Lycée Les 3 Sources **Bourg-Lès-Valence**
NSI 1re Année **2025-26**

Devoir surveillé n°4 – correction

Exercice 1 : (13pt) *Cet exercice porte sur les listes, les dictionnaires et la programmation de base en Python.*

Partie A :

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 dans la suite du sujet la liste `corr0` qui correspond à la correction de l'épreuve 0. 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.

Les copies des élèves sont également sous forme de liste, avec pour chaque question, la réponse donnée. On donne ci-dessous `copTM` qui est la copie de Tom Matt pour l'épreuve 0.

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

1) a) Calculer la note de Tom, sachant qu'une bonne réponse vaut 1 point et qu'une mauvaise réponse en vaut 0. Il a 6 bonnes réponses, donc 6.

   b) Compléter le code de la 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`. Ainsi, `note(copTM, corr0)` renvoie la réponse trouvée à la question précédente.

```
def note(cop, corr):
    n = 0
    for i in range(len(cop)):
        if cop[i] == corr[i]:
            n = n + 1
    return n
```

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, 1, 3, 1, 1, 2, 4, 4],
 'Carl Roth':      [5, 2, 4, 2, 3, 4, 5, 1, 5, 1, 2, 3, 2, 3, 3, 5, 2, 2, 3, 4],
 'Kurt Jett':      [2, 5, 5, 3, 4, 1, 5, 3, 4, 3, 3, 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]
 }
```

2) On souhaite écrire 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 dont leurs notes selon la correction `corr`.

Par exemple :

```
>>> notes_paquet(p1, corr0)
{'Tom Matt': XX, 'Lambert Ginne': 2, 'Carl Roth': 5, 'Kurt Jett': 2,
'Ayet Finzerb': 3}
```

La valeur `XX` correspond à la réponse à la question 1) a).

a) À l'aide des exemples précédents, donner la valeur correspondant à l'évaluation des expressions suivantes :

```
>>> p1['Lambert Ginne'][1]
4
>>> notes = notes_paquet(p1, corr0)
>>> notes['Kurt Jett']
2
>>> [c for c in notes if notes[c] < 4]
['Lambert Ginne', 'Kurt Jett', 'Ayet Finzerb']
```

b) Compléter le code de la fonction `notes_paquet`. Il doit y avoir un appel à la fonction `note` définie précédemment.

```
def notes_paquet(paquet, corr):
    notes = {}
    for nom in paquet:
        notes[nom] = note(paquet[nom], corr)
    return notes
```

Partie B :

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      reste = {}
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             reste[nom] = notes[nom]
19         if tmp != c and tmp != None:
20             reste[tmp[0]] = tmp[1]
21     return (a, b, c, reste)
```

3) On considère le dictionnaire suivant :

```
notes = {'Tom Matt': 6, 'Lambert Ginne': 2, 'Carl Roth': 5, 'Kurt Jett': 2,
'Ayet Finzerb': 3}
```

a) Compléter le tableau ci-dessous qui correspond à l'exécution de `enigme(notes)`. À part la première ligne qui correspond aux valeurs des variables avant l'entrée dans la boucle, les autres correspondent aux valeurs des variables après la ligne 20. Les noms ont été raccourcis pour simplifier les réponses.

nom	a	b	c	reste
	None	**None**	**None**	{}
'Tom'	('Tom', 6)	**None**	**None**	{}
'Lamb'	('Tom', 6)	('Lamb', 2)	**None**	{}
'Carl'	('Tom', 6)	('Carl', 5)	('Lamb', 2)	{}
'Kurt'	('Tom', 6)	('Carl', 5)	('Lamb', 2)	{'Kurt': 2}
'Ayet'	('Tom', 6)	('Carl', 5)	('Ayet', 3)	{'Kurt': 2, 'Lamb': 2}

b) Expliquer, en français, 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.

Solution : Cette fonction détermine les 3 meilleurs notes dans le dictionnaire et renvoie des couples de la forme `(nom, note)`. La quatrième valeur correspond au dictionnaires des notes qui ne font pas partie des 3 meilleures.

c) Quelles sont les valeurs de `c` et de `reste` renvoyées par `enigme` si le dictionnaire a strictement moins de 3 entrées?

