

Éléments de correction sujet 06

Exercice 1

1

Résultat d'exécution :

D
A

2

```
def cryptage(self, texte):  
    c = ""  
    for l in texte:  
        c = c + self.decale(l)  
    return c
```

3

```
cle = input("saisir la clé de chiffrement : ")  
cle = int(cle)  
c = CodeCesar(cle)  
txt = input("saisir le texte à chiffrer : ")  
print("le message chiffré est : "+c.cryptage(txt))
```

4

La ligne `print(CodeCesar(10).transforme("PSX"))` va permettre d'afficher **FIN**

Exercice 2

1a

```
{'type': 'classique', 'etat': 1, 'station': 'Coliseum'}
```

1b

```
0
```

1c

renvoie une erreur, car la clé 99 n'existe pas dans le dictionnaire *flotte*

2a

le paramètre *choix* peut être égal à "electrique" ou "classique"

2b

Dans le cas où le paramètre *choix* est égal à "electrique", la fonction *proposition* renvoie "Prefecture" ou "Jacobins" selon la version de Python utilisée ! Dans le cas où le paramètre *choix* est égal à "classique", la fonction *proposition* renvoie "Baraban" ou "Coliseum" selon la version de Python utilisée !

3a

```
def affiche():
    tab = []
    for v in flotte:
        if flotte[v]["station"] == "Citadelle" and flotte[v]["etat"] == 1 :
            tab.append(v)
    print(tab)
```

3b

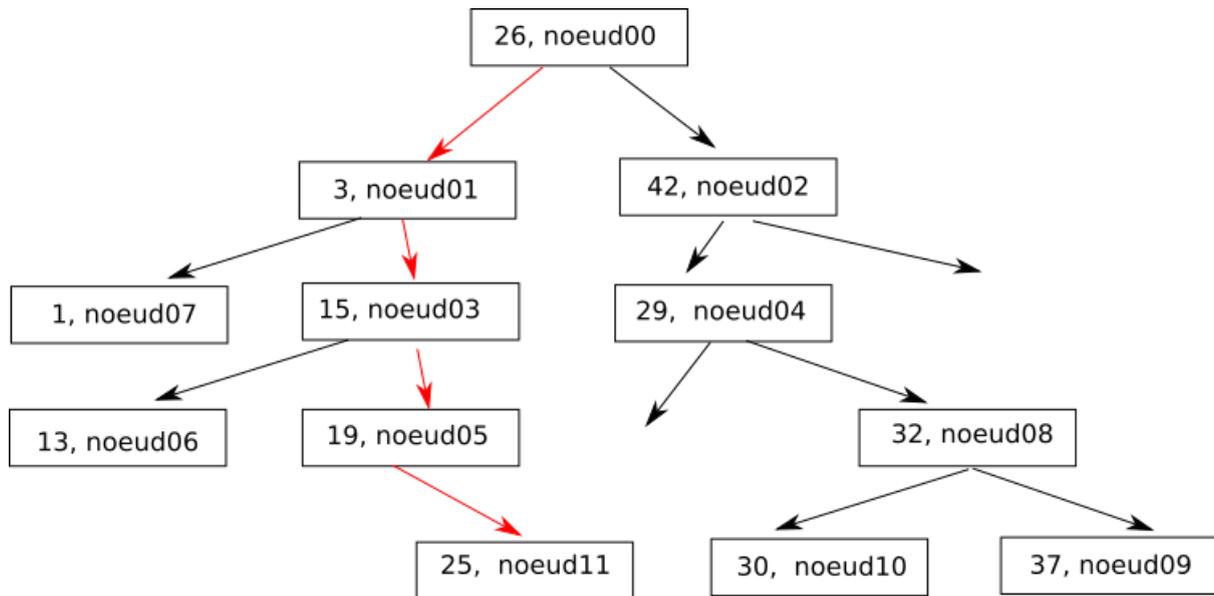
```
def affiche():
    tab = []
    for v in flotte:
        if flotte[v]["etat"] != -1 and flotte[v]["type"] == "electrique":
            tab.append((v,flotte[v]["station"]))
    print(tab)
```

4

```
def station(coord):
    d = {}
    for num,info in flotte.items() :
        nom_station = info['station']
        distance_station = distance(stations[nom_station],coord)
        if info['etat'] == 1 and distance_station < 800:
            if nom_station not in d :
                d[nom_station] = [distance_station, [num]]
            else :
                d[nom_station][1].append(num)
    return d
```

Exercice 3

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 : 26, 27 et 28 (on peut prendre 26 car il est précisé dans l'énoncé que "les valeurs de tous les nœuds du sous-arbre droit sont supérieures ou **égales** à la valeur du nœud X"

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)

