EXERCICE 2 (6 points)

Cet exercice porte sur les listes, les dictionnaires et la programmation de base en Python.

Pour son évaluation de fin d'année, l'institut d'Enseignement Néo-moderne (EN) a décidé d'adopter le principe du QCM. Chaque évaluation prend la forme d'une liste de questions numérotées de 0 à 19. Pour chaque question, 5 réponses sont proposées. Les réponses sont numérotées de 1 à 5. Exactement une réponse est correcte par question et chaque candidat coche exactement une réponse par question. Pour chaque évaluation, on dispose de la correction sous forme d'une liste `corr` contenant pour chaque question, la bonne réponse ; c'est-à-dire telle que `corr[i]` est la bonne réponse à la question `i`. Par exemple, on présente ci-dessous la correction de l'épreuve 0 :

```
corr0 = [4, 2, 1, 4, 3, 5, 3, 3, 2, 1, 1, 3, 3, 5, 4, 4, 5, 1, 3, 3].
```

Cette liste indique que pour l'épreuve 0, la bonne réponse à la question 0 est 4, et que la bonne réponse à la question 19 est 3.

Avant de mettre une note, on souhaite corriger les copies question par question ; c'est-à-dire associer à chaque copie, une liste de booléens de longueur 20, indiquant pour chaque question, si la réponse donnée est la bonne. Le candidat Tom Matt a rendu la copie suivante pour l'épreuve 0 :

```
copTM = [4, 1, 5, 4, 3, 3, 1, 4, 5, 3, 5, 1, 5, 5, 5, 1, 3, 3, 3, 3].
```

La liste de booléens correspondante est alors :

```
corrTM = [True, False, False, True, True, False, False, False,
False, False, False, False, False, True, False, False, False, False,
True, True].
```

1. Écrire en Python une fonction `corrige` qui prend en paramètre `cop` et `corr`, deux listes d'entiers entre 1 et 5 et qui renvoie la liste des booléens associée à la copie `cop` selon la correction `corr`.

   Par exemple, `corrige(copTM, corr0)` renvoie `corrTM`.

La note attribuée à une copie est simplement le nombre de bonnes réponses. Si on dispose de la liste de booléens associée à une copie selon la correction, il suffit donc de compter le nombre de `True` dans la liste. Tom Matt obtient ainsi 6/20 à l'épreuve 0. On remarque que la construction de cette liste de booléens n'est pas nécessaire pour calculer la note d'une copie.

2. Écrire en Python une fonction `note` qui prend en paramètre `cop` et `corr`, deux listes d'entiers entre 1 et 5 et qui renvoie la note attribuée à la copie `cop` selon la correction `corr`, sans construire de liste auxiliaire.

   Par exemple, `note(copTM, corr0)` renvoie 6.

L'institut EN souhaite automatiser totalement la correction de ses copies. Pour cela, il a besoin d'une fonction pour corriger des paquets de plusieurs copies. Un paquet de copies est donné sous la forme d'un dictionnaire dont les clés sont les noms des candidats et les valeurs sont les listes représentant les copies de ces candidats. On peut considérer un paquet `p1` de copies où l'on retrouve la copie de Tom Matt :

```
p1 = {('Tom', 'Matt'): [4, 1, 5, 4, 3, 3, 1, 4, 5, 3, 5, 1, 5, 5, 5,
1, 3, 3, 3, 3], ('Lambert', 'Ginne'): [2, 4, 2, 2, 1, 2, 4, 2, 2, 5,
1, 2, 5, 5, 3, 1, 1, 1, 4, 4], ('Carl', 'Roth'): [5, 4, 4, 2, 1, 4,
5, 1, 5, 2, 2, 3, 2, 3, 3, 5, 2, 2, 3, 4], ('Kurt', 'Jett'): [2, 5,
5, 3, 4, 1, 5, 3, 2, 3, 1, 3, 4, 1, 3, 1, 3, 2, 4, 4], ('Ayet',
'Finzerb'): [4, 3, 5, 3, 2, 1, 2, 1, 2, 4, 5, 5, 1, 4, 1, 5, 4, 2,
3, 4]}.
```

3. Écrire en Python une fonction `notes_paquet` qui prend en paramètre un paquet de copies `p` et une correction `corr` et qui renvoie un dictionnaire dont les clés sont les noms des candidats du paquet `p` et les valeurs sont leurs notes selon la correction `corr`.

   Par exemple, `notes_paquet(p1, corr0)` renvoie `{('Tom', 'Matt'): 6, ('Lambert', 'Ginne'): 4, ('Carl', 'Roth'): 2, ('Kurt', 'Jett'): 4, ('Ayet', 'Finzerb'): 3}`.