Solution : Avec moins de 3 entrées, `c` vaut **None** et `reste` est un dictionnaire vide.

d) Compléter la fonction `classement` qui prend 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 `(nom, note)` des candidats classés par notes décroissantes.

```
>>> classement({'Tom Matt': 6, 'Lambert Ginne': 2, 'Carl Roth': 5,
                'Kurt Jett': 2, 'Ayet Finzerb': 3})
[('Tom Matt', 6), ('Carl Roth', 5), ('Ayet Finzerb', 3),
('Lambert Ginne', 2), ('Kurt Jett', 2)]
```

```
def classement(notes):
    cl = []
    reste = notes
    while len(reste) > 0:
        a, b, c, reste = enigme(reste)
        for v in [a, b, c]:
            if v != None:
                cl.append(v)
    return cl
```

Partie C :

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. Cette fois, il n'enregistre pas les réponses des élèves mais uniquement si la réponse est bonne ou pas. On obtient des listes de booléens avec **True** si la réponse est bonne et **False** sinon.
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. Voici la première :

```
def renote_express(copcorr) :
    c = 0
    while copcorr[c] == True:
        c = c + 1
    return c
```

La seconde est incomplète :

```
def renote_express2(copcorr) :
    gauche = 0
    droite = len(copcorr)
    while droite - gauche > 1:
        milieu = (gauche + droite)//2
        if copcorr[milieu] == True:
            gauche = milieu
        else:
            droite = milieu
    if copcorr[gauche] == True:
        return droite
    else:
        return gauche
```

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

b) Expliquer pourquoi la fonction `renote_express2` va particulièrement plus vite que la fonction `renote_express` pour les longues évaluations.

Solution : Cette fonction utilise la dichotomie. Il faut donc regarder nettement moins de valeurs à regarder pour pouvoir répondre.

c) (Bonus) 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 prendre en paramètre la liste de booléens correspondant à la copie corrigée, mais la copie et le corrigé. Il ne faut pas construire la liste de booléens.

Solution : Il faut remplacer tous les tests cherchant à savoir si `copcorr[i]` est vrai par `cop[i] == corr[i]`.

Rappels sur les dictionnaires en Python :

Action	Instruction et syntaxe
Créer un dictionnaire vide	`dico={}`
Obtenir un élément d'un dictionnaire existant à partir de sa clé et renvoie une erreur si cle n'existe pas dans le dictionnaire	`dico[cle]`
Modifier la valeur d'un élément d'un dictionnaire à partir de sa clé	`dico[cle]=nouvelle_valeur`
Ajouter un élément dans un dictionnaire existant	`dico[nouvelle_cle]=valeur`
Tester l'appartenance d'un élément à un dictionnaire (renvoie un booléen)	`cle in dico`
Afficher les associations `cle:valeur` du dictionnaire `dico`	`for cle in dico:` `  print(cle, dico[cle])`

Exercice 2 : (13pt) *Cet exercice porte sur la programmation Python et la cryptographie.*
Le chiffrement Playfair, popularisé par Lord Playfair et utilisé par l'armée britannique durant les guerres du XXème siècle, est basé sur le chiffrement de paires de lettres (appelées **digrammes**).

Partie A : la clef de chiffrement

Ce chiffrement utilise un tableau de 5×5 lettres contenant un mot clef. On remplit le tableau avec les lettres du mot clef (sans doublons), puis on le complète avec les lettres restantes de l'alphabet (sans la lettre W) dans leur ordre alphabétique. Une lettre n'apparait qu'une seule fois dans le tableau.

Par exemple, si on choisit comme clef le mot `PLAYFAIR`, le carré de chiffrement obtenu est celui présenté dans la Figure 1.
On commence par les lettres de la clef (cases blanches) sans les doublons (ici le A) puis on complète le tableau (cases grisées) avec les lettres restantes de l'alphabet, dans l'ordre alphabétique.

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

Figure 1.

