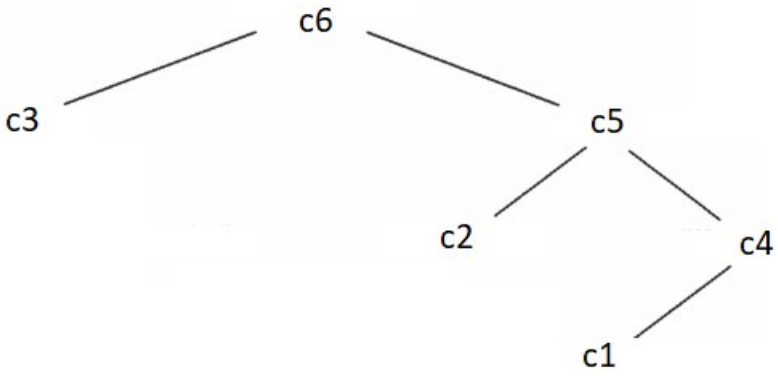
Q2.b

```
resultats.tete().temps_tot
```

Q3

```
def meilleur_concurrent(L) :
    conc_mini = L.tete()
    mini = conc_mini.temps.tot
    Q = L.queue()
    while not Q.est_vide(Q) :
            elt = Q.tete()
            if elt.temps_tot < mini :
                    conc_mini = elt
                    mini = elt.temps_tot
            Q = Q.queue()
    return conc_mini
```

Q4



Exercice 2

Q1.a

Proposition 2

Q1.b

cd lycee

Q1.c

mkdir algorithmique

Q1.d

rm image1.jpg

Q2.a

927

Q2.b

927

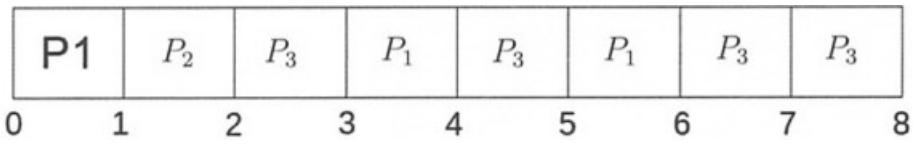
Q2.c

927 et 1058

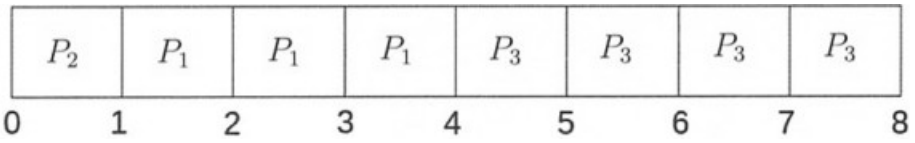
Q2.d

923 et 1036

Q3.a

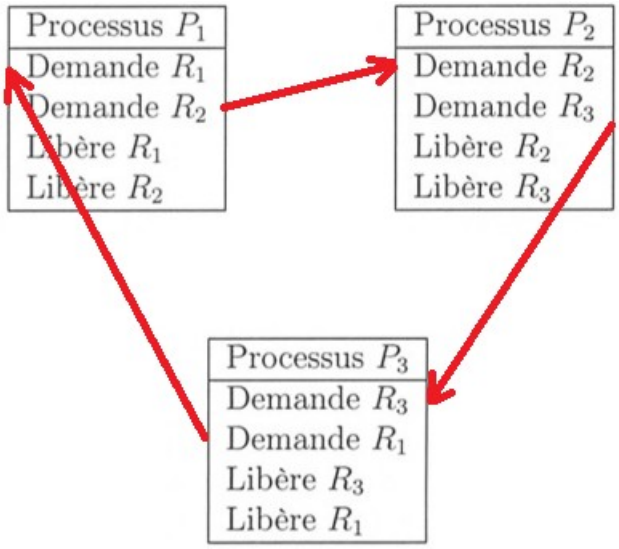


Q3.b



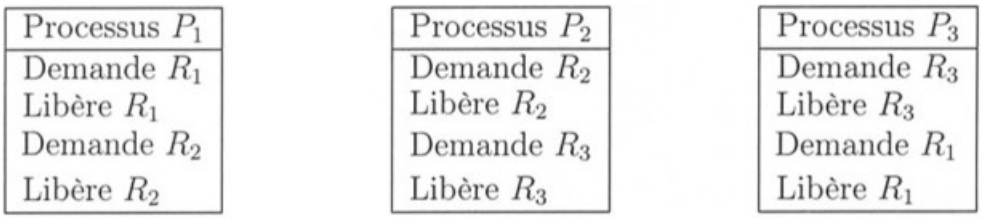
Q4.a

1. en attente (prêt)
2. bloqué (endormi)
3. actif (élu)
4. mort (terminé)



Les processus attendent des ressources qui ne seront jamais libérées

Q4.b



Exercice 3

Q1.a

identifie de manière unique un enregistrement dans une table (intégrité relationnelle).

