

[Index des sujets 2022](#)

22-NSIJ1LR1 : Corrigé

Année : **2022**

Centre : **Mayotte et réseau AEFE**

Jour : **1**

Enoncé : [PDF](#)

1. Exercice 1

structures de données (listes, piles et files)

1. a. Le `1` est défilé et enfilé

| | | | | |
|---|---|---|---|---|
| 1 | 4 | 3 | 8 | 2 |
|---|---|---|---|---|

b. Le `5` est défilé et empiler (la pile reste donc dans l'état où elle se trouvait)

| |
|---|
| 5 |
| 8 |
| 6 |
| 2 |

c. Le `5` est défilé de `p` et enfilé dans `f` puis le `8` est défilé de `p` et enfilé dans `f`.

| | | | | | | |
|---|---|---|---|---|---|---|
| 8 | 5 | 4 | 3 | 8 | 2 | 1 |
|---|---|---|---|---|---|---|

| |
|---|
| 6 |
| 2 |

d. Le `1` est défiler de `f` et empiler dans `p` puis le `2` est défilé de `f` empilé dans `p`.

| | | |
|---|---|---|
| 4 | 3 | 8 |
|---|---|---|

| |
|---|
| 2 |
| 1 |
| 5 |
| 8 |
| 6 |
| 2 |



Bug

A la question suivante, l'énoncé indique que la fonction `mystere` modifie la file mais *ne renvoie rien*. Cependant, on trouve à la fin du code de `mystere` un return `p`. Cette fonction renvoie donc une pile (qui comme nous le verrons est vide)

2. Avant le premier passage `f` contient



Puis, les états de la file `f` lors des passages successifs dans la première boucle `while` seront :

- Tour 1 : `f` contient



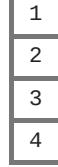
- Tour 2 : `f` contient



- Tour 3 : `f` contient



A la fin de cette première boucle, les éléments de `f` sont empilés dans `p` qui contient alors :



Dans la deuxième boucle `while` on dépile les éléments de `p` et on les enfile dans `f` :

- Tour 1 `f` contient



- Tour 2 `f` contient



- Tour 3 `f` contient



- Tour 4 `f` contient



A la fin de la seconde boucle `while` la pile `p` est vide, cette fonction renvoie donc une pile vide.

3. a.

| | | | | | | | | | | | | | | | | | | | | | |
|----------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <code>f</code> | <table border="1"><tr><td>2</td><td>1</td><td>3</td></tr></table> | 2 | 1 | 3 | <table border="1"><tr><td>2</td><td>1</td></tr></table> | 2 | 1 | <table border="1"><tr><td>3</td><td>2</td></tr></table> | 3 | 2 | <table border="1"><tr><td>3</td></tr></table> | 3 | <table border="1"><tr><td>3</td></tr></table> | 3 | <table border="1"><tr><td>2</td><td>3</td></tr></table> | 2 | 3 | <table border="1"><tr><td>1</td><td>2</td><td>3</td></tr></table> | 1 | 2 | 3 |
| 2 | 1 | 3 | | | | | | | | | | | | | | | | | | | |
| 2 | 1 | | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | | | | | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | | | | | | |
| 2 | 3 | | | | | | | | | | | | | | | | | | | | |
| 1 | 2 | 3 | | | | | | | | | | | | | | | | | | | |
| <code>p</code> | <table border="1"><tr><td></td></tr></table> | | <table border="1"><tr><td>3</td></tr></table> | 3 | <table border="1"><tr><td>1</td></tr></table> | 1 | <table border="1"><tr><td>1</td></tr></table> | 1 | <table border="1"><tr><td>2</td></tr></table> | 2 | <table border="1"><tr><td>1</td></tr></table> | 1 | <table border="1"><tr><td></td></tr></table> | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | |

b.

 Bug

Cette algorithme semble conçu pour trier les éléments de la file mais ne fonctionne pas en l'état. En effet, la fonction `knuth` semble vouloir à chaque tour de la boucle `for`, avoir une pile `p` triée. Pour cela, les éléments de `p` supérieurs au nouvel élément à empiler `e` devraient être stockées dans une pile temporaire `p_temp`, ici ces éléments sont enfilés dans la file à trier `f`.

2. Exercice 2

structures de données (programmation objet)

1. a. On augmente l'indice `i` sans dépasser la longueur de la liste `Mousse` tant qu'on ne trouve pas `None` :

 Script Python

```

1 def donnePremierIndiceLibre(Mousse):
2     """
3         Mousse est une liste.
4         La fonction doit renvoyer l'indice du premier
5         emplacement libre (contenant None) dans la liste Mousse
6         ou renvoyer 6 en l'absence d'un emplacement libre dans
7         Mousse.
8     """
9     i = 0
10    while i < len(Mousse) and Mousse[i] != None :
11        i = i + 1
12    return i

```

- b. Si un indice libre existe alors on place `B` à cet indice dans `Mousse` :

 Script Python

```

1 def placeBulle(B):
2     libre = donnePremierIndiceLibre(Mousse)
3     if libre < len(Mousse):
4         Mousse[libre] = B