1) Représenter ci-dessous le carré de chiffrement obtenu avec la clef `EPREUVEDENSI`. Les lettres de l'alphabet à utiliser sont :
`A B C D E F G H I J K L M N O P Q R S T U V X Y Z`

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

On donne ci-dessous le code Python de la fonction `creer_liste_clef` qui prend en paramètre la clef de chiffrement et renvoie une liste contenant 25 lettres ordonnées de la façon suivante : d'abord les lettres de la clef choisie (sans doublon) puis les lettres de l'alphabet restantes (classées par ordre alphabétique).

```
1  def creer_liste_clef(clef):
2      """ hypothèse : la clef ne contient pas la lettre W """
3      assert ...
4      deja_utilises = []
5      # alphabet sans la lettre W:
6      alphabet = 'ABCDEFGHIJKLMNOPQRSTUVXYZ'
7      for i in range(len(clef)):
8          if not (clef[i] in deja_utilises):
9              deja_utilises.append(clef[i])
10     for lettre in alphabet:
11         if not lettre in deja_utilises:
12             deja_utilises.append(lettre)
13     return deja_utilises
```

```
>>> creer_liste_clef('PLAYFAIR')
['P', 'L', 'A', 'Y', 'F', 'I', 'R', 'B', 'C', 'D', 'E', 'G',
'H', 'J', 'K', 'M', 'N', 'O', 'Q', 'S', 'T', 'U', 'V', 'X', 'Z']
```

2) Compléter l'assertion au début de la fonction `creer_liste_clef` afin de s'assurer que l'hypothèse sur la clef soit respectée.
Solution : `assert 'W' not in clef`

On donne ci-dessous le code incomplet de la fonction `creer_carre` qui prend en paramètre la liste créée par la fonction `creer_liste_clef` et renvoie le carré de chiffrement.

```
def creer_carre(liste_clef):
    carre = [[0 for j in range(5)] for i in range(5)]
    for i in range(25):
        carre[............][............] = liste_clef[i]
    return carre
```

```
>>> creer_carre(creer_liste_clef('PLAYFAIR'))
[['P', 'L', 'A', 'Y', 'F'], ['I', 'R', 'B', 'C', 'D'], ['E', 'G', 'H', 'J', 'K'],
['M', 'N', 'O', 'Q', 'S'], ['T', 'U', 'V', 'X', 'Z']]
```

3) Compléter le code de la fonction `creer_carre`, en utilisant les opérateurs `%` (reste de la division entière) et `//` (division entière).
Solution : `carre[i//5][i%5] = liste_clef[i]`

Partie B : chiffrer un message Le chiffrement d'un message se fait en deux étapes :

- 1ère étape : on découpe le message en digrammes (paires de lettres) ;
- 2ème étape : on chiffre chacun des digrammes avec le carré de chiffrement.

Pour découper le message en digrammes, on prend les lettres deux par deux en tenant compte de deux cas particuliers :

- si les deux lettres sont identiques, on ajoute un `'X'` après la première lettre et on poursuit le découpage deux à deux à partir de la deuxième lettre ;
- s'il ne reste qu'une seule lettre, on forme une dernière paire en lui ajoutant la lettre un `'X'`.

Par exemple, le découpage de `'BACCALAUREAT'` donnera `'BA'`, `'CX'`, `'CA'`, `'LA'`, `'UR'`, `'EA'`, `'TX'`.
Le chiffrement d'un message se fait ensuite en chiffrant chaque digramme (paire de lettres), de la manière suivante :

- si les lettres du digramme se trouvent sur la même ligne du carré de chiffrement, il faut les remplacer par celles se trouvant immédiatement à leur droite (en bouclant sur la gauche si le bord est atteint) ;
- si les lettres apparaissent sur la même colonne du carré de chiffrement, les remplacer par celles qui sont juste en dessous (en bouclant par le haut si le bas de la table est atteint) ;
- sinon, remplacer les lettres par celles se trouvant sur la même ligne du carré de chiffrement, mais dans le coin opposé du rectangle défini par la paire originale.

Par exemple, si le message est `'VIVELANSI'`, les digrammes sont : `'VI'`, `'VE'`, `'LA'`, `'NS'`, `'IX'` et leurs codages avec la clef `PLAYFAIR` sont :

