Exercices Corrigés Python (série 8)

Exercice 1 :

Etant donné un fichier texte nommé 'F_IPV4.txt' contenant dans chaque ligne une adresse IPV4. On se propose de vérifier la validité des adresses IPV4 stockés dans ce fichier, de déterminer la classe à laquelle appartient chacune des adresses valides, de les faire migrer vers le système IPV6 et de stocker dans un fichier d'enregistrements nommé 'F_IPV6.txt' chaque adresse IPV4 valide ainsi que la classe à laquelle elle appartient et son équivalent en IPV6.

Pour ce faire, on dispose des informations suivantes :

1. Une adresse IPV4 valide est codée sur quatre octets (32 bits) et représentée sous la forme W.X.Y.Z avec W, X, Y et Z sont quatre entiers naturels appartenant chacun à l'intervalle [0,255] et séparés par le caractère '.'

   NB. Pour vérifier la validité d'une adresse IPV4, le candidat est appelé uniquement à vérifie si W, X, Y et Z sont dans l'intervalle [0,255].
2. Chaque adresse IPV4 valide appartient à une classe :
   - Classe A, si la valeur du premier bit à gauche de la représentation en binaire de W est 0.
   - Classe B, si la valeur des deux premiers bits à gauche de la représentation en binaire de W est 10.
   - Classe C, si la valeur des trois premiers bits à gauche de la représentation en binaire de W est 110.
   - Classe D, si la valeur des quatre premiers bits à gauche de la représentation en binaire de W est 1110
   - Classe E, si la valeur des quatre premiers bits à gauche de la représentation en binaire de W est 1111
3. Une adresse IPV6 est codée sur 16 octets (128 bits). Pour faire migrer une adresse IPV4 valide vers le système IPV6, on va s'intéresser uniquement au bloc de 32 bits dans l'adresse IPV6 qui représente la conversion en hexadécimal de l'adresse IPV4.

   Pour ce faire, on convertit chacun des nombres W, X, Y et Z en hexadécimal, puis, les concaténer en insérant le caractère ' :' au milieu du résultat obtenu.

Exemple

L'adresse 155.105.50.68 est valide et elle appartient à la classe B car la valeur des deux premiers bits à gauche de la représentation en binaire de 155 qui est 10011011 est 10.

- L'équivalent du nombre décimal 155 en hexadécimal est 9B
- L'équivalent du nombre décimal 105 en hexadécimal est 69
- L'équivalent du nombre décimal 50 en hexadécimal est 32
- L'équivalent du nombre décimal 69 en hexadécimal est 45

Donc, le bloc de 32 bits dans l'adresse IPV6 qui représente la conversion en hexadécimal de l'adresse IPV4 est 9B69 :3245

Travail demandé

1. Ecrire une fonction ***valide(ip)*** qui permet de vérifier la validité d'une adresse IPV4 (True or False)
2. Ecrire une fonction ***classe(ip)*** qui retourne la classe d'une adresse ip
3. Ecrire une fonction ***adresseip6(ip)*** qui permet de convertir une adresse ip en V4 vers une adresse IPV6
4. Ecrire la fonction ***Genere()*** qui permet de générer le fichier ***'F_IPV6.txt'***

Remarque :

- *La fonction bin(nb) permet de convertir en binaire un nombre nb (**bin(155) → 0b10011011**)*
- *La fonction hex(nb) permet de convertir un nombre décimal en hexadécimal (**hex(155) → 0x9b**)*

Correction :

```
def valide(ip):
	ad=ip.split('.')
	if len(ad)==4:
		if 0<int(ad[0])<255 and 0<int(ad[1])<255 and 0<int(ad[2])<255 and 0<int(ad[3])<255:
			return True
		else:	return False
	else:	return False


def binaire(nb):
	val=['0','0','0','0','0','0','0','0']
	binn=bin(int(nb))
	i=len(val)-len(binn[2:])
	for lettre in binn[2:]:
		val[i]=lettre
		i+=1
	return ''.join(val)


def classe(ip):
	if valide(ip):
		ad=ip.split('.')
		binn=binaire(ad[0])
		if binn[0]=='0':
			return 'A'
		elif binn[:2]=='10':
			return 'B'
		elif binn[:3]=='110':
			return 'C'
		elif binn[:4]=='1110':
			return 'D'
		else:
			return 'E'
	else:	return False
```

```
def adresseip6(ip):
	if valide(ip):
		ad=ip.split('.')
		adresse=hex(int(ad[0]))[2:]+hex(int(ad[1]))[2:]+':'+hex(int(ad[2]))[2:]+hex(int(ad[3]))[2:]+'\n'
		return adresse
	else:	return ''



def Genere():
	source=open('F_IPV4.txt')
	dest=open('F_IPV6.txt','a')
	for ligne in source:
		if valide(ligne.strip()):
			chaine=ligne.strip()+' : '+classe(ligne.strip())+' : '+adresseip6(ligne.strip())
			dest.write(chaine)
	source.close()
	dest.close
```

