Proposition de correction

Exercice 1

Q1.a

1	4	3	8	2

Q1.b

5
8
6
2

Q1.c

8	5	1	4	3	8	2

6
2

Q1.d

1	4	3

8
2
5
8
6
2

Q2

file:

1	2	3	4
1	2	3	
1	2		
1			
1			
2	1		
3	2	1	
4	3	2	1

La pile est vide

Q3.a

f	2,1,3	2,1	2	3,2	3,2	3	3	2, 3	1,2,3			
p	⌊⌋	⌊3⌋	⌊3⌋	⌊⌋	⌊1⌋	⌊1⌋	⌊2 1⌋	⌊1⌋	⌊⌋			

Q3.b

Tri ascendant de la liste

Exercice 2

Q1.a

```
def donnePremierIndiceLibre(Mousse):
    """
    Mousse est une liste.
    La fonction doit renvoyer l'indice du premier
    emplacement libre (contenant None) dans la liste Mousse
    ou renvoyer 6 en l'absence d'un emplacement libre dans Mousse.
    """
    i = 0
    while i < 6 and Mousse[i] != None :
            i += 1
    return i
```

Q1.b

```
def placeBulle(B : object) :
    """ reçoit en paramètre un objet de type Cbulle
```

```
    et place cet objet dans la liste Mousse """
    place = donnePremierIndiceLibre(Mousse)
    if place < len(Mousse) :
            Mousse[place] = B
    else :
            Mousse.append(B)
```

Q2

```
def bullesEnContact(B1 : object, B2 : object) -> bool :
    """ renvoie True si B1 et B2 en contact, False sinon """
    return distanceEntreBulles(B1, B2) <= B1.rayon + B2.rayon
```

Q3

```
def collision(indPetite, indGrosse, mousse) :
    """
    Absorption de la plus petite bulle d'indice indPetite
    par la plus grosse bulle d'indice indGrosse. Aucun test
    n'est réalisé sur les positions.
    """
    # calcul du nouveau rayon de la grosse bulle
    surfPetite = pi*Mousse [indPetite].rayon**2
    surfGrosse = pi*Mousse [indGrosse].rayon**2
    surfGrosseApresCollision =  surfGrosse + surfPetite
    rayonGrosseApresCollision = sqrt(surfGrosseApresCollision/pi)
    #réduction de 50% de la vitesse de la grosse bulle
    Mousse[indGrosse].dirx =  Mousse[indGrosse].dirx // 2
    Mousse [indGrosse].diry =  Mousse [indGrosse].diry // 2
    #suppression de la petite bulle dans Mousse
    Mousse [indPetite] = None
```

Exercice 3

Q1.a

affiche les titres des QCM créés antérieurement au 10 janvier 2022.

- POO
- Arbre Parcours

Q1.b

SELECT note

FROM lien_eleve_qcm

WHERE ideleve = 4

ORDER BY note DESC

Q2.a

la clef primaire doit être unique

Q2.b

- Lors de la 1ère question, un enregistrement est créé dans la table lien_eleve_qcm (INSERT).
- Puis à chaque réponse de l'élève la valeur de l'attribut note est modifiée (UPDATE).

Q2.c

INSERT INTO eleves

VALUES(6, "Lefèvre", "Kevin")

Q2.d

DELETE FROM lien_eleve_qcm WHERE ideleve = 2 ;

DELETE FROM eleves WHERE ideleve = 2 ; -- toutes les références à cet élève doivent être supprimées

Q3.a

SELECT noms, prenom FROM eleves

JOIN lien_eleve_qcm ON eleves.ideleve = lien_eleve_qcm.ideleve

WHERE idqcm = 4 ;

Q3.b

Dubois Thomas

Marty Mael

Bikila Abebe

Q4

SELECT eleves.nom, eleves.prenom, lien_eleve_qcm.note

FROM eleves, lien_eleve_qcm, qcm

WHERE qcm.titre = "Arbre Binaire"

AND lien_eleve_qcm.idqcm = qcm.idqcm AND eleves.ideleve = lien_eleve_qcm.ideleve

ORDER BY eleves.nom, eleves.prenom

EXERCICE 4

Q1.a

chaque élément possède au plus deux éléments fils au niveau inférieur

Q1.b

il n'y a pas de notion de relation d'ordre

Q2.a

Albert Normand, Jules Normand, Michel Normand, Jules Normand, Odile Picard, Hélène Breton, Evariste Breton

Q2.b

Jules Normand, Michel Normand, Odile Picard, Jules Normand, Evariste Breton, Hélène Breton, Camélia Charentais

Q2.c

```
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)
```

Q2.d

```
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)
```

Q3.a

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

Q3.b

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

Q4

parcours préfixe qui affiche le prénom (identite[0] !) de la mère

Odile, Hélène, Camélia, Marie, Eulalie, Gabrielle, Janet

Exercice 5

Q1.a

4

Q1.b

255.255.255.0

Q2

Adresse IP (V4) du PC3	Ligne 1	172								150								4								30					
	Ligne 2	1	0	1	0	1	1	0	0	1	0	0	1	0	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0
Masque de sous réseau	Ligne 3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	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 (ligne3)																															
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
	Ligne 5	172								150								4								0					

Q3.a

1) 172.154.4.30 : hors réseau
2) 172.150.4.10 : déjà pris
3) 172.150.10.257 : mauvaise @
4) 172.150.4.11 : ok

5) 172.150.4.0 : @ réseau
6) 172.150.4.200 : ok

Q3.b

ifconfig (unix), ipconfig (DOS)

Q4

ce ne sont pas les même réseaux, il faut un routeur pour les relier

Q5

```
def adresse(ip : list, liste_IP : list) :
    """ pré : ip -- liste de 4 entiers [0-255] """
    if ip in list :
            print("trouvée")
    else :
            print("pas trouvée, ajoutée")
            liste_IP.append(ip)
```