Correction

NSI - 2021 Métropole Libre Jour 2 (21-NSIJ2ME2)

Exercice 1 - Base de données d'un CDI

1. Ce code provoque une erreur car deux éleves ont le même identifiant `idEleve` or celui-ci servant de clé primaire, il doit permettre d'identifier chaque élève de manière unique dans la table.

2. Dans la définition de la relation `Emprunts` ce qui assure qu'on ne puisse pas enregistrer un emprunt pour un élève qui n'est pas inscrit dans la relation `Eleves` est la clé étrangère `idEleve`.

3. `SELECT titre FROM Livres WHERE auteur = 'Molière' ;`

4. Le résultat renvoyé par cette requête est le nom et le prénom de tous les élèves de la classe T2 qui ont emprunté au moins un livre.

5. `UPDATE Emprunts SET dateRetour = '2020-09-30' WHERE idEmprunt = 640 ;`

6. Cette requête renvoie tous les noms et prénoms des élèves de la classe T2 (en évitant les doublons) qui ont emprunté un livre

7.

```
SELECT Eleves.nom, Eleves.prenom
FROM Eleves, Emprunts, Livres
WHERE Livres.titre = 'Les misérables' AND Livres.isbn = Emprunts.isbn
AND Emprunts.idEleve =Eleves.idEleve ;
```

Autre solution :

```
SELECT Eleves.nom, Eleves.prenom FROM Eleves
JOIN Emprunts ON Emprunts.idEleve = Eleves.idEleve
JOIN Livres ON Livres.isbn = Emprunts.isbn
WHERE Livres.titre = 'Les misérables' ;
```

Et avec des alias :

```
SELECT El.nom, El.prenom FROM Eleves AS El
JOIN Emprunts AS Em ON Em.idEleve = El.idEleve
JOIN Livres AS Li ON Li.isbn = Em.isbn
WHERE Li.titre = 'Les misérables' ;
```

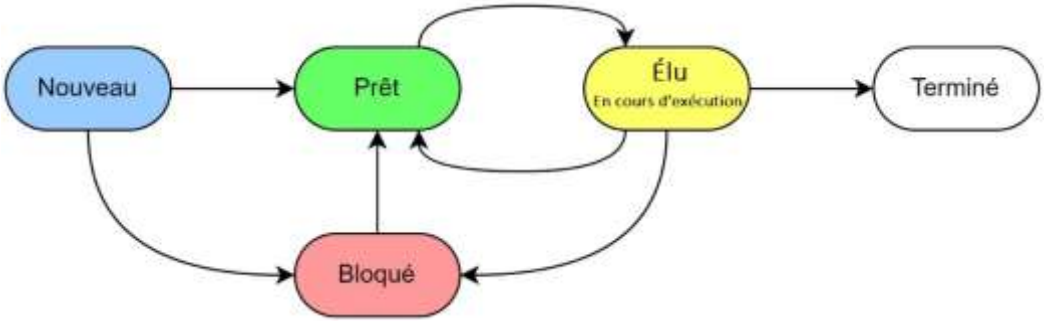
Correction

NSI - 2021 Métropole Libre Jour 2 (21-NSIJ2ME2)