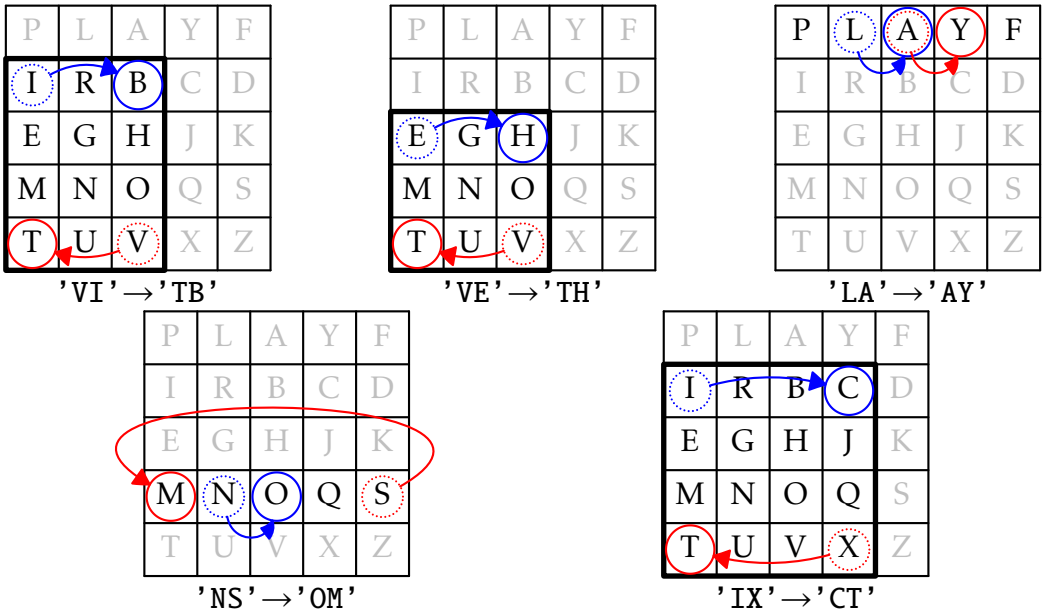

Figure 2. Chiffrement de quelques digrammes

On donne ci-dessous le code incomplet de la fonction `couper_en_digrammes` qui prend en paramètre une chaîne de caractères et renvoie la liste des digrammes la constituant :

```
1  def couper_en_digrammes(message):
2      digrammes = []
3      i = 0
4      while i < len(message) - 1:
5          if message[i] == message[i+1]:
6              digrammes.append(message[i] + 'X')
7              i = i + 1
8          else:
9              ...
10             i = i + 2
11     if i == len(message) - 1:   # il reste une lettre isolée
12         digrammes.append(message[i] + 'X')
13     return digrammes
```

4) Compléter la ligne 9 de la fonction `couper_en_digrammes`.
Solution : `digrammes.append(message[i]+message[i+1])`

5) Donner le résultat de l'appel `couper_en_digrammes('POMMES')`.
Solution : `['PO', 'MX', 'ME', 'SX']`

6) Donner le chiffrement du message `POMMES` avec le carré de chiffrement `PLAYFAIR` donné en Figure 1.
Solution : `AMQTTMQZ`

La fonction `ligne_colonne` prend en paramètres une lettre et le carré de chiffrement créé par la fonction `creer_carre`, et renvoie les coordonnées de la lettre dans le carré de chiffrement. On suppose que la lettre donnée se trouve dans le carré.

```
1  def ligne_colonne(lettre, carre):
2      for i in range(.............):
3          for j in range(.............):
4              if ............................:
5                  return ...
```

Exemple (avec le carré de chiffrement de la Figure 1) :

```
>>> ligne_colonne('A', carre)
(0, 2)
>>> ligne_colonne('N', carre)
(3, 1)
```

7) Compléter le code de la fonction `ligne_colonne`.
Solution :

```
def ligne_colonne(lettre, carre):
    for i in range(5):
        for j in range(5):
            if lettre == carre[i][j]:
                return (i, j)
```

Voici le code incomplet de la fonction `chiffrer_digramme` qui prend en paramètres le carré de chiffrement et un digramme, et qui renvoie le digramme chiffré correspondant :

```python
def chiffrer_digramme(digramme, carre):
    lettre1 = digramme[0]
    lettre2 = digramme[1]
    i1, j1 = ligne_colonne(lettre1, carre)
    i2, j2 = ligne_colonne(lettre2, carre)
    if i1 == i2:  # même ligne
        c1 = carre[i1][(j1+1)%5]
        c2 = carre[i2][(j2+1)%5]
    elif ..............:  # même colonne
        c1 = ...
        c2 = ...
    else:  # autres cas
        c1 = ...
        c2 = ...
    digramme_chiffre = c1 + c2
    return digramme_chiffre
```

