Exercice 1 :

1. a.

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

1. b. Le `temps_tot` de `c1` est égal à 99,67 (87,67 + 12).

1. c. L'instruction permettant d'accéder au temps total de `c1` est : `c1.temps_tot`

2. a. Instructions pour accéder à c4 :

```
L1 = resultats.queue()
L2 = L1.queue()
c4 = L2.tete()
```

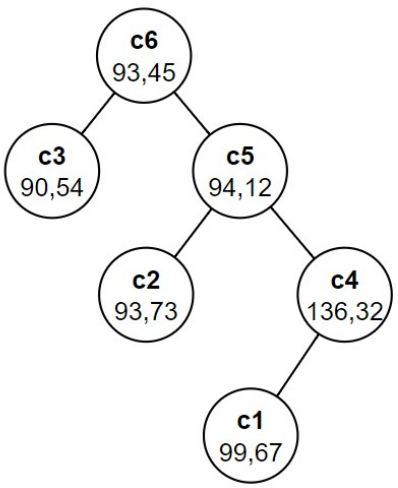
3. Instruction pour obtenir le temps total du concurrent stocké en tête :

```
temps_total = resultats.tete().temps_tot
```

4.

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

5.



CORRECTION – 2022 Amérique du Nord Jour 2

Exercice 2 :

1. a. Le commande `ls` affiche la proposition 2 : lycee perso.

1. b. Commande pour atteindre le répertoire `lycee` : `cd lycee`

1. c. Commande pour créer un répertoire nommé `algorithme` : `mkdir algorithme`

1. d. Commande pour supprimer le fichier `image1.jpg` : `rm image1.jpg`

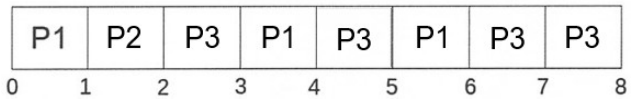
2. a. Le PID du parent du processus démarré par la commande `vi` est 927.

2. b. Les deux PID des processus enfants du processus démarré par la commande `xfce4-terminal` sont 927 et 1058.

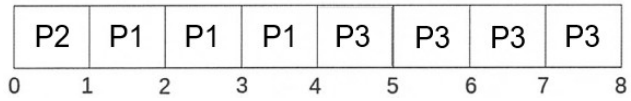
2. c. Les processus 927 et 1058 ont le même parent (PPID 923).
Ou bien encore 900 et 913 (PPID 739), ou 918 et 919 (PPID 823) ou bien 1153 et 1154 (PPID 927) ou bien encore 1132, 1134 et 1149 (PPID 2).

2. d. Les deux processus ayant consommé le plus de temps du processus ont les PID 923 et 1036.

3. a.



3. b.



4. a. Un processus peut être :
- Prêt : En attente d'un accès au processeur .
- Bloqué : En attente d'une ressource.
- Élu : En cours d'exécution par le processeur.

L'interblocage est un situation où des processus sont bloqués car ils attendent une ressource réservée par l'autre pour poursuivre leur exécution. Ici, si la politique d'ordonnancement est celle du « tourniquet » :
- P1 réserve R1
- P2 réserve R2
- P3 réserve R3
- P1 est bloqué car il demande R2 (réservée par P2)
- P2 est bloqué car il demande R3 (réservée par P3)
- P3 est bloqué car il demande R1 (réservée par P1)

on arrive à un interblocage où les trois processus sont tous bloqués.

4. b. Avec une politique d'ordonnancement du « premier arrivé, premier servi » chacun des processus s'exécutent entièrement l'un après l'autre et ne bloquent pas de ressources pour les suivants.

Exercice 3 :

1. a. Une clé primaire permet d'implémenter la contrainte d'intégrité de relation qui stipule que chaque enregistrement dans une table doit pouvoir être identifier de manière unique.

1. b. Une clé étrangère est un attribut qui permet de faire référence à un autre enregistrement via sa clé primaire. Elle implémente la contrainte d'intégrité de référence qui empêche de faire référence à un enregistrement inexistant.

1. c. Un abonné ne peut pas réserver plusieurs fois pour la même séance car ce qui sert de clé primaire à la relation (table) **Réservation** est le couple **idAbonné**, **idSéance** qui doit donc être unique dans cette table. Il est ainsi impossible d'avoir deux fois le même **idAbonné** pour le même **idSéance**.

