Amérique du nord - mai 2022 - sujet 2

Exercice 1 (Arbres binaires de recherche et POO - 4 points)
Lors d'une compétition de kayak, chaque concurrent doit descendre le même cours d'eau en passant dans des portes en un minimum de temps. Si le concurrent touche une porte, il se voit octroyé une pénalité en secondes. Son résultat final est le temps qu'il a mis pour descendre le cours d'eau auquel est ajouté l'ensemble des pénalités qu'il a subies.
Un gestionnaire de course de kayak développe un programme Python pour gérer les résultats lors d'une compétition.
Dans ce programme, pour modéliser les concurrents et leurs résultats, une classe `Concurrent` est définie avec les attributs suivants :

- `nom` de type `str` qui représente le pseudonyme du compétiteur ;
- `temps` de type `float` qui est le temps mis pour réaliser le parcours en secondes ;
- `penalite` de type `int` qui est le nombre de secondes de pénalité cumulées octroyées au concurrent ;
- `temps_tot` de type `float` qui correspond au temps total, c'est-à-dire au temps mis pour réaliser le parcours auquel on a ajouté les pénalités.

Dans cet exercice, on suppose que tous les concurrents ont des temps différents.

Le code Python incomplet de la classe `Concurrent` est donné ci-dessous :

```
1 class Concurrent:
2     def __init__(self, pseudo, temps, penalite):
3         self.nom = pseudo
4         self.temps = temps
5         self.penalite = ...
6         self.temps_tot = ...
```

1. (a) Recopier et compléter le code du constructeur de la classe `Concurrent`.
   On exécute l'instruction suivante : `c1 = Concurrent("Mosquito", 87.67, 12)`
   (b) Donner la valeur de l'attribut `temps_tot` de `c1`.
   (c) Donner l'instruction permettant d'accéder à la valeur `temps_tot` de `c1`.
2. Pendant la course, des instances de la classe `Concurrent` sont créées au fur et à mesure des arrivées des concurrents. On définit une classe `Liste` pour les stocker au fur et à mesure. Cette classe implémente la structure de données abstraite liste dont l'interface est munie :
   - du constructeur qui ne prend pas de paramètre et qui crée une liste vide.
     Exemple : `L = Liste()`
   - de la méthode `est_vide` qui ne prend pas de paramètre et qui renvoie un booléen : `True` si la liste est vide, `False` sinon.
     Exemple : on considère la liste `L = <c1, c2, c3>`, où `c1`, `c2` et `c3` sont des instances de `Concurrent`. Alors, l'appel `L.est_vide()` renvoie `False`.
   - de la méthode `tete` qui ne prend pas de paramètre et qui renvoie un objet de type `Concurrent` ayant pour valeur le premier élément de la liste. Cet élément sera appelé tête de la liste dans la suite de l'exercice.
     Cette méthode ne s'applique que sur des listes non vides.
     Exemple : on considère la liste `L = <c1, c2, c3>`, où `c1`, `c2` et `c3` sont des instances de `Concurrent`. Alors, l'appel `L.tete()` renvoie `c1`.
     Remarque : après exécution de `L.tete()`, la liste `L` reste inchangée.
   - de la méthode `queue` qui ne prend pas de paramètre et qui renvoie la liste sur laquelle elle s'applique privée de son premier élément.
     Cette méthode ne s'applique que sur des listes non vides. Exemple : on considère la liste `L = <c1, c2, c3>`, où `c1`, `c2` et `c3` sont des instances de `Concurrent`. Alors, l'appel `L.queue()` renvoie la liste `<c2, c3>`.
     Remarque : après exécution de `L.queue()`, la liste `L` reste inchangée.
   - de la méthode `ajout` qui prend en paramètre un concurrent `c` et qui modifie la liste sur laquelle elle s'applique en ajoutant `c` en tête.
     Exemple 1 : si `L` est la liste vide, alors `L.ajout(c)` modifie la liste `L` en la liste `<c>`.
     Exemple 2 : si `L` est la liste `<c1, c2, c3>`, alors `L.ajout(c)` mofifie la liste `L` en la liste `<c, c1, c2, c3>`.

On considère le script Python suivant :

```
c1 = Concurrent("Mosquito", 87.67, 12)
c2 = Concurrent("Python Fute", 89.73, 4)
c3 = Concurrent("Piranha Vorace", 90.54, 0)
c4 = Concurrent("Truite Agile", 84.32, 52)
c5 = Concurrent("Tortue Rapide", 92.12, 2)
c6 = Concurrent("Lièvre Tranquille", 93.45, 0)

resultats = Liste()
resultats.ajout(c1)
resultats.ajout(c2)
resultats.ajout(c3)
resultats.ajout(c4)
resultats.ajout(c5)
resultats.ajout(c6)
```

Après exécution, ce script géère une liste `resultats` que l'on peut représenter par :

`<c6, c5, c4, c3, c2, c1>`

(a) On considère la liste `resultats` ci-dessus.
Donner la ou les instruction(s) qui permet(tent) d'accéder à `c4`.

