Proposition de correction

Exercice 1

Q1

Il n'est pas possible de connaître les valeurs des clés étrangères idPiece et idActeur.

Q2

INSERT INTO Role(46721, 389761, 'Tartuffe')

Q3

Met à jour l'attribut Langue avec la valeur 'Anglais' dans les cas où cet attribut a les valeurs 'Américain' ou 'Britannique'

Q4a

SELECT nom, prenom FROM Acteur WHERE anneeNaiss > 1990

Q4b

SELECT anneeNaiss FROM Acteur ORDER BY anneeNaiss ASC LIMIT 1

Q4c

SELECT Role.nomRole
FROM Role, Acteur
WHERE Acteur.prenom = 'Vincent' AND Acteur.nom = 'Macaigne' AND Role.idActeur = Acteur.idActeur

Q4d

SELECT Piece.titre
FROM Piece, Acteur, Role
WHERE Piece.langue = 'Russe' AND Acteur.prenom = 'Jeanne' AND Acteur.nom = 'Balibar' AND Role.idActeur = Acteur.idActeur AND Role.idPiece = Piece.idPiece

Exercice 2

Q1a

```
class Pile:
    def __init__(self):
        """Initialise la pile comme une pile vide."""
        self.pile = []

    def est_vide(self):
        """Renvoie True si la liste est vide, False sinon."""
```

```
        return self.nb_elements() == 0

    def empiler(self, e):
        """Ajoute l'élément e sur le sommet de la pile, ne renvoie rien."""
        self.pile.append(e)

    def depiler(self):
        """Retire l'élément au sommet de la pile et le renvoie."""
        return self.pile.pop()

    def nb_elements(self):
        """Renvoie le nombre d'éléments de la pile. """
        return len(self.pile)

    def afficher(self):
        """Affiche de gauche à droite les éléments de la pile, du fond
        de la pile vers son sommet. Le sommet est alors l'élément
        affiché le plus à droite. Les éléments sont séparés par une
        virgule. Si la pile est vide la méthode affiche « pile vide »."""
        taille = self.nb_elements()
        if taille != 0:
            for i in range(taille):
                if i < taille-1:
                    print(self.pile[i], end=',')
                else:
                    print(self.pile[i])
        else:
            print('pile vide')
```

Q1b

>>> 7,5,5,2

Q2a

Cas n°1 : >>> 3,2

Cas n°2 : >>> 3,2,5,7

Cas n°3 : >>> 3

Cas n°4 : >>> pile vide

Q2b

Une pile inversée limitée à l'élément passé en paramètre

Q3

```
def etendre(pile1 : object, pile2 : object):
    """ modifie pile1 en lui ajoutant les éléments de pile2 rangés dans l'ordre inverse. """
    while not pile2.est_vide():
```

```
        pile1.empiler(pile2.depiler())
```

Q4

```
def supprime_toutes_occurences(pile : object, element : int):
    """ supprime tous les éléments element de pile. """
    p = Pile()
    while not pile.est_vide():
        e = pile.depiler()
        if e != element:
            p.empiler(e)
    while not p.est_vide():
        pile.empiler(p.depiler())
```

Exercice 3

Partie A

Q1

init

Q2

- pid 5440 : python (Running)
- pid 5450 : bash (Running)

Q3

- bash (ppid 1912)
- python, bash, bash

Q4

- bash → python programme1.py
- bash → bash → python programme1.py

Q5

Non. python programme2.py est actif mais il peut devenir dormant (eg : attente de ressource)

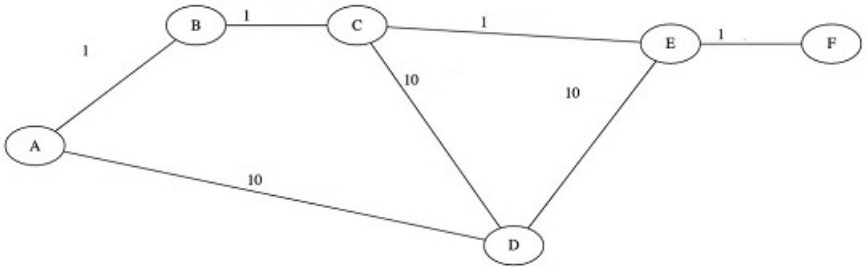
Partie B

Q1

Machine	Prochain saut	Distance
A	D	3
B	C	3
C	E	2
D	E	2

E	F	1

Q2



Machine	Prochain saut	Distance
A	B	4
B	C	2
C	E	2
D	E	11
E	F	1

Q3

OSPF car le chemin emprunté possède un débit de 100 Mb/s sur toute la ligne, alors que RIP emprunte une route qui limite le débit à 10 Mb/s.