1. d.

idAbonné	idSéance	nbPlaces_plein	nbPlaces_réduit
13	737	3	2

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

2. b. Cette requête compte le nombre de séances proposées les 22 et 23 octobre 2021.

3. a.
```
SELECT nom, prénom
FROM Abonné ;
```

3. b.
```
SELECT f.titre, f.durée FROM Film AS f
JOIN Séance AS s ON f.idFilm = s.idFilm
WHERE s.date = '2021-10-12' AND s.heure = '21:00' ;
```

4. a.
```
UPDATE Film SET durée = 127
WHERE titre = 'Jungle Cruise' ;
```

Attention, ici nous ne connaissons pas l'idFilm du film « Jungle Cruise » qui aurait permis de l'identifier de manière unique. La requête modifiera la durée de tous les films ayant le même titre.

4. b. Cette requête de suppression de séance peut violer la contrainte d'intégrité de référence si des réservations (qui font référence à cette séance) ont déjà été effectuées pour cette séance.

4. c.
```
DELETE FROM Séance
WHERE diSéance = 135 ;
```

Exercice 4 :

1. a. La racine de l'arbre est Milka.

1. b. Les feuilles sont Nemo, Moka, Maya, Museau et Noisette.

1. c. Nuage est une femelle car c'est la mère (sous-arbre droit) de Nougat.

1. d. Le père et la mère d'Etoile sont respectivement Ulk et Maya.

2. a.

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

2. b.

```
def parents(arb) :
    if est_vide(gauche(arb)):
        pere = ""
    else :
        pere = racine(gauche(arb))
    if est_vide(droit(arb)):
        mere = ""
    else :
        mere = racine(droit(arb))
    return (pere, mere)
```

3. a. Mango et Cacao sont frères par leur père et Cacao et Milka sont frère et sœur par leur mère.

3. b.

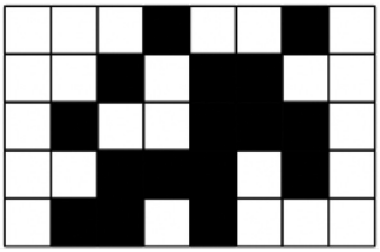
```
def frere_soeur(arbre1, arbre2):
    parents1 = parents(arbre1)
    parents2 = parents(arbre2)
    return parents1[0]==parents2[0] or parents1[1]==parents2[1]
```

4.

```
def nombre_chiens(arb, n):
    if est_vide(arb):
        return 0
    elif n == 0:
        return 1
    else:
        return nombre_chiens(gauche(arb), n-1) + nombre_chiens(droit(arb), n-1)
```

Exercice 5 - Partie A :

1. a.



1. b. Le pixel situé juste au dessus à gauche est à la ligne 3 et à la colonne 0. Celui situé juste au dessus à droite est à la ligne 3 et à la colonne 2.

1. c. Le pixel situé juste au dessus à gauche est à la ligne **`li-1`** et à la colonne **`co-1`**. Celui situé juste au dessus à droite est à la ligne **`li-1`** et à la colonne **`co+1`**.

2. a. Pour que **`image[li][co]`** prenne la valeur 1 il faut que les valeurs de **`image[li-1][co-1]`** et **`image[li-1][co+1]`** soient différentes.

2. b.

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

2. c.

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

Exercice 5 - Partie B :

1. a. $00101100_{(2)} = 2^5 + 2^3 + 2^2 = 32 + 8 + 4 = 44_{(10)}$

1. b.

```
def conversion2_10(tab):
    nb_bit = len(tab)
    somme = 0
    for i in range(nb_bit):
        somme += tab[i]*2**(nb_bit-1-i)
    return somme
```

1.c. $78_{(10)} = 64 + 8 + 4 + 2 = 2^6 + 2^3 + 2^2 + 2^1 = 0100\ 1110_{(2)}$
Le tableau correspondant à l'entier 78 est donc : `[0, 1, 0, 0, 1, 1, 1, 0]`

2. a. Les pixels à gauche et à droite sont forcément blancs donc la plus grande valeur représentable est $0111\ 1110_{(2)} = 126_{(10)}$. Le premier bit à droite est toujours nul donc la valeur représentée est obligatoirement paire.
En conclusion : **n** est forcément un entier pair inférieur ou égal à 126.

2. 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)  # Écriture de la première ligne
    remplir(image)                # Remplissage des autres lignes
    for li in range(1, k+1):
        tab[li-1] = conversion2_10(image[li])
    return tab
```