Éléments de correction sujet 04

Exercice 1

1

Si les relations Piece et Acteur sont vides, il ne sera pas possible de renseigner les attributs idPiece et idActteur de la relation, car Role.idPiece fait référence à la clé primaire Piece.idPiece de la relation Piece et Role.idActeur fait référence à la clé primaire Acteur.idActeur de la relation Acteur.

2

```
INSERT INTO Role
(idPiece, idActeur, nomRole)
VALUES
(46721, 389761, 'Tartuffe')
```

3

Cette requête remplace ‘Américain’ et ‘Britanique’ par ‘Anglais’ pour toutes les entrées concernées de la table Piece

4a

```
SELECT nom, prenom
FROM Acteur
WHERE anneeNaiss > 1990
```

4b

```
SELECT MAX(anneeNaiss)
FROM Acteur
```

4c

```
SELECT nomRole
FROM Role
INNER JOIN Acteur ON Role.idActeur = Acteur.idActeur
WHERE Acteur.nom = 'Macaigne' AND Acteur.prenom = 'Vincent'
```

4d

```
SELECT Piece.titre
FROM Role
INNER JOIN Piece ON Role.idPiece = Piece.idPiece
INNER JOIN Acteur ON Role.idActeur = Acteur.idActeur
WHERE Piece.langue = 'Russe' AND Acteur.nom = 'Balibar' AND
Acteur.prenom = 'Jeanne'
```

Exercice 2

1a

```
pile1 = Pile()
pile1.empiler(7)
pile1.empiler(5)
pile1.empiler(2)
```

1b

7, 5, 5, 2

2a

cas n°1 : 3, 2

cas n°2 : 3, 2, 5, 7

cas n°3 : 3

cas n°4 : pile vide

2b

La fonction mystere renvoie une pile qui contiendra tous les éléments de la pile passée en paramètre (*pile*) à condition qu'ils soient situés au-dessus de l'élément passé en paramètre (*element*). L'élément *element* sera lui aussi présent dans la pile renvoyée par la fonction.

3

```
def etendre(pile1, pile2):
    while not pile2.est_vide():
        x = pile2.depiler()
        pile1.empiler(x)
```

4

```
def supprime_toutes_occurences(pile, element):
    p2 = Pile()
    while not pile.est_vide():
        x = pile.depiler()
        if x != element:
            p2.empiler(x)
    while not p2.est_vide():
        x = p2.depiler()
        pile.empiler(x)
```

Exercice 3

PARTIE A

1

La première commande exécutée par le système d'exploitation lors du démarrage est la commande **init**

2

Les processus actifs sont les processus ayant pour PID 5440 et 5450 (présence de l'indicateur R dans la colonne STAT pour ces 2 processus).

3

La commande ps a été exécutée depuis l'application Bash (car le processus ps a pour PPID 1912 qui correspond au PID de Bash).
Deux autres processus Bash (PID 2014 et PID 2013) et un processus python programme1.py (PID 5437) ont été lancés depuis le processus Bash de PID 1912.

4

Le processus python programme1.py a un PID de 5437 alors que le processus python programme2.py a un PID de 5440. python programme1.py a été exécuté avant python programme2.py.

5

Non, aucune prédiction n'est possible.

PARTIE B

1

Machine	Prochain saut	Distance
A	D	3
B	C	3
C	E	2
D	E	2
E	F	1

2

coût A-D = 10
coût A-B = 1
coût B-C = 1
coût C-D = 10
coût C-E = 1
coût D-E = 10
coût E-F = 1

Machine	Prochain saut	Distance
A	B	4
B	C	3
C	E	2
D	E	11
E	F	1

3

RIP ne tient pas compte du débit alors que OSPF en tient compte. OSPF sera donc le plus performant.
coût A-D = 10

Exercice 4

Partie A

1

```
lab2[1][0] =  2
```

2

```
def est_valide(i,j,n,m):
    return i>=0 and j>=0 and i<n and j<m
```

3

```
def depart(lab):
    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)
```

4

```
def nb_cases_vides(lab):
    n = len(lab)
    m = len(lab[0])
    compt = 0
    for i in range(n):
        for j in range(m):
            if lab[i][j]==2 or lab[i][j]==3 or lab[i][j]==0:
                compt = compt + 1
    return compt
```

Partie B

1

L'appel de la fonction renvoie : `[(2, 2), (1, 1)]`

2a

```
# 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))
```

2b

```
def solution(lab):
    chemin = [depart(lab)]
    case = chemin[0]
    i = case[0]
    j = case[1]
    while lab[i][j] != 3:
        lab[i][j]=4
        v = voisines(i,j,lab)
        if len(v) != 0 :
            prochaine = v.pop()
            chemin.append(prochaine)
            i = prochaine[0]
            j = prochaine[1]
        else :
            chemin.pop()
            n = len(chemin)
            i = chemin[n-1][0]
            j = chemin[n-1][1]
    return chemin
```

Exercice 5

1
À l'indice 1 du tableau on trouve 8, à l'indice 3 on trouve 7.
Nous avons 1 < 3 alors que 8 > 7, nous avons donc bien une inversion

2
À l'indice 2 du tableau on trouve 3, à l'indice 3 on trouve 7.
Nous avons 2 < 3 et 3 < 7, nous n'avons donc pas d'inversion

PARTIE A

1a

cas n°1 : 0
cas n°2 : 1
cas n°3 : 2

1b

Si on considère l'élément *b* situé à l'indice *i* dans le tableau tab. La fonction *fonction1* permet de déterminer le nombre d'éléments plus grands que *b* situés dans le tableau à un indice supérieur à *i*.

2

```
def nombre_inversions(tab):
    nb_inv = 0
    n = len(tab)
    for i in range(n-1):
        nb_inv = nb_inv + fonction1(tab, i)
    return nb_inv
```

3

L'ordre de grandeur de la complexité en temps de l'algorithme est $O(n^2)$

Partie B

1

Le tri fusion a une complexité en $O(n.\log_2(n))$

2

```
def moitie_gauche(tab):
    n = len(tab)
    nvx_tab = []
    if n==0:
        return []
    mil = n//2
    if n%2 == 0:
        lim = mil
    else :
        lim =mil+1
    for i in range(lim):
        nvx_tab.append(tab[i])
    return nvx_tab
```

une autre possibilité un peu plus concise :

```
def moitie_gauche(tab):
    return [tab[i] for i in range(len(tab)//2+len(tab)%2)]
```

3

```
def nb_inversions_rec(tab):
    if len(tab) >  1:
        tab_g = moitie_gauche(tab)
        tab_d = moitie_droite(tab)
        return nb_inv_tab(tri(tab_g), tri(tab_d)) +
nb_inversions_rec(tab_g) + nb_inversions_rec(tab_d)
    else:
        return 0
```