(b) Donner la ou les instruction(s) qui permet(tent) d'accéder au temps total du concurrent stocké en tête de la liste `resultats`.

3. On souhaite créer une fonction `meilleur_concurrent` qui prend en paramètre un liste `L` de concurrents et qui renvoie l'objet `Concurrent` correspondant au concurrent le plus rapide. On suppose que la liste est non vide.
Recopier et compléter le code Python ci-dessous de la fonction `meilleur_concurrent`.

```
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
```

4. Pour simplifier le stockage des résultats, on décide de stocker les objets de la classe `Concurrent` dans un arbre binaire de recherche. Chaque nœud de cet arbre est donc un objet `Concurrent`. Dans cet arbre binaire de recherche, en tout nœud :

- le concurrent enfant à gauche est plus rapide que le nœud ;
- le concurrent enfant à droite est moins rapide que le nœud.

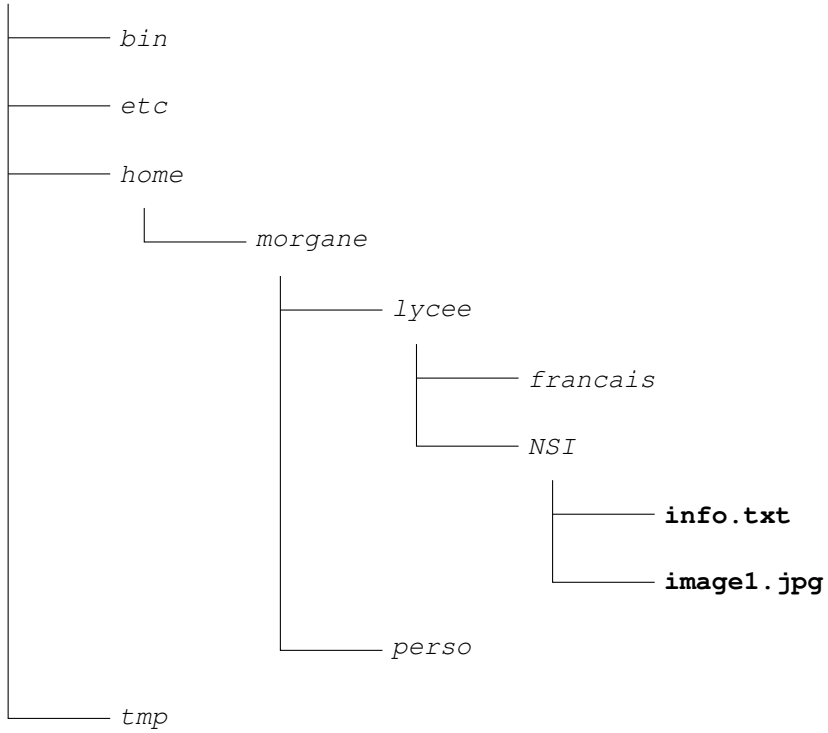
Pour implémenter la structure d'arbre binaire de recherche, on dispose d'une classe `Arbre` munie, entre autres, d'une méthode `ajout` qui prend en paramètre un objet `c` de type `Concurrent` et qui modifie l'arbre binaire sur lequel elle s'applique en y ajoutant le concurrent `c` tout en maintenant la propriété d'arbre binaire de recherche.
On ajoute dans un arbre vide successivement les concurrents de la liste `resultats` en partant de la tête de la liste (soit, dans le cas présent, `c6`, puis `c5`, puis `c4`, ...).
Dessiner l'arbre binaire de recherche obtenu. On rappelle le temps total de chaque concurrent :

Concurrent	c6	c5	c4	c3	c2	c1
temps_tot	93.45	94.12	136.32	90.54	93.73	99.67

Exercice 2 (OS et processus - 4 points)

Cet exercice pourra utiliser des commandes de systèmes d'exploitation de type UNIX telles que `cd`, `ls`, `mkdir`, `rm`, `rmd`, `mv` et `cat`.

1. Dans un système d'exploitation de type UNIX, on considère l'arborescence des fichiers suivante dans laquelle les noms de dossiers sont en italique et ceux des fichiers en gras :



On souhaite, grâce à l'utilisation du terminal de commande, explorer et modifier les répertoires et fichiers présents, en supposant qu'on se trouve actuellement à l'emplacement `/home/morgane`.

(a) Parmi les quatre propositions suivantes, donner celle correspondant à l'affichage obtenu lors de l'utilisation de la commande `ls`.
Proposition 1 : `lycee francais NSI info.txt image1.jpg perso`
Proposition 2 : `lycee perso`
Proposition 3 : `morgane`
Proposition 4 : `bin etc home tmp`

(b) Ecrire la commande qui permet, à partir de cet emplacement, d'atteindre le répertoire `lycee`.

On suppose maintenant qu'on se trouve dans le répertoire `/home/morgane/lycee/NSI`.

(c) Ecrire la commande qui permet de créer à cet emplacement un répertoire nommé `algorithmique`.

(d) Ecrire la commande qui permet, à partir de cet emplacement, de supprimer le fichier `image1.jpg`.

