Proposition de correction

Exercice 1

Q1

>>> D

>>> A

Q2

```
    def cryptage(self, texte : str) -> str:
        """ renvoie le message chiffré """
        crypte = ""
        for i in texte:
            crypte += self.decale(i)
        return crypte
```

Q3

```
try:
    #demande de saisir la clé de chiffrement
    cle = int(input("clé de chiffrement :"))
    if -26 < cle < 26:
        # crée un objet de classe CodeCesar
        code = CodeCesar(cle)
        # demande de saisir le texte à chiffrer
        texte = input("texte à chiffrer :")
        # affiche le texte chiffré en appelant la méthode cryptage
        print(code.cryptage(texte))
except ValueError:
    print("clé de chiffrement incorrect")
except KeyboardInterrupt:
    pass
```

Q4

Le programme instancie la classe César avec une clef de chiffrement (> 0) et appelle la méthode transforme().

Cette méthode appelle à son tour la méthode cryptage() écrite en Q2. On obtient alors :

P –> F

S –> I

X –> N

qui affiche le message « FIN »

Exercice 2

Q1a

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

Q1b

0

Q1c

Lève l'exception KeyError

Q2a

classique ou électrique

Q2b

La fonction renvoie le nom de la 1ère station trouvée pour un vélo électrique ou classique (choix) disponible (etat = 1)

Q3a

```
def trouve_identifiant(station : str) -> list:
    """ renvoie les identifiants de tous les vélos disponibles à une station"""
    id_velo = []
    for v in flotte:
        if flotte[v]["station"] == station and flotte[v]["etat"] == 1:
            id_velo.append( v )
    return id_velo

print( trouve_identifiant("Citadelle") )
```

Q3b

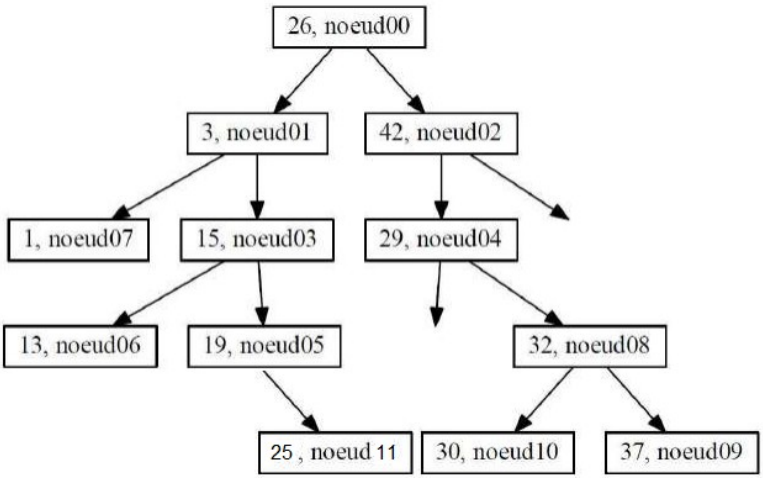
```
def trouve_velo() -> list:
    """ renvoie les identifiants de tous les vélos électrique qui ne sont pas en panne"""
    id_velo = []
    for v in flotte:
        if flotte[v]["type"] == 'electrique' and flotte[v]["etat"] != -1:
            id_velo.append( v )
    return id_velo
```

Q4

```
def proximite(coordonnees : tuple) -> list
    """ renvoie, pour chaque station située à moins de 800 mètres de l'utilisateur :
    o le nom de la station ;
    o la distance entre l'utilisateur et la station ;
    o les identifiants des vélos disponibles dans cette station.
    """
    infos = []
    for nom in stations:
        velos = trouve_identifiant(nom)
        d = distance(coordonnees, stations[nom])
        if d < 800 and len(velos):
            infos.append( (nom, d, velos) )
    return infos
```

Exercice 3

Q1



25 < 26 : insertion fils gauche

25 > 3 : insertion fils droit

25 > 15 : insertion fils droit

25 > 19 : insertion fils droit

Q2

26 < valeur ≤ 29, soit 26, 27 et 28 (on autorise les doublons)

