

[← Index des sujets 2021](#)

## 21-NSIJ1G11 : Corrigé

Année : **2021**

Centre : **Etranger**

Jour : **1**

Enoncé : [!\[\]\(e3f8612927870f2e0f9f5989e6dd3064\_img.jpg\) PDF](#)

### 1. Exercice 1

*programmation objet (code de César)*

### 2. Exercice 2

*structures de données (dictionnaires)*

1. a.

 Script Python

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

b. 0

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

2. a. le paramètre choix peut être égal à "electrique" ou "classique"

b. 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 !

1. a.

 Script Python

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

b.

### Script Python

```
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.

### Script Python

```
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
```

## 3. Exercice 3

*arbres binaires de recherche*

```
graph TD
    N0["26.noeud00"] --> N1["3.noeud01"]
    N0 --> N2["42.noeud02"]
    N1 --> N7["1.noeud07"]
    N1 --> N3["15.noeud03"]
    N2 --> N4["29.noeud04"]
    N2 --> V1[" "]
    N3 --> N6["13.noeud06"]
    N3 --> N5["19.noeud05"]
    N4 --> V2[" "]
    N4 --> N8["32.noeud08"]
    N8 --> N10["30.noeud10"]
    N8 --> N9["37.noeud09"]
    N5 --> V3[" "]
    N5 --> N11["25.noeud11"]
    style V1 fill:#FFFFFF, stroke:#FFFFFF
    style V2 fill:#FFFFFF, stroke:#FFFFFF
    style V3 fill:#FFFFFF, stroke:#FFFFFF
    style N11 fill:#AA2222,stroke:#333
    linkStyle 0,3,7,13 stroke:#FF0000,stroke-width:2px
```

1. Etapes de l'insertion du noeud 11 de valeur 25 :

- A gauche du noeud00 car il a pour valeur 26 et  $25 < 26$

- A droite du noeud01 car il a pour valeur 3 et  $25 > 3$
  - A droite du noeud03 car il a pour valeur 15 et  $25 > 15$
  - A droite du noeud05 car il a pour valeur 19 et  $25 > 19$
2. A gauche du noeud04, on peut stocker les valeurs strictement inférieures à 29 et supérieures ou égales à 26. C'est à dire : 26,27, et 28.

### Note

Le sujet précise dans son introduction que :

les valeurs du sous-arbre droit sont **supérieures ou égales** à valeur du noeud.

Avec cette définition, la valeur 26 est donc possible même si elle est déjà présente dans l'arbre. Si on considère que les valeurs sont uniques seules 27 et 28 sont possibles.

3.

- a. 26, 3, 1, 15, 13, 19, 25, 42, 29, 32, 30, 37
  - b. C'est un parcours préfixé car la valeur du noeud est listé *avant* celle des valeurs présentes dans le sous arbre gauche et le sous arbre droit. La valeur du noeud serait listé *entre* ces valeurs pour un parcours infixé et *après* pour un parcours suffixé.
4. Afin d'afficher les valeurs par ordre croissant, on doit effectuer un parcours infixé. C'est à dire afficher la valeur du noeud *entre* les valeurs du sous arbre gauche et du sous arbre droit.

### Texte

Parcours2(A) :

```
Parcours(A.fils_gauche)
Afficher(A.valeur)
Parcours(A.fils_droit)
```

## 4. Exercice 4

réseau

1. a. On utilise 4 octets dans une adresse IP V4

2.b, 2.c et 2.d

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 0 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			
	Ligne 6	192	168	20	0

2. Adresses IP possibles : 192.168.20.30 et 192.168.20.230,

Partie B

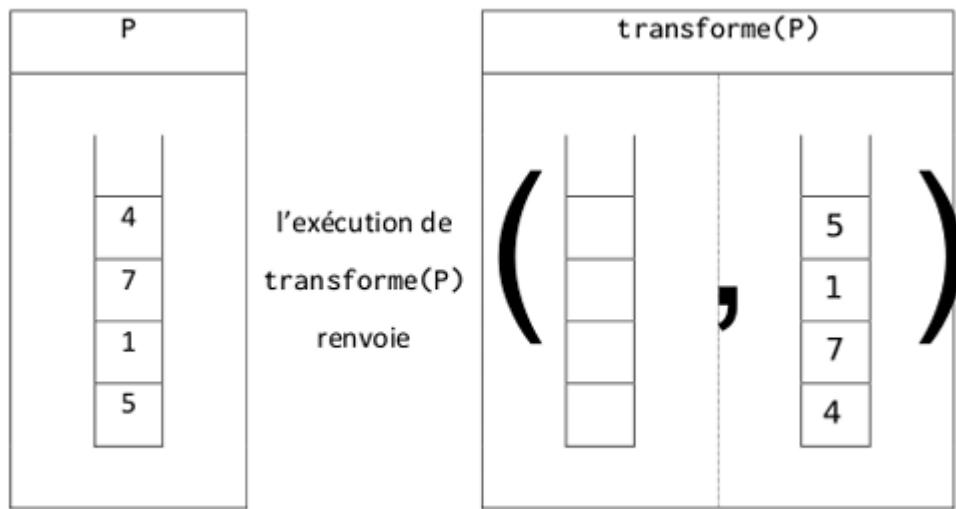
### 🐍 Script Python

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

## 5. Exercice 5

*structure de données (piles)*

1.



2.

3.

### Script Python

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

1. a. 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 incrémentera 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 original de pile. Il suffit ensuite de renvoyer la valeur du compteur.

b.

### Script Python

```
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
```