2. On rappelle qu'un processus est une instance d'application. Un processus peut être démarré par l'utilisateur, par un périphérique ou par un autre processus appelé parent.
La commande UNIX `ps` présente un cliché instantané des processus en cours d'exécution.
On a exécuté la commande `ps` (avec quelques options qu'il n'est pas nécessaire de connaître pour la réussite de cet exercice). Un extrait du résultat de la commande est présenté ci-dessous :

```
UID          PID  PPID  C STIME TTY          TIME CMD
test         900   739  0 10:51 ?        00:00:00 /usr/lib/gvfs/gvfs-udisks2-vol
test         907   838  0 10:51 ?        00:00:00 /usr/lib/gvfs/gvfsd-trash --sp
test         913   739  0 10:51 ?        00:00:00 /usr/lib/gvfs/gvfsd-metadata
test         918   823  0 10:51 ?        00:00:00 /usr/lib/x86_64-linux-gnu/xfce
test         919   823  0 10:51 ?        00:00:00 /usr/lib/x86_64-linux-gnu/xfce
test         923     1  0 10:51 ?        00:00:02 xfce4-terminal
test         927   923  0 10:51 pts/0    00:00:00 bash
test        1036     1  0 11:18 ?        00:00:02 mousepad /home/test/Documents/
test        1058   923  0 11:22 pts/1    00:00:00 bash
root        1132     2  0 11:37 ?        00:00:00 [kworker/0:0-ata_sff]
root        1134     2  0 11:43 ?        00:00:00 [kworker/0:2-ata_sff]
test        1140   739  0 11:43 ?        00:00:00 /usr/lib/x86_64-linux-gnu/tumb
root        1149     2  0 11:43 ?        00:00:00 [kworker/u2:0-events_unbound]
test        1153   927  0 11:44 pts/0    00:00:00 vi
test        1154   927  0 11:44 pts/0    00:00:00 python3 prog.py
test        1155  1058  0 11:44 pts/1    00:00:00 ps -aef
```

On rappelle que :

- ⋆ l'`UID` est l'identifiant de l'utilisateur propriétaire du processus ;
- ⋆ le `PID` est l'identifiant du processus ;
- ⋆ le `PPID` est l'identifiant du processus parent ;
- ⋆ `C` indique l'utilisation processeur ;
- ⋆ `STIME` est l'heure de démarrage du processus ;
- ⋆ `TTY` est le nom du terminal de commande auquel le processus est attaché ;
- ⋆ `TIME` est la durée d'utilisation du processeur par le processus ;
- ⋆ `CMD` est le nom de la commande utilisée pour démarrer le processus.

(a) Donner le `PID` du parent du processus démarré par la commande `vi`.

(b) Donner le `PID` d'un processus enfant du processus démarré par la commande `xfce4-terminal`.

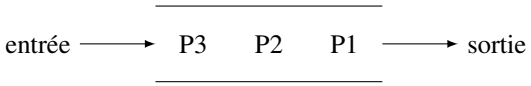
(c) Citer le `PID` de deux processus qui ont le même parent.

(d) Parmi tous les processus affichés, citer le `PID` des deux qui ont consommé le plus de temps du processeur.

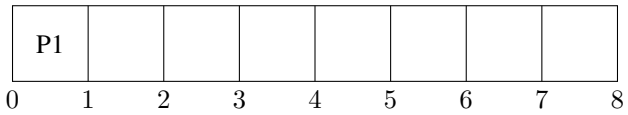
3. On considère les trois processus P1, P2 et P3, tous soumis à l'instant 0 dans l'ordre 1, 2 et 3 :

Nom du processus	Durée d'exécution en unité de temps	Ordre de soumission
P1	3	1
P2	1	2
P3	4	3

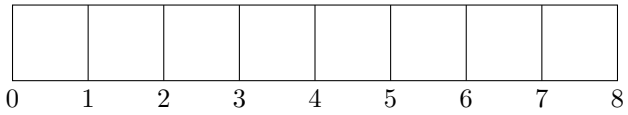
(a) Dans cette question, on considère que les processus sont exécutés de manière concurrente selon la politique du tourniquet : le temps est découpé en tranches nommées *quantums de temps*.
Les processus prêts à être exécutés sont placés dans une file d'attente selon leur ordre de soumission. Lorsqu'un processus est élu, il s'exécute au plus durant un quantum de temps. Si le processus n'a pas terminé son exécution à l'issue du quantum de temps, il réintègre la file des processus prêts (côté entrée). Un autre processus, désormais en tête de la file (côté sortie) des processus prêts, est alors à son tour élu pour une durée égale à un quantum de temps maximum.



Reproduire le tableau ci-dessous sur la copie et indiquer dans chacune des cases le processus exécuté à chaque cycle. Le quantum correspond à une unité de temps.



(b) Dans cette question, on considère que les processus sont exécutés en appliquant la politique du « plus court d'abord » : les processus sont exécutés complètement dans l'ordre croissant de leurs temps d'exécution, le plus court étant exécuté en premier.
Reproduire le tableau ci-dessous sur la copie et indiquer dans chacune des cases le processus exécuté à chaque cycle.