   La fonction `notes_paquet` peut faire appel à la fonction `note` demandée en question 2, même si cette fonction n'a pas été écrite.

Pour éviter les problèmes d'identification des candidats qui porteraient les mêmes noms et prénoms, un employé de l'institut EN propose de prendre en compte les prénoms secondaires des candidats dans les clés des dictionnaires manipulés.

4. Expliquer si on peut utiliser des listes de noms plutôt qu'un couple comme clés du dictionnaire.

5. Proposer une autre solution pour éviter les problèmes d'identification des candidats portant les mêmes prénoms et noms. Cette proposition devra prendre en compte la sensibilité des données et être argumentée succinctement.

Un ingénieur de l'institut EN a démissionné en laissant une fonction Python énigmatique sur son poste. Le directeur est convaincu qu'elle sera très utile, mais encore faut-il comprendre à quoi elle sert.

Voici la fonction en question :

```
1  def enigme(notes) :
2      a = None
3      b = None
4      c = None
5      d = {}
6      for nom in notes :
7          tmp = c
8          if a == None or notes[nom] > a[1] :
9             c = b
10            b = a
11            a = (nom, notes[nom])
12         elif b == None or notes[nom] > b[1] :
13            c = b
14            b = (nom, notes[nom])
15         elif c == None or notes[nom] > c[1] :
16            c = (nom, notes[nom])
17         else :
18             d[nom] = notes[nom]
19         if tmp != c and tmp != None :
20             d[tmp[0]] = tmp[1]
21     return (a, b, c, d)
```

6. Calculer ce que renvoie la fonction `enigme` pour le dictionnaire `{('Tom', 'Matt'): 6, ('Lambert', 'Ginne'): 4, ('Carl', 'Roth'): 2, ('Kurt', 'Jett'): 4, ('Ayet', 'Finzerb'): 3}`.

7. En déduire ce que calcule la fonction `enigme` lorsqu'on l'applique à un dictionnaire dont les clés sont les noms des candidats et les valeurs sont leurs notes.

8. Expliquer ce que la fonction `enigme` renvoie s'il y a strictement moins de 3 entrées dans le dictionnaire passées en paramètre.

9. Écrire en Python une fonction `classement` prenant en paramètre un dictionnaire dont les clés sont les noms des candidats et les valeurs sont leurs notes et qui, en utilisant la fonction `enigme`, renvoie la liste des couples ((prénom, nom), note) des candidats classés par notes décroissantes.

   Par exemple, `classement({('Tom', 'Matt'): 6, ('Lambert', 'Ginne'): 4, ('Carl', 'Roth'): 2, ('Kurt', 'Jett'): 4, ('Ayet', 'Finzerb'): 3})` renvoie `[(('Tom', 'Matt'), 6), (('Lambert', 'Ginne'), 4), (('Kurt', 'Jett'), 4), (('Ayet', 'Finzerb'), 3), (('Carl', 'Roth'), 2)]`.

Le professeur Paul Tager a élaboré une évaluation particulièrement innovante de son côté. Toutes les questions dépendent des précédentes. Il est donc assuré que dès qu'un candidat s'est trompé à une question, alors toutes les réponses suivantes sont

également fausses. M. Tager a malheureusement égaré ses notes, mais il a gardé les listes de booléens associées. Grâce à la forme particulière de son évaluation, on sait que ces listes sont de la forme

`[True, True, …, True, False, False, …, False].`

Pour recalculer ses notes, il a écrit les deux fonctions Python suivantes (dont la seconde est incomplète) :

```
1  def renote_express(copcorr) :
2      c = 0
3      while copcorr[c] :
4          c = c + 1
5      return c
```

```
1  def renote_express2(copcorr) :
2      gauche = 0
3      droite = len(copcorr)
4      while droite - gauche > 1 :
5          milieu = (gauche + droite)//2
6          if copcorr[milieu] :
7              ...
8          else :
9              ...
10      if copcorr[gauche] :
11         return ...
12     else :
13         return ...
```

10. Compléter le code de la fonction Python `renote_express2` pour qu'elle calcule la même chose que `renote_express`.

11. Déterminer les coûts en temps de `renote_express` et `renote_express2` en fonction de la longueur n de la liste de booléens passée en paramètre.

12. Expliquer comment adapter `renote_express2` pour obtenir une fonction qui corrige très rapidement une copie pour les futures évaluations de M. Tager s'il garde la même spécificité pour ses énoncés. Cette fonction ne devra pas construire la liste de booléens correspondant à la copie corrigée, mais directement calculer la note.