Exercice 2 - Système Exploitation, Gestion des processus

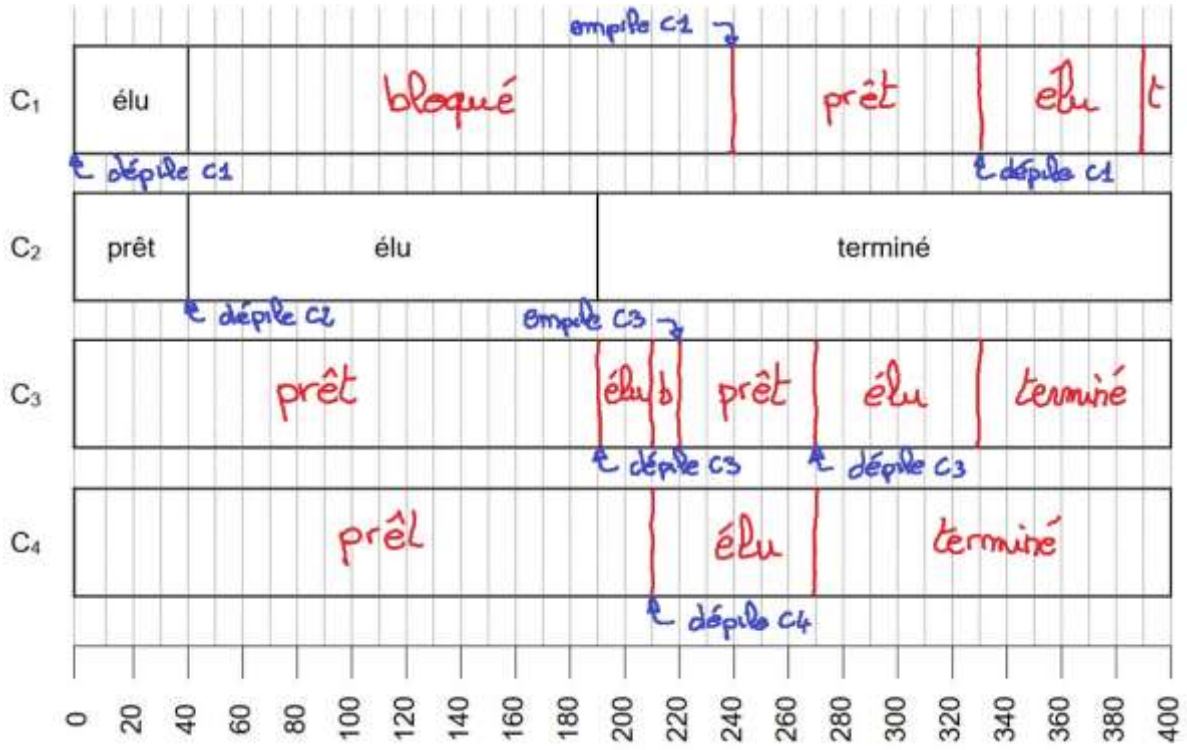
1. a. L'état *élu* correspond à un processus en cours d'exécution, c'est-à-dire en train d'utiliser le processeur (CPU).

1. b.



2. a. Le principe d'une file est « Premier entré, premier sorti ».

2. b.



3. a. Les deux programmes s'attendent mutuellement : il y a interblocage (deadlock).
Ainsi quand le programme 1 verrouille le fichier 1, il est impossible au programme 2 d'y accéder, il est donc mis en attente. Et le programme 2 verrouille le fichier 2 ce qui empêche le programme 1 d'y accéder, il y a un problème d'interblocage.

3. b. Pour éviter l'interblocage, il faut inverser l'ordre de demande d'accès aux fichiers :

```
Verrouiller fichier_1
Verrouiller fichier_2
Calculs sur fichier_1
Calculs sur fichier_2
Déverrouiller fichier_1
Déverrouiller fichier_2
```

Correction

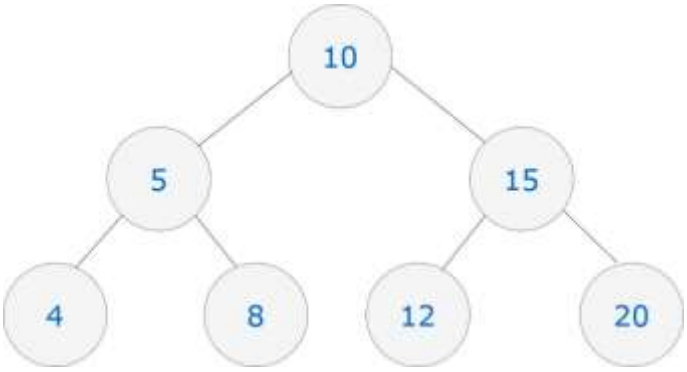
NSI - 2021 Métropole Libre Jour 2 (21-NSIJ2ME2)