4. On considère trois ressources R1, R2 et R3, et trois processus P1, P2 et P3 dont les files d'exécution des instructions élémentaires sont indiquées ci-dessous :

Processus P1
Demande R1
Demande R2
Libère R1
Libère R2

Processus P2
Demande R2
Demande R3
Libère R2
Libère R3

Processus P3
Demande R3
Demande R1
Libère R3
Libère R1

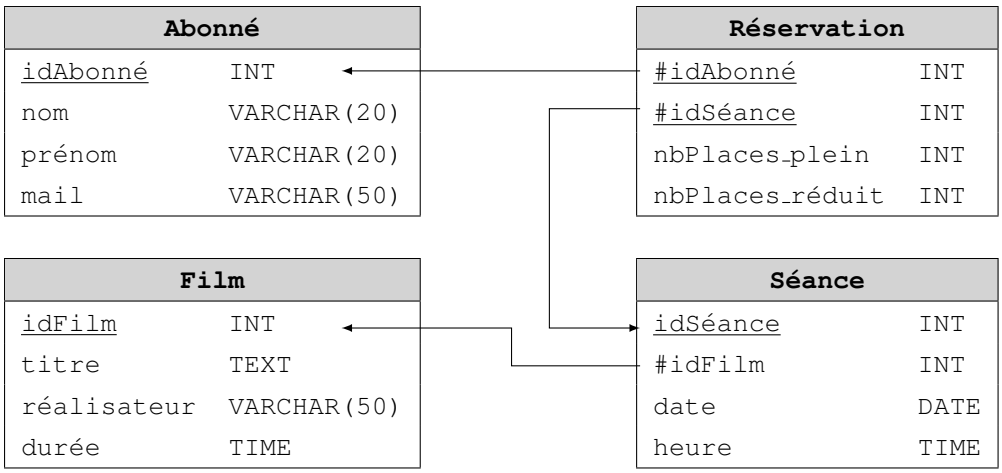
(a) Rappeler les différents états d'un processus et expliquer pourquoi il y a ici risque d'interblocage, en proposant un ordre d'exécution des instructions élémentaires le provoquant.

(b) Proposer un ordre d'exécution des instructions élémentaires sans interblocage.

Exercice 3 (Bases de données et SQL - 4 points)
Cet exercice utilise les mots du langage SQL suivants :

`SELECT`, `FROM`, `WHERE`, `JOIN`, `ON`, `UPDATE`, `SET`, `DELETE`, `COUNT`, `AND` et `OR`.

Une salle de cinéma propose un site Web à ses abonnés afin d'effectuer des réservations de séances en ligne. Deux tarifs sont proposés : plein et réduit (-16 ans, sénior de plus de 65 ans, étudiants, etc.). Le site est associé à une base de données dont le modèle relationnel contient quatre relations décrites ci-dessous :



Un attribut souligné correspond à une clé primaire et un attribut précédé du symbole # à une clé étrangère.

Voici un extrait de quelques enregistrements des relations `Film`, `Séance` et `Abonné` :

idFilm	titre	réalisateur	durée
1	Le sens de la famille	Jean-Patrick Benes	90
2	Les croods 2	Joel Crawford	95
8	Black widow	Cate Shortland	134

Extrait de la relation `Film` (les durées sont en minutes)

idSéance	idFilm	date	heure
35	1	2021-10-11	21:00
737	8	2021-10-11	21:00
738	8	2021-10-13	16:15

Extrait de la relation `Séance`

idAbonné	nom	prénom	mail
1	Henry	Jean	jean.henry@envoi.fr
2	Jacquin	Morgane	jacquin.morgane@mail.com
13	Dupont	Charles	charles.dupont@envoi.fr

Extrait de la table `Abonné`

1. (a) Définir le rôle d'une clé primaire.
   (b) Définir le rôle d'une clé étrangère.
   (c) Déterminer, en justifiant, si un abonné peut réserver plusieurs fois une même séance.
   (d) M. Charles Dupont réserve trois places au tarif plein et deux places au tarif réduit pour assister à la projection du film `Black widow` le 11 octobre 2021 à 21h. A l'aide des extraits des relations donnés précédemment, recopier et compléter l'enregistrement correspondant dans la relation `Réservation` ci-dessous :

idAbonné	idSéance	nbPlaces_plein	nbPlaces_réduit

2. (a) Parmi les trois requêtes SQL suivantes, recopier celle qui permet d'afficher le titre et le réalisateur des films de moins de 120 minutes.

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

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

```
SELECT titre
FROM Film
WHERE durée < 120
```

(b) En SQL, la fonction `COUNT` permet de compter le nombre d'enregistrements dans une table. Exprimer en langage naturel la requête SQL suivante :

```
SELECT COUNT(*)
FROM Séance
WHERE date = "2021-10-22" OR date = "2021-10-23"
```

On remarquera que les dates apparaissent sous la forme `aaaa-mm-jj`.
Par exemple, le 11 octobre 2021 apparaît sous la forme `2021-10-11`.