```

2. Les deux bulles sont en contacts si la distance les séparant est inférieur à la somme de leurs rayons.

 Script Python

```

1 def bullesEnContact(B1,B2):
2     return distanceEntreBulles(B1,B2) <= B1.rayon + B2.rayon

```

 Aide

Le code ci-dessus est équivalent à :

 Script Python

```

1  def bullesEnContact(B1,B2):
2      if distanceEntreBulles(B1,B2) <= B1.rayon + B2:
3          return True
4      else:
5          return False

```

3. A la ligne 10, on indique que la surface de la nouvelle bulle est la somme des surfaces des deux bulles entrant en collision. Les lignes 13 et 14 divisent par 2 les deux composantes de la vitesse. Et enfin ligne 16, la petite bulle disparaît donc la valeur de l'indice qu'elle occupait devient `None`

 Script Python

```

1  def collision(indPetite, indGrosse, Mousse) :
2      """
3          Absorption de la plus petite bulle d'indice indPetite
4          par la plus grosse bulle d'indice indGrosse. Aucun test
5          n'est réalisé sur les positions.
6      """
7
8      # calcul du nouveau rayon de la grosse bulle
9      surfPetite = pi*Mousse[indPetite].rayon**2
10     surfGrosse = pi*Mousse[indGrosse].rayon**2
11     surfGrosseApresCollision = surfPetite + surfGrosse
12     rayonGrosseApresCollision = sqrt(surfGrosseApresCollision/pi)
13     #réduction de 50% de la vitesse de la grosse bulle
14     Mousse[indGrosse].dirx = Mousse[indGrosse].dirx/2
15     Mousse[indGrosse].diry = Mousse[indGrosse].diry/2
16     #suppression de la petite bulle dans Mousse
17     Mousse[indPetite] = None

```

 Aide

On rappelle que la surface d'un disque de rayon r est πr^2

 Bug

- Le rayon de la grosse bulle est modifié lors d'une collision, on devrait donc trouver dans le code de la fonction `collision` la ligne `Mousse[indGrosse].rayon = rayonGrosseApresCollision`.
- Dans les paramètres d'appel `mousse` est en minuscule dans l'énoncé.
- Des espaces superflus figurent dans l'énoncé (par exemple entre `Mousse` et `[indPetite]`), on les a supprimé dans la correction pour respecter la notation usuelle de Python.

3. Exercice 3

bases de données relationnelles et langage SQL

1. a. Cette requête retourne les `titre` de la table `qcm` dont la date est après le 10/01/2022. C'est à dire :

| titre |
|----------------|
| POO |
| Arbre Parcours |

b.

Requête SQL

```
SELECT note FROM lien_eleve_qcm WHERE ideleve = 4;
```

2. a. Le couple `(ideleve, idqcm)` est la clé primaire de la table `lien_eleve_qcm`, or une clé primaire est *unique* et donc deux enregistrements dans cette table ne peuvent avoir les mêmes valeurs pour le couple `(ideleve, idqcm)` c'est à dire qu'un même élève ne peut pas avoir fait deux fois le même qcm.
 b. La table `lien_eleve_qcm` est modifiée, on doit y ajouter l'enregistrement `(4, 2, 18)` car l'`ideleve` de *Marty Mael* est 4, qu'il a fait le `qcm` d'`idqcm` 2 et qu'il a eu la note de 18.

c.

Requête SQL

```
INSERT INTO eleves VALUES (6, "Lefèvre", "Kevin")
```

d.

Requête SQL

```
DELETE FROM lien_eleve_qcm WHERE ideleve=2
```

3. a.

Requête SQL

```
SELECT nom, prenom FROM eleves
JOIN lien_eleve_qcm ON eleves.ideleve = lien_eleve_qcm.ideleve
WHERE idqcm = 4
```

b. Le résultat de cette requête sera :

| nom | prenom |
|--------|--------|
| Marty | Mael |
| Bikila | Abebe |

 Note

On a supposé l'élève Dubois Thomas ne figure plus dans la base suite à la requête de la question 2.d. Dans le cas contraire, il faudrait le rajouter au résultat précédent.

4.

 Requête SQL

```
SELECT eleves.nom, eleves.prenom, lien_eleve_qcm.note FROM eleves
JOIN lien_eleve_qcm ON eleves.ideleve = lien_eleve_qcm.ideleve
JOIN qcm ON qcm.idqcm = lien_eleve_qcm.idqcm
WHERE qcm.titre = "Arbre Binaire"
```

4. Exercice 4

algorithmique (arbres binaires en profondeurs préfixe et infixé)