Exercice 3 - Arbres binaires, Programmation Orientée Objet

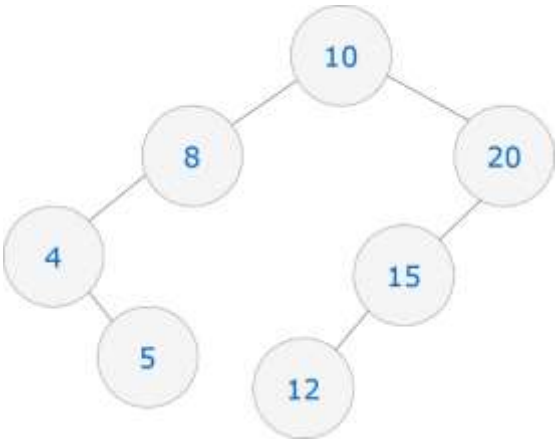
1. a. La taille de l'arbre est de 7.

1. b. La hauteur de cet arbre est de 4.

2.



3.



4.
```
def hauteur(self):
    return self.racine.hauteur()
```

5. Méthode taille de la classe Nœud :

```
def taille(self):
    if self.gauche and self.droit:
        return 1 + self.gauche.taille() + self.droit.taille()
    elif self.gauche:
        return 1 + self.gauche.taille()
    elif self.droit:
        return 1 + self.droit.taille()
    else:
        return 1
```

Correction

NSI - 2021 Métropole Libre Jour 2 (21-NSIJ2ME2)

Autre solution pour la méthode taille de la classe Nœud :

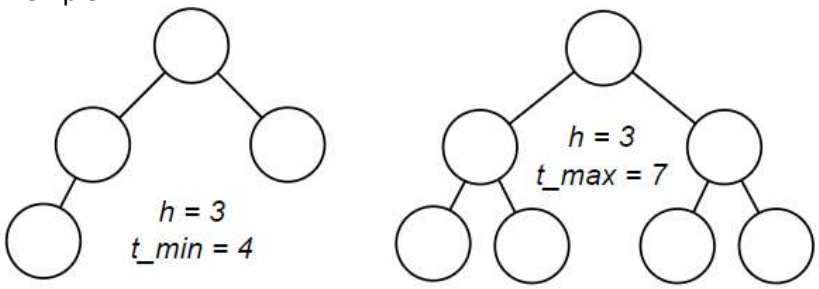
```
def taille(self):
    if self.gauche == None and self.droite == None:
        return 1
    elif self.gauche == None:
        return 1 + self.droite.taille()
    elif self.droite == None:
        return 1 + self.gauche.taille()
    else:
        return 1 + self.gauche.taille() + self.droite.taille()
```

Méthode taille de la Classe Arbre :

```
def taille(self):
    return self.racine.taille()
```

6. a. Pour trouver la taille minimale d'un arbre binaire de recherche « bien construit » de hauteur h il faut considérer la taille maximale d'un arbre de hauteur $h-1$ auquel on ajoute un nœud.
Soit $t\_\min = 2\hat{}(h-1) - 1 + 1 = 2\hat{}(h-1)$.

Exemple :



6. b.

```
def bien_construit(self):
    h = self.racine.hauteur()
    return 2**(h - 1) <= self.racine.taille() <= 2**h - 1
```

Correction

NSI - 2021 Métropole Libre Jour 2 (21-NSIJ2ME2)

Exercice 4 - Mélange des éléments d'une liste

1. Ce code ne réalise pas un échange, il écrase la valeur d'indice i2 par la valeur d'indice i1. Ainsi la valeur initiale d'indice i2 est perdue.