3. Ecrire en SQL les requêtes permettant d'effectuer les tâches suivantes :
   (a) Afficher le nom et le prénom de tous les abonnés.
   (b) Afficher le titre et la durée des films projetés le 12 octobre 2021 à 21h.
   On remarquera que les heures apparaissent sous la forme `hh:mm`.
4. (a) Ecrire une requête SQL permettant de modifier la durée du film `Jungle Cruise` (initialement enregistré avec `90` à `127`.
   (b) On souhaite écrire une requête SQL permettant de supprimer la séance dont l'attribut `idSéance` vaut `135`.
   Déterminer la contrainte d'intégrité que pourrait violer cette requête.
   (c) Ecrire la requête précédente en SQL.

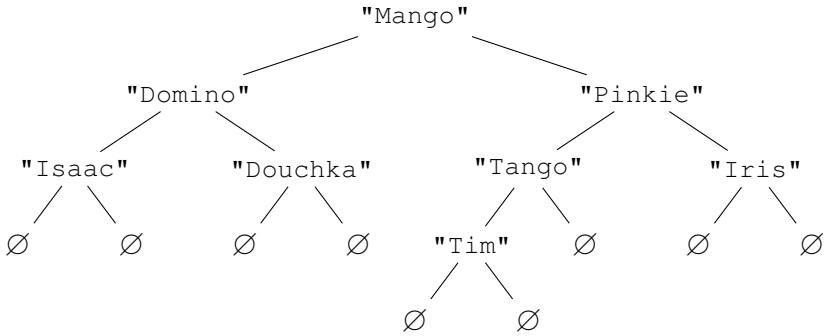
Exercice 4 (Arbres binaires - 4 points)

Un éleveur de chiens gère les informations sur ses animaux à l'aide d'un logiciel qui mémorise le pédigrée de chacun de ses chiens. Le pédigrée d'un chien correspond à son arbre généalogique.

Une structure **arbre de pédigrée** est définie récursivement, soit par un arbre vide, noté $\varnothing$, soit par un arbre binaire où

- la valeur du nœud est une chaîne de caractères qui représente le nom de l'animal ;
- le sous-arbre gauche est l'arbre de pédigrée du père du chien ;
- le sous-arbre doirt est l'arbre de pédigrée de la mère du chien.

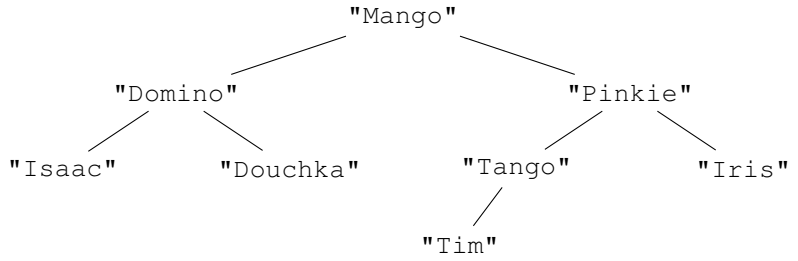
On représente donc graphiquement un arbre de pédigrée comme l'arbre `A` suivant :



Dans cet arbre :

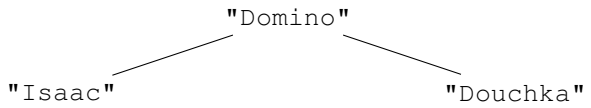
- le père de Mango est Domino et sa mère Pinkie ;
- les parents de Douchka ne sont pas connus ;
- Iris est la mère de Pinkie ;
- la mère de Tango n'est pas connue.

Pour alléger la représentation d'un arbre de pédigrée, on ne notera pas les arbres vides. L'arbre précédent sera donc représenté comme ci-dessous :

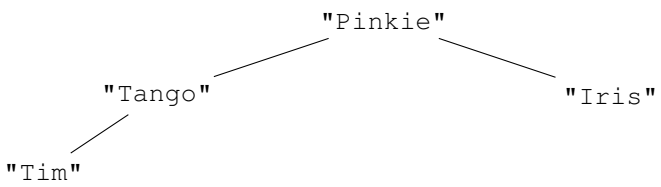


Pour manipuler les arbres de pédigrée, on dispose des quatre fonctions suivantes :

- La fonction `racine` qui prend en paramètre un arbre de pédigrée non vide et renvoie la valeur de sa racine.
  Exemple : en reprenant l'arbre de pédigrée `A` précédent, l'appel `racine(A)` renvoie `"Mango"`.
- La fonction `gauche` qui prend en paramètre un arbre de pédigrée non vide et renvoie son sous-arbre gauche correspondant à l'arbre de pédigrée du père.
  Exemple : en reprenant l'arbre de pédigrée `A` précédent, l'appel `gauche(A)` renvoie l'arbre représenté graphiquement ci-après :