Exercice 4

Partie A

Q1

lab2[1][0] = 2

Q2

```
def est_valide(i : int, j : int, n : int, m : int) -> bool:
    """ renvoie True si le couple (i, j) correspond à des coordonnées valides
    pour un labyrinthe de taille (n, m), et False sinon."""
    return -1 < i < n and -1 < j < m
```

Q3

```
def depart(lab : list) -> tuple:
    """ renvoie les coordonnées du départ d'un labyrinthe """
    n = len(lab)
    m = len(lab[0])
    for i in range(n):
        for j in range(m):
            if lab[i][j] == 2:
                return i,j
    return None
```

Q4

```
def nb_cases_vides(lab : list) -> int:
    """ renvoie le nombre de cases vides d'un
    labyrinthe (comprenant donc l'arrivée et le départ)."""
    total = 0

    if len(lab):
        for i in range(len(lab)):
            for j in range(len(lab[0])):
                if lab[i][j] in [0,2,3]:
                    total += 1
    return total
```

Partie B

Q1

[(1,1), (2,2)]

Q2a

```
# entrée: (1, 0), sortie (1, 5)
chemin = [(1, 0)]
chemin.append((1, 1))
chemin.append((2, 1))
chemin.pop()
chemin.append((1, 2))
chemin.append((1, 3))
chemin.append((2, 3))
chemin.append((3, 3))
chemin.append((3,4))
chemin.pop()
chemin.pop()
chemin.pop()
chemin.append((1, 4))
chemin.append((1, 5))
```

Q2b

```
def solution(lab : list) -> list:
    """ renvoie la solution du labyrinthe."""
    chemin = [depart(lab)]  #ajout des coordonnées de la case départ à la liste chemin

    case = chemin[0]
    i, j     = case[0], case[1]
    while lab[i][j] != 3:   #Tant que l'arrivée n'a pas été atteinte...
        lab[i][j] = 4           #on marque la case visitée avec la valeur 4
        cases = voisines(i, j, lab)
        if len(cases):  #si la case visitée possède une case voisine libre...
            case = cases[0]             #la première case de la liste renvoyée par la fonction voisines
            i, j     = case[0], case[1]   #devient la prochaine case à visiter
```

```
            chemin.append(case)     #et on ajoute à la liste chemin
        else:          #sinon, il s'agit d'une impasse...
            chemin.pop()                    #On supprime alors la dernière case dans la liste
            i, j = chemin[len(chemin)-1]    #La prochaine case à visiter
                                            #est celle qui est en dernière position de chemin

    return chemin
```

Exercice 5

Q1

8 < 7

Q2

3 < 7

Partie A

Q1a

- Cas n°1 : 0
- Cas n°2 : 1
- Cas n°3 : 2

Q1b

Détermine le nombre de valeurs < à la valeur de l'indice i dans le vecteur supérieur à l'indice i.

Q2

```
def nombre_inversions(tab : list) -> int:
    """ renvoie le nombre d'inversions du tableau. """
    total = 0
    for i in range(len(tab)):
        total += fonction1(tab, i)
    return total
```

Q3

quadratique (2 boucle for imbriquées)

Partie B

Q1

Tri par fusion ~ $O(n\ln_2 n)$

Q2

```
def moitie_gauche(tab : list) -> list:
    """ renvoie un nouveau tableau contenant la moitié gauche de tab.
    Si le nombre d'éléments de tab est impair, l'élément du centre est inclus. """
    gauche = []
```

```
    milieu = len(tab) // 2
    if len(tab) % 2:
        milieu += 1
    for i in range(milieu):
        gauche.append(tab[i])
    return gauche
```

Q3

```
def nb_inversions_rec(tab : list, n : int = 0) -> int:
    """ renvoie le même nombre que nombre_inversions(tab) de la partie A. """
    if len(tab) <= 1:
        return 0
    else:
        #Séparer le tableau en deux tableaux de tailles égales (à une unité près).
        gauche = moitie_gauche(tab)
        droite   = moitie_droite(tab)
        #Compter le nombre d'inversions dans chacun des deux tableaux.
        n = nb_inv_tab(sorted(gauche), sorted(droite))
        #Appeler récursivement la fonction nb_inversions_rec
        return n + nb_inversions_rec(gauche, n) + nb_inversions_rec(droite, n)
```