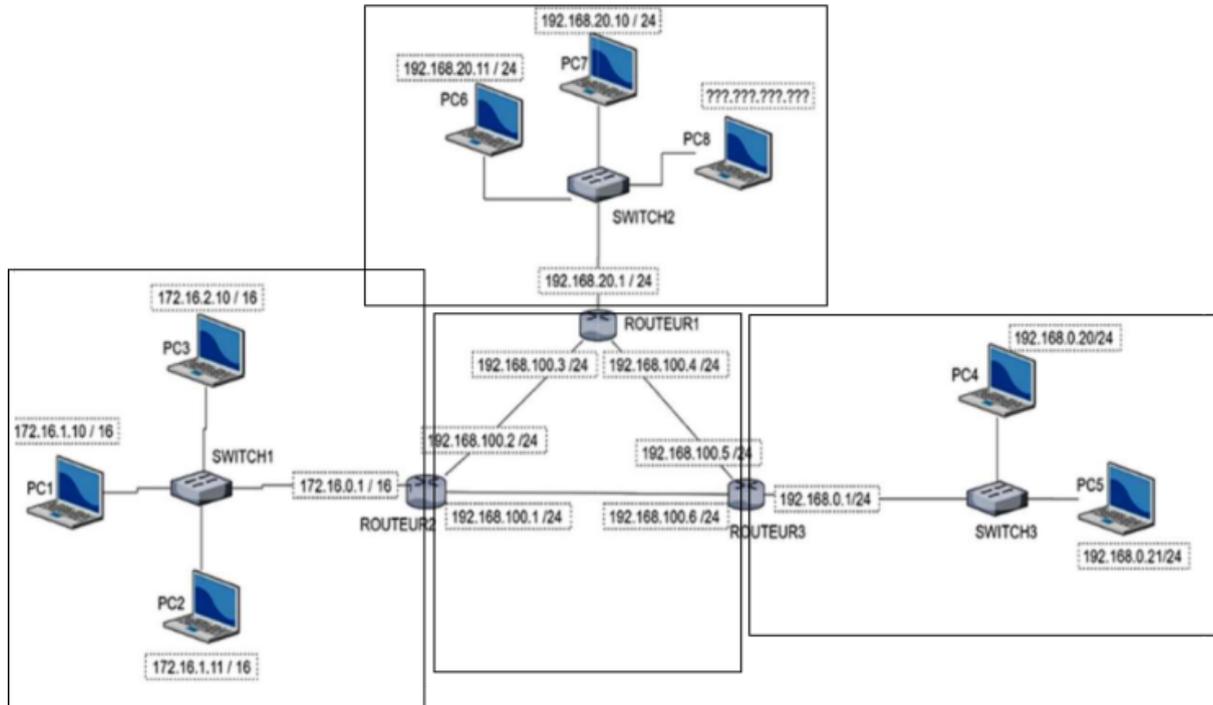
Afficher(A.valeur)

Parcours2(A.fils_droit)

Exercice 4

Partie A

1



2a

On utilise 4 octets dans une adresse IP V4

2b, 2c et 2d

Adresse IP (V4) du PC7	Ligne 1	192	168	20	10
	Ligne 2	1 1 0 0 0 0 0 0	1 0 1 0 1 0 0 0	0 0 0 1 0 1 0 0	0 0 0 0 1 0 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	1 0 0 0 0 0 0 0
	Ligne 4	255	255	255	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 5	1 1 0 0 0 0 0 0	1 0 1 0 1 0 0 0	0 0 0 1 0 1 0 0	0 0 0 0 0 0 0 0
	Ligne 6	192	168	20	0

3

Adresses IP possibles : 192.168.20.30 et 192.168.20.230,

Partie B

```
def IP_bin(adr):
    conv=[]
    for o in adr:
        conv.append(dec_bin(o))
    return conv
```

ou encore :

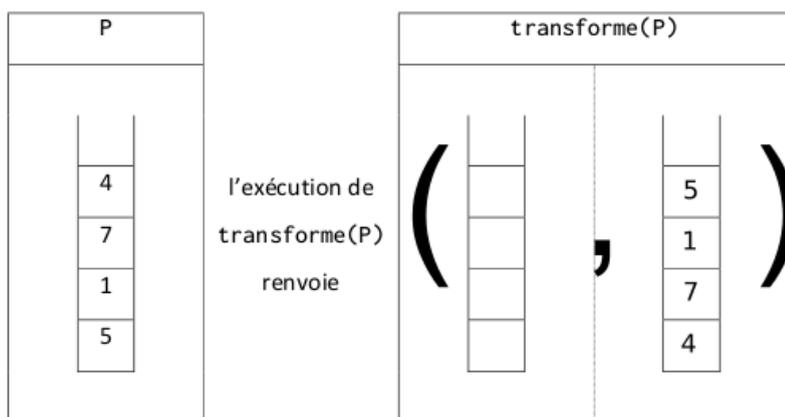
```
def IP_bin(adr):
    return [dec_bin(o) for o in adr]
```

Exercice 5

1

	Etape 0 Pile d'origine P	Etape 1 empiler(P,8)	Etape 2 depiler(P)	Etape 3 est_vider(P)
				
Retour de la fonction		None	8	False

2



3

```
def maximum(P):  
    m=depiler(P)  
    while not est_vide(P):  
        v = depiler(P)  
        if v > m:  
            m = v  
    return m
```

4a

Il suffit de mettre place une boucle qui s'arrêtera quand la pile P sera vide. À chaque tour de boucle, on dépile P, on empile les valeurs précédemment dépilées dans une pile auxiliaire Q et on incrémente un compteur de 1. Une fois la boucle terminée, on crée une nouvelle boucle où on dépile Q et on empile P avec les valeurs dépilées (l'idée est de retrouver l'état originel de pile. Il suffit ensuite de renvoyer la valeur du compteur.

4b

```
def taille(P):  
    cmp = 0  
    Q = creer_pile()  
    while not est_vide(P):  
        v = depiler(P)  
        empiler(Q,v)  
        cmp = cmp + 1  
    while not est_vide(Q):  
        v = depiler(Q)  
        empiler(P,v)  
    return cmp
```