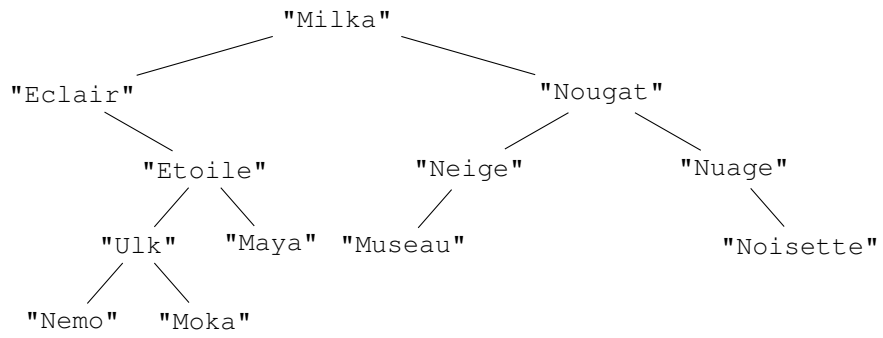
- La fonction `droit` qui prend en paramètre un arbre de pédigrée non vide et renvoie son sous-arbre droit correspondant à l'arbre de pédigrée de la mère.
  Exemple : en reprenant l'arbre de pédigrée `A` précédent, l'appel `droit(A)` renvoie l'arbre représenté graphiquement ci-après :



⋆ La fonction `est_vide` qui prend en paramètre un arbre de pédigrée et renvoie `True` si l'arbre est vide, `False` sinon.
Exemple : en reprenant l'arbre de pédigrée `A` précédent, l'appel `est_vide(A)` renvoie `False`.

Pour toutes les questions de l'exercice, on suppose que tous les chiens d'un même pédigrée ont un nom différent.

1. On considère l'arbre de pédigrée `B` suivant :



   (a) Déterminer la valeur de la racine de cet arbre.

   (b) On appelle feuille d'un arbre de pédigrée un nœud dont les sous-arbres gauche et droit sont vides.
   Déterminer l'ensemble des valeurs des feuilles de cet arbre.

   (c) Déterminer si `"Nuage"` est un mâle ou une femelle.

   (d) Déterminer le père et la mère de `"Etoile"`.

2. (a) Recopier et compléter la fonction récursive Python `present` ayant pour paramètre un arbre de pédigrée `arb` et le nom d'un chien `nom`, et qui renvoie `True` si ce nom est présent dans l'arbre de pédigrée et `False` sinon.

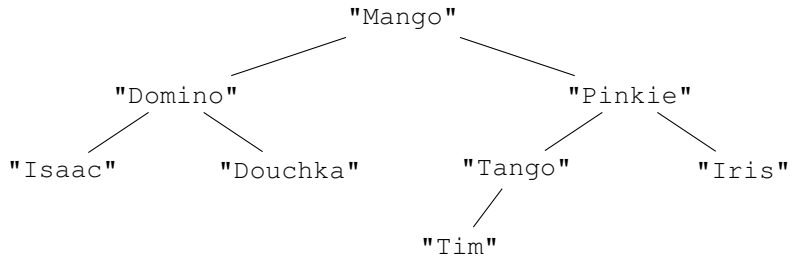
```
1  def present(arb, nom):
2      if est_vide(arb):
3          return False
4      elif racine(arb) == nom:
5          return True
6      else :
7          return present(droit(arb), nom) or present(gauche(arb), nom)
```

   Pour toute la suite de l'exercice, on pourra utiliser la fonction `present` même si la question 2.(a) n'a pas été traitée.

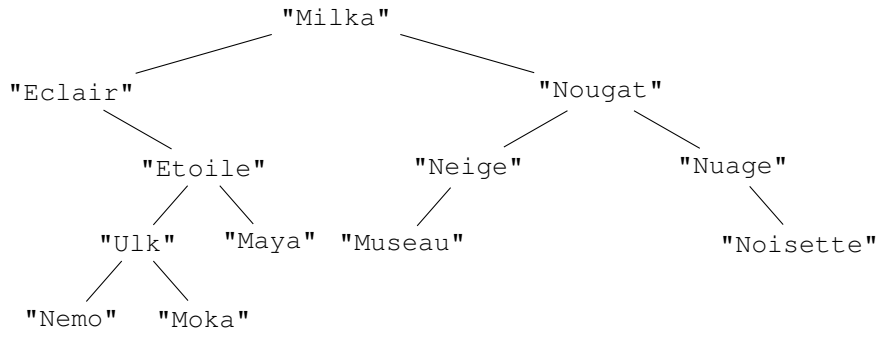
   (b) Ecrire une fonction Python `parents` ayant pour paramètre un arbre de pédigrée `arbre` d'un chien et qui renvoie le p-puplets des deux parents de ce chien dans l'ordre père, mère. Si un des parents est inconnu, il sera noté `""`.
   Exemple : l'instruction `parents(B)` renvoie `("Eclair","Nougat")`.