Q1.b

établit des relations entre les tables (intégrité référentielle).

Q1.c

Il n'y a pas de contrainte d'unicité sur le n-uplet (idAbonné, idSéance) de la table Réservation.

Q1.d

idAbonné	idSéance	nbPlaces_plein	nbPlaces-réduit
13	737	3	2

Q2.a

```
SELECT titre, réalisateur
FROM Film
WHERE durée < 120 ;
```

Q2.b

Affiche le nombre total de séances diffusées les 22 et 23 octobre 2021

Q3.a

```
SELECT nom, prénom
FROM Abonné
ORDER BY nom, prénom
```

Q3.b

```
SELECT Film.titre, Film.durée
FROM Film, Séance
WHERE Séance.date = '2021-10-12' AND Séance.heure = '21:00'
AND Séance.idFilm = Film.idFilm
ORDER BY Film.titre
```

Q4.a

```
UPDATE Film
```

SET durée = 120

WHERE titre = 'Jungle Cruise'

Q4.b

Pb de contrainte d'intégrité référentielle

Aucune réservation ne doit avoir été effectuées au préalable (ou configurer la suppression en cascade)

Q4.c

DELETE FROM Séance

WHERE idSéance = 135

Exercice 4

Q1.a

Milka

Q1.b

{ Nemo, Moka, Maya, Museau, Noisette }

Q1.c

femelle

Q1.d

Père : Ulk

Mère : Maya

Q2.a

```
def present(arb, nom) :
    if est-vide(arb) :
            return False
    elif racine(arb) == nom :
            return True
    else
            return present(gauche(arb), nom) or present(droit(arb), nom)
```

Q2.b

```
def parents(arbre : object) -> tuple :
    if est_vide(arbre) :
            return None
```

```
    return racine(gauche(arbre)), racine(droit(arbre))
```

Q3.a

- Mango et Cacao ont le même père
- Milka et Cacao ont la même mère

Q3.b

```
def frere_soeur(arbre1 : object, arbre2 : object) -> bool :
    return racine(droit(arbre1)) == racine(droit(arbre2)) or \
           racine(gauche(arbre1)) == racine(gauche(arbre2))
```

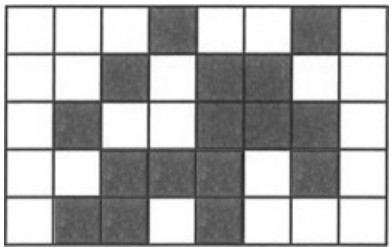
Q4

```
def nombre_chiens(arb : object, n : int) -> int :
    if n == 1 :
        return int(parents(arb) != None) + int(parents(arb) != None)
    else :
        return nombre_chien(gauche(arb), n – 1) + nombre_chiens(droit(arb), n – 1)
```

Exercice 5

Partie A

Q1.a



Q1.b

- Ligne 3, colonne 0
- Ligne 3, colonne 2

Q1.c

- li-1, co-1
- li-1, co+1

Q2.a

```
image[li-1][co-1] != image[li-1][co+1]
```

Q2.b

```
def remplir_ligne(image, li) :
    image[li][0] = 0
    image[li][7] = 0
    for i in range(1, 7) :
            image[li][i] = int(image[li-1][i-1] != image[li-1][i+1])
```

Q2.c

```
def remplir(image : object) :
    for li in range(1, 5) :
            remplir_ligne(image, li)
```

Partie B

Q1.a

32 + 8 + 4 = 44

Q1.b

```
def conversion2_10(tab : list) -> int :
    max = 2 ** (len(tab) – 1)          # à priori sur 8 bits
    dec = 0
    for b in tab :
            if b in [0, 1] :           # vérification binaire
                    if b :
                            dec += max
            max = max // 2
    return dec
```

Q1.c

78 = 64 + 8 + 4 + 2 → [0,1, 0, 0, 1, 1, 1, 0]

Q2.a

n < 128 ET n%2 = 0 car bits de poids fort et de poids faible = 0

Q2.b

```
def generer(n , k) :
    tab     = [None for i in range(k)]
    image = [[0 for j in range(8)] for i in range(k+1)]

    image[0] = conversion10_2(n)
    tab[0]     = conversion2_10(image[0])
    for li in range(1, k) :
            remplir_ligne(image, li)
```

```
            tab[li] = conversion2_10(image[li])

    return tab
```