Solution 1 :

```
def echange(lst, i1, i2):
    a = lst[i1]
    lst[i1] = lst[i2]
    lst[i2] = a
```

Solution 2 : Affectation multiple :

```
def echange(lst, i1, i2):
    lst[i2], lst[i1] = lst[i1], lst[i2]
```

2. Les valeurs qui peuvent être renvoyée par la fonction `randint(0, 10)` sont : 0, 1, 9 et 10.

3. a. La fonction mélange se termine car à chaque appel récursif, la valeur `ind` diminue, ce qui rendra fausse la condition `ind > 0` et arrêtera l'empilement d'appel récursif.

3. b. Pour une liste de longueur n, il y a $n-2$ appels récursifs. En effet, on ne compte pas le premier appel (pour la première valeur de n) et pour la dernière valeur de n ($n = 0$) il n'y a pas d'appel.

3. c. Tableau d'exécution :

Lst **(affichage)**	ind	j
[0, 1, 2, 3, 4]	4	2
[0, 1, 4, 3, 2]	3	1
[0, 3, 4, 1, 2]	2	2
[0, 3, 4, 1, 2]	1	0
[3, 0, 4, 1, 2]	0	

3. d.

Solution 1 :

```
def melange_it(lst, ind):
    for i in range(ind):
        j = randint(0, ind)
        echange(lst, ind-i, j)
```

Solution 2 :

```
def melange_it(lst, ind):
    for i in range(ind, 0, -1):
        j = randint(0, i)
        echange(lst, i, j)
```

Correction

NSI - 2021 Métropole Libre Jour 2 (21-NSIJ2ME2)

Exercice 5 - Programmation, Niveau 1re

1. a. Si tous les éléments du tableau sont positifs, en prenant comme sous-séquence le tableau entier, on obtiendra la valeur la plus grande.

1. b. Si tous les éléments du tableau sont négatifs, il faudra prendre la sous-séquence constituée d'une seule valeur, celle qui est la plus proche de zéro.

2. a.

Solution 1 :

```
def somme_sous_sequence(lst, i, j):
    somme = 0
    for i in range(i, j+1):
        somme = somme + lst[i]
    return somme
```

Solution 2 :

```
def somme_sous_sequence2(lst, i, j):
    return sum(lst[i:j+1])
```

2. b. La bonne réponse est 55. En effet on constate qu'il y a deux boucles imbriquées parcourant la liste.

```
for i in range(n):
      for j in range(i, n):
```

La complexité de l'algorithme est quadratique $O(n^2)$.
Pour un tableau de 10 cases, on peut former, 10 blocs d'une case, 9 blocs de 2 cases, 8 blocs de 3 cases, etc … 1 bloc de 10 cases.
$1 + 2 + 3 + \cdots + n = n(n+1)/2$ soit pour $n = 10$, $10 \times (10+1)/2 = 55$ comparaisons.

2. c.

```
def pgsp(lst):
    n = len(lst)
    somme_max = lst[0]
    for i in range(n):
        for j in range(i, n):
            s = somme_sous_sequence(lst, i, j)
            if s > somme_max :
                somme_max = s
                deb, fin = i, j
    return somme_max, deb, fin
```

3. a.

i	0	1	2	3	4	5	6	7
lst[i]	-8	-4	6	8	-6	10	-4	-4
S(i)	-8	-4	6	14	8	18	14	10

Correction

NSI - 2021 Métropole Libre Jour 2 (21-NSIJ2ME2)

3. b.

```
def pgsp2(lst):
    sommes_max = [lst[0]]
    for i in range(1, len(lst)):
        if sommes_max[i-1] <= 0:  # sommes_max[-1] revient au même
            sommes_max.append(lst[i])
        else:
            sommes_max.append(lst[i] + sommes_max[i-1])  # ou avec sommes_max[-1]
    return max(sommes_max)
```

3. c. La solution obtenue par cette approche est plus avantageuse que celle de la question 2. b. car elle permet d'éviter de refaire plusieurs calculs identiques. Sa complexité est inférieure car on ne parcourt qu'une fois le tableau. C'est une complexité linéaire $O(n)$ ce qui est plus efficace que la complexité quadratique précédente $O(n^2)$. L'exécution de la fonction sera plus rapide.