3. (a) On dit que deux chiens sont frères et sœurs s'ils ont le même père ou la même mère.
   On considère les trois arbres de pédigrée suivants :

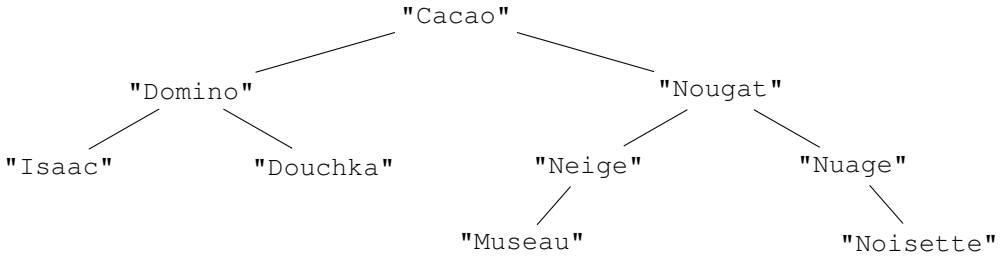
   Arbre `A` :



Arbre `B` :



Arbre `C` :

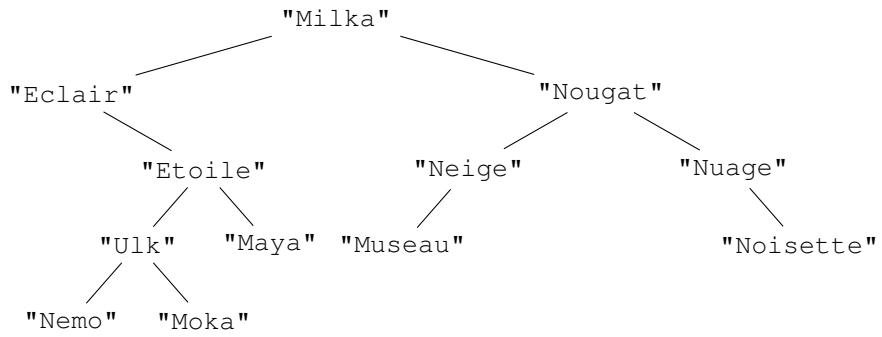


Parmi les trois chiens Mango, Milka et Cacao, déterminer les liens de fratrie.

(b) Ecrire une fonction Python `frere_soeur` ayant pour paramètres deux arbres de pédigrée `arbre1` et `arbre2` correspondant à deux chiens. Cette fonction renvoie `True` si les deux chiens ont le même père ou la même mère, et `False` sinon.

4. Etant donné l'arbre de pédigrée d'un chien, on considère que :
   - le niveau 0 est le niveau de la racine contenant le nom du chien ;
   - le niveau 1 est le niveau des parents du chien ;
   - le niveau 2 est le niveau des grands-parents du chien ;
   - etc.

   Proposer une fonction Python `nombre_chiens` ayant pour paramètres un arbre de pédigrée `arb` et un entier `n` et renvoyant le nombre de noms connus dans l'arbre de pédigrée `arb` au niveau `n`.
   Exemple : on considère l'arbre de pédigrée `B` suivant :



   L'instruction `nombre_chiens(B,3)` renvoie `4` car les noms des chiens mentionnés dans l'arbre de pédigrée au niveau 3 sont `"Ulk"`, `"Maya"`, `"Museau"` et `"Noisette"`.

Exercice 5 (Tableaux et programmation - 4 points)

Partie A.

Dans cette partie, on s'intéresse à un dessin pixelisé en noir et blanc dans lequel chaque ligne est obtenue séquentiellement à partir de la ligne juste au-dessus d'elle.

Au départ, on dispose donc de la ligne du haut et on déduit les lignes en-dessous les unes après les autres. Les règles sont les suivantes :

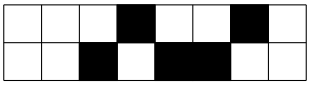
- pour chaque ligne, le pixel complètement à droite et le pixel complètement à gauche sont blancs ;
- pour les autres pixels, un pixel est noir si les deux pixels à droite et à gauche du pixel juste au-dessus de ce pixel ne sont pas de la même couleur, sinon il est blanc.

a		b
	x	

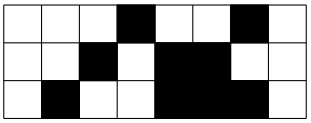
Le pixel noté "x" est donc blanc si les pixels notés "a" et "b" sont de la même couleur, et noir sinon.

1. (a) Dans cette question uniquement, on considère un dessin de cinq lignes et huit colonnes dont la première ligne est colorée comme ci-dessous :

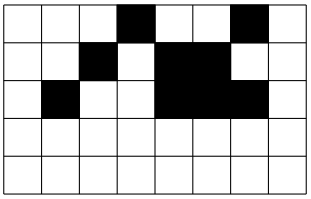
En respectant les règles de coloration, on complète la figure en ajoutant la deuxième ligne :



De même, en ajoutant la troisième ligne :



Recopier et colorier les deux dernières lignes du dessin ci-dessous en respectant les règles :



Dans la suite cet exercice :

- les lignes sont numérotées du haut vers le bas, en commençant à 0 ;
- les colonnes sont numérotées de la gauche vers la droite, en commençant à 0.

(b) On considère le pixel de la ligne 4 et de la colonne 1.

- Indiquer sur quelle ligne et sur quelle colonne se trouve le pixel situé sur la ligne juste au-dessus, à sa gauche.
- Même question pour le pixel situé sur la ligne juste au-dessus, à sa droite.

