Exercice 1

1.a


```
F = creer_file_vide()
enfiler(F,"jaune")
enfiler(F,"rouge")
enfiler(F,"jaune")
enfiler(F,"vert")
enfiler(F,"rouge")

P = creer_pile_vide()
while not(est_file_vide(F)):
    empiler(P,defiler(F))
print(P)
```

1.b

```
F = creer_file_vide()
enfiler(F,"jaune")
enfiler(F,"rouge")
enfiler(F,"jaune")
enfiler(F,"vert")
enfiler(F,"rouge")

def taille_file(F):
    t = 0
    ft = creer_file_vide()
    while not est_file_vide(F):
        t = t + 1
        enfiler(ft, defiler(F))
    while not est_file_vide(ft):
        enfiler(F, defiler(ft))
    return t

print("taille file F : ",taille_file(F))
```

Question 2

```
def former_pile(F):
    p = creer_pile_vide()
    pt = creer_pile_vide()
    while not est_file_vide(F):
        empiler(pt,defiler(F))
    while not est_pile_vide(pt):
        empiler(p,depiler(pt))
    return p

F = creer_file_vide()
enfiler(F,"jaune")
enfiler(F,"rouge")
enfiler(F,"jaune")
enfiler(F,"vert")
enfiler(F,"rouge")
print(former_pile(F))
```

Question 3

```
def nb_elements(F, ele):
    nb = 0
    ft = creer_file_vide()
    while not est_file_vide(F):
        x = defiler(F)
        if x==ele:
            nb = nb + 1
        enfiler(ft, x)
    while not est_file_vide(ft):
        enfiler(F, defiler(ft))
    return nb

F = creer_file_vide()
enfiler(F,"jaune")
enfiler(F,"rouge")
enfiler(F,"jaune")
enfiler(F,"vert")
enfiler(F,"rouge")
print(nb_elements(F,"jaune"))
print(nb_elements(F,"rouge"))
print(nb_elements(F,"vert"))
```

Question 4

```
def verifier_contenu(F, nb_rouge, nb_vert, nb_jaune):
    return nb_elements(F, "rouge") <= nb_rouge and nb_elements(F,"vert") <=
nb_vert and nb_elements(F, "jaune") <= nb_jaune

print(verifier_contenu(F,2,1,2))
print(verifier_contenu(F,2,2,1))
```

Exercice 2

1.

Prenons un exemple où au départ on a : `lst[i1] = 3` et `lst[2] = 8`
Après la ligne `lst[i2] = lst[i1]`, nous avons `lst[i2] = 3`
Après la ligne `lst[i1] = lst[i2]`, nous avons `lst[i1] = 3`
Le résultat attendu était `lst[i1]` = 8 et `lst[2] = 3`, le résultat obtenu est `lst[i1] = 3` et `lst[2] = 3`, le code Python proposé ne réalise pas l'échange attendu.

Il faut utiliser une variable temporaire pour que cela fonctionne :

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

Ou bien comme ceci :

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

2 Les valeurs qui pourront être renvoyées par `randint(0, 10)` sont : 0, 1, 9 et 10.

3.a

Nous avons un appel récursif avec `melange(lst, ind-1)`. À chaque appel récursif on soustrait 1 au paramètre `ind`. Au bout d'un certain nombre d'appels récursifs, le paramètre sera égal à 0, les instructions "contenues" dans le "`if`" (`if ind>0`) ne seront plus exécutées et le programme s'arrêtera.

3.b

Pour l'appel initial de la fonction nous avons `ind = n-1`. Pour le premier appel récursif nous avons `ind = n-2`. Pour le dernier appel récursif nous avons `ind = 0`, nous avons donc eu `n-1` appels récursifs.

3.c

```
[0, 1, 2, 3, 4]
[0, 1, 4, 3, 2] j = 2
[0, 3, 4, 1, 2] j = 1
[0, 3, 4, 1, 2] j = 2
[3, 0, 4, 1, 2] j = 0
```