1. a. Un arbre binaire est un arbre d'arité 2, c'est à dire un arbre dans lequel chaque noeud possède au plus deux fils. C'est bien le cas ici, une personne ayant au maximum deux parents connus.
 - b. Dans un arbre binaire de recherche, on dispose d'une relation d'ordre entre les clés associées à chaque noeud et pour tout noeud, sa clé est supérieure aux clés du sous arbre gauche et inférieure aux clés du sous arbre droit. Ici les clés sont des personnes sur lesquelles on n'a pas de relation d'ordre.
 2. a. On rappelle que dans un parcours en *profondeur préfixe*, on liste en premier la racine puis récursivement les clés du sous arbre gauche et du sous arbre droit. Ce qui donne ici :
- Albert Normand — Jules Normand — Michel Normand — Jules Normand — Odile Picard — Hélène Breton — Evariste Breton
- b. Dans le parcours en *profondeur infixé*, on liste récursivement les clés du SAG puis la racine puis les clés du SAD. Ce qui donne ici : Jules Normand — Michel Normand — Odile Picard — Jules Normand — Evariste Breton — Hélène Breton — Camélia Charentais
 - c. En parcours **prefixe** on insère l'affichage du tuple `(prenom, nom)` **avant** de relancer les parcours récursifs sur les deux sous arbres.

 Script Python

```
def parcours(racine_de_l_arbre) :
    if racine_de_l_arbre != None :
```

```

noeud_actuel = racine_de_l_arbre
print(noeud_actuel.identite)
parcours(noeud_actuel.gauche)
parcours(noeud_actuel.droite)

```

- d. En parcours **infixe** on insère l'affichage du tuple `(prenom, nom)` entre les parcours récursifs sur les deux sous arbres.

Script Python

```

def parcours(racine_de_l_arbre) :
    if racine_de_l_arbre != None :
        noeud_actuel = racine_de_l_arbre
        parcours(noeud_actuel.gauche)
        print(noeud_actuel.identite)
        parcours(noeud_actuel.droite)

```

3. a.

Script Python

```

class Noeud() :
    def __init__(self, prenom, nom) :
        self.identite = (prenom, nom)
        self.gauche = None
        self.droite = None
        self.génération = 0

```

Bug

Dans l'énoncé, `self` ne figure pas dans les paramètres de `__init__` (ajouté dans cette correction)

b.

Script Python

```

def numerotation(racine_de_l_arbre, num_gen=0) :
    if racine_de_l_arbre != None:
        racine_de_l_arbre.génération = num_gen
        numerotation(racine_de_l_arbre.gauche, num_gen+1)
        numerotation(racine_de_l_arbre.droite, num_gen+1)

```

4. Cette fonction parcourt l'arbre en préfixe mais affiche seulement les noeuds droit, ce qui donne : Odile Picard — Hélène Breton — Camélia Charentais — Marie Comtois — Eulalie Lorrain — Gabrielle Savoyard — Janet Chesterfield

5. Exercice 5

réseau, protocoles de routage, langage et programmation

1. a. Un adresse IPv4 se compose de 4 octects.
 b. Le PC3 a pour adresse IPv4 : 172.150.4.30/24 le masque de sous réseau est donc
 $11111111.11111111.11111111.00000000$ c'est à dire 255.255.255.0

2. Tableau complété :

| Adresse IPv4 du PC3 | Ligne 1 | 172 | 150 | 4 | 30 |
|--|---------|-----------------|-----------------|-----------------|-----------------|
| Masque de sous réseau | Ligne 3 | 1 0 1 0 1 1 0 0 | 1 0 0 1 0 1 1 0 | 0 0 0 0 0 1 0 0 | 0 0 0 1 1 1 1 0 |
| Adresse du réseau | Ligne 4 | 1 0 1 0 1 1 0 0 | 1 0 0 1 0 1 1 0 | 0 0 0 0 0 1 0 0 | 0 0 0 0 0 0 0 0 |
| Pour obtenir l'adresse réseau binaire, on réalise un ET (&) logique entre chaque bit de l'adresse IP (ligne 2) et du masque de sous réseau (ligne 3) | | | | | |
| Adresse du réseau | Ligne 5 | 172 | 150 | 4 | 30 |

 Aide

- Exemple de la conversion binaire décimal : $150 = \boxed{1|0|0|1|0|1|1|0}$
- On rappelle qu'un & logique vaut 1 uniquement lorsque les deux entrées valent 1.

3. a. L'adresse 172.150.10.257 n'est pas valide (le dernier chiffre n'est pas entre 0 et 255). L'adresse 172.154.4.30 ne fait pas partie du réseau (ne commence pas par 172.150.4) L'adresse 172.150.4.0 est celle du réseau. Et enfin, * 172.150.4.10 est déjà utilisée. Pour un nouvelle ordinateur on peut donc utiliser :
- 172.150.4.11
 - 172.150.4.200
- b. Pour connaître l'adresse IP, on peut utiliser la commande `ifconfig` (système Linux) ou `ipconfig` (Windows).
4. Les machines sont sur des réseaux différents (172.16.1.10\16 d'un côté et 192.168.5.10\16 de l'autre) un switch ne permet donc pas de les relier. Pour que cela fonctionne, il faudrait changer la configuration de toutes les machines d'un des sous réseau. L'alternative est d'utiliser un *routeur* qui permet d'interconnecter les deux sous réseau en conservant leur configuration.

5.  Script Python

```
def adresse(adresse, liste_ip):
    if adresse not in liste_ip:
        liste_ip.append(adresse)
        print("pas trouvée, ajoutée")
    else:
        print("trouvée")
```