(c) Plus généralement, on considère le pixel de la ligne `li` et de la colonne `co` et situé ni sur la première ligne, ni sur la colonne tout à gauche, ni sur la colonne tout à droite.

- Indiquer sur quelle ligne et sur quelle colonne se trouve le pixel situé sur la ligne juste au-dessus, à sa gauche.
- Même question pour le pixel situé sur la ligne juste au-dessus, à sa droite.

2. On dispose d'un tableau `image` à deux dimensions (m lignes et 8 colonnes) qui contient uniquement des 0 ou des 1. `image[li][co]` vaut `1` si le pixel de la ligne `li` et de la colonne `co` est noir et `0` sinon.
Pour chaque valeur de `li`, `image[li][0] = 0` et `image[li][7] = 0`.

(a) On suppose que le tableau `image` a été rempli en respectant les règles de coloration jusqu'à la ligne `li - 1` et contient des `0` partout ailleurs. On suppose de plus que `0 < co < 7`.
On cherche à connaître quelle valeur la case `image[li][co]` doit prendre si on respecte les règles de coloration. Donner les conditions portant sur les valeurs de la ligne `li - 1` qui doivent être satisfaites pour que `image[li][co]` prenne la valeur `1`.

(b) On suppose toujours que le tableau `image` a été rempli en respectant les règles de coloration jusqu'à la ligne `li - 1` et contient des `0` partout ailleurs.
Recopier et compléter la fonction Python `remplir_ligne` qui prend en paramètre le tableau `image` et un entier `li`. Cette fonction affecte à chaque case de la ligne `li` de `image` la valeur correspondant à la couleur du pixel en respectant les règles de coloration (dans le code de la fonction, les trois points `...` peuvent correspondre à une ou plusieurs lignes de programme).

```
def remplir_ligne(image, li):
    image[li][0] = 0
    image[li][7] = 0
    for ... in range(..., ...):
        ...
```

(c) La première ligne du tableau `image` est remplie correctement.
Ecrire une fonction `remplir` qui prend en paramètre le tableau `image`. Cette fonction modifie les valeurs des cases du tableau `image` des lignes restantes.

Partie B.

A un tableau de taille 8 contenant des 0 ou des 1, on associe l'entier dont la représentation binaire est donnée par ce tableau.
Par exemple, le tableau `[0, 0, 0, 1, 0, 0, 1, 0]` représente le nombre binaire `00010010`, correspondant à 18 en base 10.

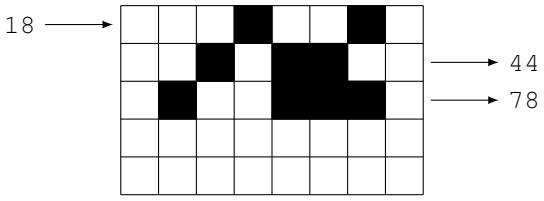
1. (a) Soit le tableau `[0, 0, 1, 0, 1, 1, 0, 0]`. Déterminer la représentation en base 10 de l'entier correspondant.

   (b) Ecrire la fonction Python `conversion2_10` qui prend en paramètre un tableau `tab` de taille 8 composé de 0 et de 1, et qui renvoie l'entier correspondant, écrit en base 10.
   Exemple : l'instruction `conversion2_10([0, 0, 0, 1, 0, 0, 1, 0])` renvoie `18`.

   (c) Réciproquement, donner le tableau associé à l'entier dont l'écriture en base 10 est 78.

   Dans la suite, on suppose qu'on dispose de la fonction Python `conversion10_2` qui prend en paramètre un entier `n` compris entre 0 et 255, et qui renvoie le tableau `tab` correspondant.
   Exemple : l'instruction `conversion10_2(18)` renvoie `[0, 0, 0, 1, 0, 0, 1, 0]`.

2. A partir d'un entier n, on construit une liste de k entiers en utilisant l'image étudiée dans la partie A de la façon suivante :
   - ⋆ l'entier n fournit la première ligne de l'image sous forme d'un tableau de taille 8 représentant son écriture binaire ;
   - ⋆ chaque ligne suivante de l'image est obtenue selon les règles appliquées dans la partie A et permet de déterminer un entier de la liste.



Par exemple, l'entier 18 fournit la liste 44, 78, etc.

(a) On rappelle que pour chaque ligne de l'image, le pixel complètement à gauche et le pixel complètement à droite restent blancs. En déduire une ou des précondition(s) sur l'entier n.

(b) Recopier et compléter la fonction `generer` qui prend en paramètres deux entiers `n` et `k`, et renvoie le tableau `tab` de taile `k` contenant les `k` entiers obtenus à partir de `n` (dans le code de la fonction, les trois points `...` peuvent correspondre à une ou plusieurs lignes de programme).

```
def generer(n,k):
    tab = [None for i in range(k)]
    image = [[0 for j in range(8)] for i in range(k+1)]
    t = conversion10_2(n)
    for i in range(8):
        image[0][i] = t[i]
    tab[0]=n
    for li in range(1,k):
        remplir_ligne(image, li)
        tab[li] = conversion2_10(image[li])
    return tab
```