Exercice 2 :

On se propose de crypter un message, formé uniquement par des lettres majuscules et des espaces, en utilisant la méthode de chiffrement de Polybe qui consiste à :

- Ranger, dans une matrice carrée de dimension 5x5, les lettres d'un mot-clé donné suivies des lettres restantes de l'alphabet dans l'ordre, à l'exception de la lettre 'W'.

  Le mot-clé est une chaine de caractères formée uniquement de L lettres majuscules, sans doublons et ne contenant pas la lettre 'W' (avec 3<=L<=10).
- Remplacer chaque lettre du message à crypter par les coordonnées de sa position dans la matrice (le numéro de la ligne suivi du numéro de la colonnes), sachant que :
  - Le caractère espace ne subit aucun cryptage ;
  - La lettre 'W' sera remplacée par les coordonnées de la lettre 'V'

Exemple

Pour le mot-clé 'MYSTER, on construit la matrice suivante :

	1	2	3	4	5
1	M	Y	S	T	E
2	R	A	B	C	D
3	F	G	H	I	J
4	K	L	N	O	P
5	Q	U	V	X	Z

Le cryptage du message 'CHERCHER POLYBE DANS WIKIPEDIA' sera :

'2433152124331521_454442122315_25224313_533441344515253422' ou le mot 'WIKIPEDIA' est crypté comme suit : '**53**3441344515253422' car 'W' remplacé par '53' (les coordonnées de la lettre 'V'), 4I4 est remplacé par '34', 'K' est remplacé par '41', 'P' est remplacé par '45', 'E' est remplacé par '15', 'D' est remplacé par '25' et 'A' est remplacé par '22'.

Travail demandé

1. Ecrire une fonction Python **Crypter_Polybe(message, motcle)**, qui permet de crypter un message donnéselon le mot-clé en respectant les contraintes cités ci-dessus selon la méthode de chiffrement de Polybe.
2. Ecrire une fonction **Decrypter_Polybe(message, motcle),** qui permet de décrypter le message donnéselon le mot-clé selon le chiffrement de Polybe.

Correction :

```
def matrice(motcle):
    alpha='ABCDEFGHIJKLMNOPQRSTUVXYZ'
    Mat=[[0]*5 for _ in range(5)]
    Mat[0][0]=motcle[0]
    i=0
    j=1
    for pos in range(1,len(motcle)):
        if j==5:
            i+=1
            j=0
        if motcle[pos] not in motcle[:pos]:
            Mat[i][j]=motcle[pos]
            j+=1
    for pos in range(len(alpha)):
        if j==5:
            i+=1
            j=0
        if alpha[pos] not in motcle:
            Mat[i][j]=alpha[pos]
            j+=1
    return Mat
def Crypter_Polybe(msg, motcle):
    Mat=matrice(motcle)
    res=''
    for pos in range(len(msg)):
        if msg[pos]==' ':
            res+='_'
        elif msg[pos]=='W':
            res+='00'
        else:
            i=0
            j=0
            while Mat[i][j]!=msg[pos]:
                j+=1
                if j==5:
                    j=0
                    i+=1
            res+=str(i+1)+str(j+1)
    return res
```

```
def Decrypter_Polybe(msg, motcle):
    Mat=matrice(motcle)
    res=''
    k=0
    while k<len(msg)-1:
        s=msg[k]
        if msg[k]=='_':
            res+=' '
            k+=1
        elif msg[k]=='0':
            res+='W'
            k+=2
        else:
            i=int(msg[k])-1
            j=int(msg[k+1])-1
            res+=Mat[i][j]
            k+=2
    return res
```