3d

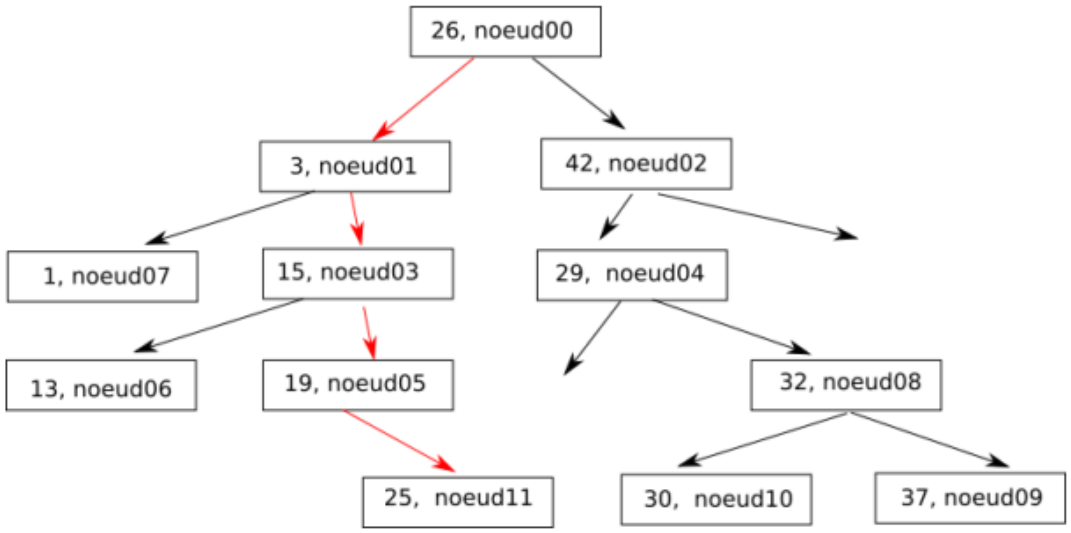
```
def melange(lst):
      ind = len(lst)-1
      while ind > 0 :
```

```
            j = randint(0, ind)
            echange (lst, ind, j)
            ind = ind - 1
```

Exercice 3 :

1.

1



On désire insérer le noeud11 (valeur 25). On part de la racine (noeud00 de valeur 26), 25 est plus petit que 26, on considère donc le sous-arbre gauche et on se retrouve au niveau du noeud01 (valeur 3). 25 est plus grand que 3, on considère donc le sous-arbre droit au noeud01 et on se retrouve au niveau du noeud03 (valeur 15). 25 est plus grand que 15, on considère donc le sous-arbre droit au noeud03 et on se retrouve au niveau du noeud05 (valeur 19). 25 est plus grand que 19, on considère donc le sous-arbre droit du noeud05, ce sous-arbre droit est vide et on insère donc le noeud11 à cet emplacement. Le noeud11 est donc inséré sous le noeud5 en fils droit.

2

Il est possible de stocker toutes les valeurs comprises entre 26 et 29, c'est à dire : 27 et 28

3a

26 - 3 - 1 - 15 - 13 - 19 - 25 - 42 - 29 - 32 - 30 - 37

3b C'est un parcours préfixe.

4

```
Parcours2(A) :
      Parcours2(A.fils_gauche)
      Afficher(A.valeur)
      Parcours2(A.fils_droit)
```

Exercice 4 :

1

La première et la troisième requête utilisent toutes les deux la même valeur pour l'attribut *idEleves* (128). L'attribut *idEleve* étant une clé primaire, nous allons donc avoir une erreur(on ne doit pas trouver dans toute la relation 2 fois la même valeur pour une clé primaire)

2

Dans la relation Emprunts l'attribut *idEleve* est une clé étrangère, c'est ce qui assure que l'on ne pourra pas enregistrer un emprunt pour un élève qui n'a pas encore été inscrit dans la relation Eleves.

3

```
SELECT
titreFROM
Livres
WHERE auteur = 'Molière'
```

4

Cette requête permet d'avoir le nombre d'élèves de la classe T2 inscrits au CDI.

5

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

6

Cette requête permet d'avoir le nom et le prénom de tous les élèves de la classe T2 qui ont déjà emprunté un livre au CDI.

7

```
SELECT nom,
prenomFROM
Emprunts
```

```
INNER JOIN Livres ON Livres.isbn = Emprunts.isbn
INNER JOIN Eleves ON Eleves.idEleve =
Emprunts.idEleveWHERE titre = 'Les misérables'
```

Exercice 5 :

1.1

A -> C -> F -> G

1.2

Destination	Routeur suivant	Distance
A	F	3
B	E	3
C	F	2
D	E	2
E	E	1
F	F	1

2

Destination	Routeur suivant	Distance
B	B	1
D	D	1
E	D	2
F	D	4
G	D	3

3.1

A -> B : 10 Gb/s soit le coût = $\frac{10^8}{10\times10^9}$ = 0,01

3.2

$\frac{10^8}{d} = 5 \Rightarrow d = \frac{10^8}{5} = 2 \times 10^7 b/s = 20\ Mb/s$

4

On part de A (possible d'aller en B, en D ou en C), on trouve le débit le plus important auniveau de la liaison A->D. Une fois en D, on va vers E et une fois en E, on rejoint directement G (car le coût du chemin E -> C -> F -> G serait plus grand).

D'ôù le chemin A -> D -> E -> G avec un coût de :

$\frac{10^8}{10\times10^9} + \frac{10^8}{100\times10^9} + \frac{10^8}{100\times10^6} = 0{,}01 + 0{,}001 + 1 = 1{,}011$