Devoir à la maison n°2 - Corrigé

Il s'agissait du sujet 2023 CCINP, filière TSI (sans la partie sur les base de données)

Extrait du rapport de jury :

Les erreurs les plus fréquentes portent globalement sur :
- la manipulation des listes en général ;
- la manipulation du slicing mal comprise, surtout avec les indices négatifs ;
- la manipulation des chaînes de caractères, notamment des confusions entre le type int et le type str ;
- une mauvaise compréhension de la valeur retournée par une fonction ;
- la notion de copie superficielle ou profonde sur des listes.
Bien que cela ne soit pas une erreur, trop de candidats utilisent la formulation suivante sur les manipulations des expressions logiques : ***`True if test==True else False`*** *ou son équivalant sur 2 lignes.*

Q1. Voici une proposition de code :

```
def num_secu(chaine): #chaine est de type str
    chaine2='' #pour recopier la chaine de départ sans les espaces
    for c in chaine:
        if c!=' ':
            chaine2+=c
    return int(chaine2) #on noublie pas de transformer en entier
```

Q2. Voici une proposition de code :

```
def clef(numero):
    return 97-numero%97
```

Q3. Voici une proposition de code :

```
def num_secu_complet(numero):
    chaine1=str(numero)
    chaine2=str(clef(numero))
    return int(chaine1+chaine2)
```

Remarque : si la clef ne possède qu'un seul chiffre il faudrait penser à rajouter un 0 entre le nombre et la clef. J'imagine que le sujet ne parlant pas de cette éventualité, tous les points ont été mis avec un script comme celui ci-dessus.

Q4. Voici une proposition de code, qui transforme d'abord le numéro en chaine de caractères pour pouvoir facilement parcourir ses chiffres de gauche à droite :

```
def test_num_secu(chaine):
    num_complet=num_secu(chaine) #pour avoir un entier
    num_tronque=num_complet//100 #le numéro sans la clef
    bon_num=num_secu_complet(num_tronque) #avec la bonne clef
    return num_complet==bon_num
```

Q5. Voici une proposition de code :

```
def num_en_liste(numero):
    chaine=str(numero)
     liste =[]
    for c in chaine:
         liste .append(int(c))
    return  liste
```

Q6. Voici une proposition de code :

```
def  tuple_pairs_impairs (numero):
     liste =num_en_liste(numero)
    pairs=[]
    impairs=[]
    n=len( liste )
    for i in range(n):
        if i%2==0:
            impairs .append( liste [n-i-1])
        else:
            pairs .append( liste [n-i-1])
    return ( pairs , impairs )
```

Q7. Voici une proposition de code :

```
def cree_dico (numero):
    couple= tuple_pairs_impairs (numero)
    d={'pair':couple [0], 'impair':couple[1]}
    return d
```

Q8. Voici une proposition de code :

```
def  traitement_nb_pairs (d):
     liste =d['pair']
    d_new=d.copy() #le nouveau dictionnaire que l'on renverra
    n=len( liste )
    for i in range(n):
        double=liste [i]*2
        if double<10:
            liste [i]=double
        else:
            d=double//10 #chiffre des dizaines
            u=double%10 #chiffre des unités
             liste [i]=d+u
    d_new['pair']= liste
    return d_new
```

Q9. Voici une proposition de code :

```
def  test_num_carte_credit (numero):
    d=traitement_nb_pairs( cree_dico (numero))
    s=0
    for c in d['pair']:
        s+=c
    for c in d['impair']:
        s+=c
    return s%10==0
```

Q10. Voici une proposition de code :

```
def init (n):
    return [[0 for i in range(n)]for j in range(n)]
```

Q11. Voici une proposition de code :

```
def charge_valeur(img):
    liste =init(21)

    for i in range(21):
        for j in range(21):
            liste [k][l]=img.getpixel(20*i,20*j)
    return( liste )
```

Q12. Voici une proposition de code, qui compare pixel par pixel le bon bloc de positionnement et le bloc présent dans le QRcode :

```
def cree_bloc():
    liste =init(7)   #on cree un tableau de zeros
    for j in range(1,6):
        liste [1][j]=1
        liste [5][j]=1
    for i in range(2,5):
        liste [i][1]=1
        liste [i][5]=1
    return liste
```

Q13. Voici une proposition de code :

```

def test_bloc(x,y, liste_depart ):
    bon_motif=cree_bloc()
    test=True
    for i in range(7):
        for j in range(7):
            if bon_motif[i][j]!= liste_depart [x+i][y+j]:
                test=False
    return test
```

Q14. Voici une proposition de code :

```
def test_QRcode(mat):
    return test_bloc (0,0,mat) and test_bloc (0,14,mat) and test_bloc (14,0,mat)
```

Q15. Remarque : l'énoncé n'était pas très clair, il fallait comprendre qu'ici on ne fait tourner que 4 éléments du QRcode (le premier étant en position (x, y))

```
def tourHoraire(x,y,mat):
    b=mat[x][y]
    h=mat[y][n-x]
    o=mat[n-x][n-y]
    i=mat[n-y][x]
    mat[y][n-x]=b
    mat[n-x][n-y]=h
    mat[n-y][x]=o
    mat[x][y]=i
    return None
```

Q16. On va utiliser la focntion précédente, mais attention il ne faut pas faire tourner tous les pixels, sinon ils vont tourner plusieurs fois. Ici on ne va faire tourner que le quart supérieur gauche :

```
def rotationHoraire(mat):
    n=len(mat)
    for x in range(n//2):
        for y in range((n+1)//2):
            tourHoraire(x,y,mat)
    return None
```

Q17. On suppose ici que `mat` représente bien un QRcode valide, sinon la boucle while ne s'arrêtera jamais... :

```
def QRcode_posi(mat):
    while test_QRcode(mat)==False:
        rotationHoraire (mat)
    return None
```

Q18. Voici le tableau complété :

	A	B	C	D	E	F
étape initiale	0	∞	∞	∞	∞	∞
$A(0)$	–	2	∞	1	∞	∞
$D\,(1_A)$	–	2	∞	–	3	∞
$B\,(2_A)$	–	–	5	–	3	7
$E\,(3_D)$	–	–	5	–	–	4
$F\,(4_E)$	–	–	5	–	–	–
$C\,(5_B)$	–	–	–	–	–	–

Q19. Le plus court chemin entre A et F vaut 4. Il suffit de regarder, dans la liste finale, la valeur située à l'indice F.

Pour obtenir le détail du chemin permettant d'obtenir cette valeur, il faut remonter les prédécesseurs à partir de F : F précédé de E précédé de D précédé de A.

Le chemin de longueur 4 est donc $\boldsymbol{A \to D \to E \to F}$.