Q3a

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

Q3b

Parcours préfixe

Q4

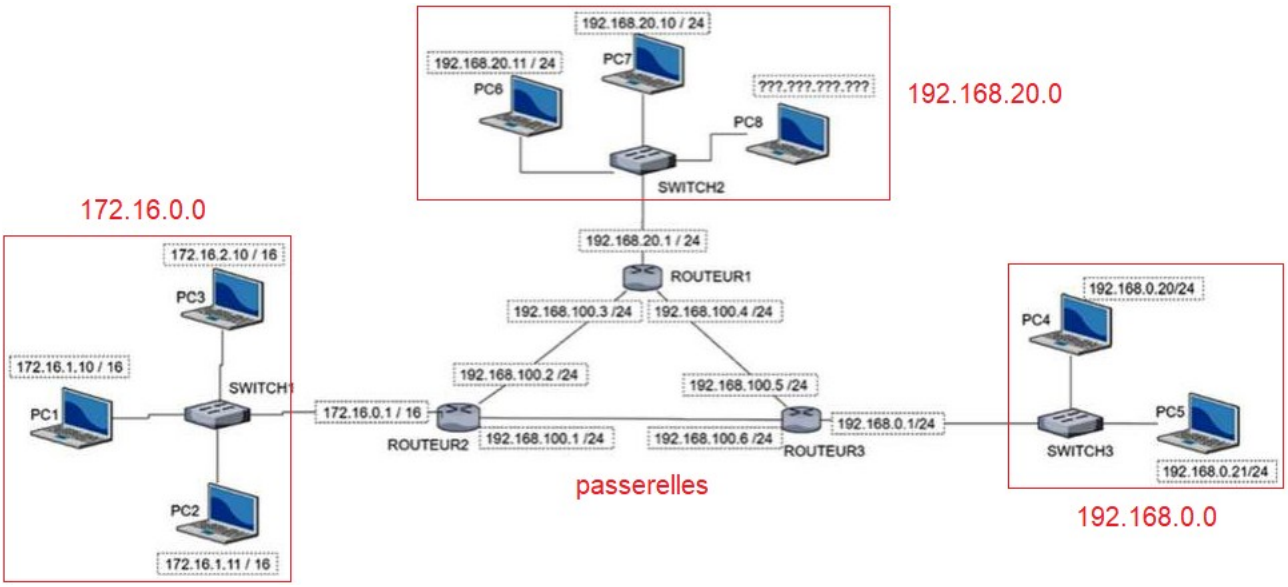
Parcours infixe :

Parcours(A) # A est un arbre binaire de recherche
    Parcours(A.fils_gauche)
    Afficher(A.valeur)
    Parcours(A.fils_droit)

Exercice 4

Partie A

Q1



Q2a

4

Q2b, Q2c, Q2d

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	0 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

Q3

☐ 192.168.20.0 @ réseau

☐ 192.256.20.11 @ pris par PC 6

X 192.168.20.30

X 192.168.20.230

☐ 192.168.20.260 @ hote < 255

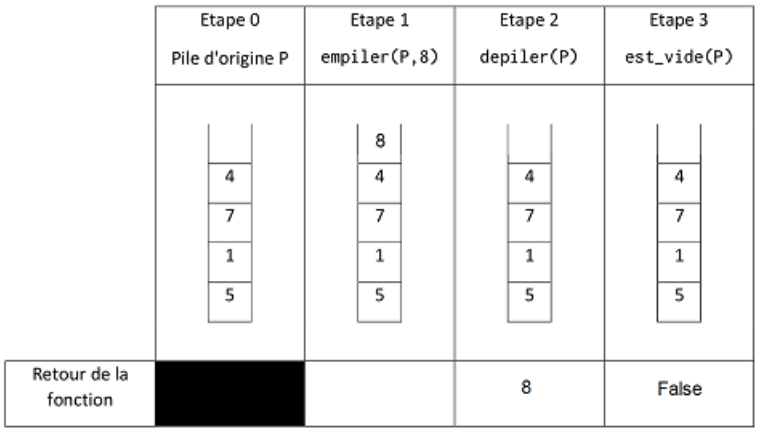
☐ 192.168.27.11 ne fait pas partie du réseau

Partie B

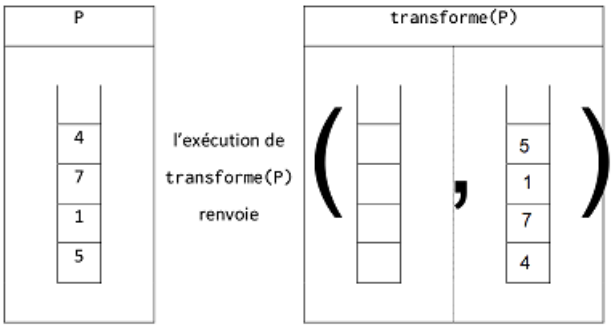
```
def IP_bin(IP_dec : list) -> list:
    """ renvoie une liste de 4 listes correspondant à l'adresse IP en notation binaire.
    pré: liste de 4 entiers compris entre 0 et 255 """
    if len(IP_dec) != 4:
        return None

    IP_bin = []
    for i in IP_dec:
        if 0 <= i <= 255:
            IP_bin.append(dec_bin(i))
        else:
            return None
    return IP_bin
```

Exercice 5

Q1



Q2



Q3

```
def maximum(pile : object) -> int:
    """ renvoie la valeur maximale d'une pile """
    if len(pile) == 0:
        return None

    max = depiler(pile)
    while not est_vide(pile):
        temp = depiler(pile)
        if temp > max:
            max = temp
    return max
```

Q4a

Pour compter les éléments d'une pile, il faut dépiler cette pile. A la fin du comptage la pile est vide et il faut la restorer.

On utilise alors une pile temporaire qui va recevoir les éléments dépiler lors du comptage et on redépiler les éléments de cette pile temporaire pour repeupler la pile initiale aux valeurs d'origines.

Q4b

```
def taille(pile : object) -> int:
    """ renvoie la taille d'une pile """
    taille = 0

    temp   = creer_pile()
    while not est_vide(pile):           # comptage
        empiler(temp, depiler(pile))
        taille += 1
    while not est_vide(temp):           # restoration
        empiler(pile, depiler(temp))

    return taille
```