8) Compléter le code de la fonction `chiffrer_digramme`.

```python
def chiffrer_digramme(digramme, carre):
    lettre1 = digramme[0]
    lettre2 = digramme[1]
    i1, j1 = ligne_colonne(lettre1, carre)
    i2, j2 = ligne_colonne(lettre2, carre)
    if i1 == i2: # même ligne
        c1 = carre[i1][(j1+1)%5]
        c2 = carre[i2][(j2+1)%5]
    elif j1 == j2: # même colonne
        c1 = carre[(i1+1)%5][j1]
        c2 = carre[(i2+1)%5][j2]
    else: # autres cas
        c1 = carre[i1][j2]
        c2 = carre[i2][j1]
    digramme_chiffre = c1 + c2
    return digramme_chiffre
```

9) Écrire le code python de la fonction `chiffrer_playfair` qui prend en paramètres deux chaînes de caractères `message` et `clef` correspondant au message à chiffrer et au mot-clef choisi, et qui renvoie le message chiffré, en utilisant les fonctions déjà écrites précédemment.

Exemple :

```
>>> chiffrer_playfair('VIVELANSI', 'PLAYFAIR')
'TBTHAYOMCT'
```

```python
def chiffrer_playfair(message, clef):
    carre = creer_carre(clef)
    digrames = couper_en_digrammes(message)
    res = ""
    for d in digramme:
        res = res + chiffrer_digramme(d, carre)
    return res
```

Lycée Les 3 Sources **Bourg-Lès-Valence**
NSI 1re Année **2025-26**

Devoir surveillé n°4 – correction

Exercice 1 : (13pt) *Cet exercice porte sur les listes, les dictionnaires et la programmation de base en Python.*

Partie A :

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 dans la suite du sujet la liste `corr0` qui correspond à la correction de l'épreuve 0. 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.

Les copies des élèves sont également sous forme de liste, avec pour chaque question, la réponse donnée. On donne ci-dessous `copTM` qui est la copie de Tom Matt pour l'épreuve 0.

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

1) a) Calculer la note de Tom, sachant qu'une bonne réponse vaut 1 point et qu'une mauvaise réponse en vaut 0. Il a 8 bonnes réponses, donc 8.

   b) Compléter le code de la 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`. Ainsi, `note(copTM, corr0)` renvoie la réponse trouvée à la question précédente.

```
def note(cop, corr):
    n = 0
    for i in range(len(cop)):
        if cop[i] == corr[i]:
            n = n + 1
    return n
```

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, 5, 1, 4, 5, 1, 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, 3, 1, 5, 2, 2, 3, 2, 3, 3, 5, 2, 2, 3, 4],
 'Kurt Jett':      [2, 2, 5, 3, 4, 1, 5, 3, 2, 3, 1, 3, 3, 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, 5, 2, 3, 4]
 }
```

2) On souhaite écrire 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 dont leurs notes selon la correction `corr`.

Par exemple :

```
>>> notes_paquet(p1, corr0)
{'Tom Matt': XX, 'Lambert Ginne': 4, 'Carl Roth': 3, 'Kurt Jett': 6,
'Ayet Finzerb': 4}
```

La valeur `XX` correspond à la réponse à la question 1) a).

a) À l'aide des exemples précédents, donner la valeur correspondant à l'évaluation des expressions suivantes :

```
>>> p1['Ayet Finzerb'][1]
3
>>> notes = notes_paquet(p1, corr0)
>>> notes['Kurt Jett']
6
>>> [c for c in notes if notes[c] < 5]
['Lambert Ginne', 'Carl Roth', 'Ayet Finzerb']
```

b) Compléter le code de la fonction `notes_paquet`. Il doit y avoir un appel à la fonction `note` définie précédemment.

```
def notes_paquet(paquet, corr):
    notes = {}
    for nom in paquet:
        notes[nom] = note(paquet[nom], corr)
    return notes
```

Partie B :

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      reste = {}
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             reste[nom] = notes[nom]
19         if tmp != c and tmp != None:
20             reste[tmp[0]] = tmp[1]
21     return (a, b, c, reste)
```

3) On considère le dictionnaire suivant :

```
notes = {'Tom Matt': 8, 'Lambert Ginne': 4, 'Carl Roth': 3, 'Kurt Jett': 6,
'Ayet Finzerb': 4}
```

a) Compléter le tableau ci-dessous qui correspond à l'exécution de `enigme(notes)`. À part la première ligne qui correspond aux valeurs des variables avant l'entrée dans la boucle, les autres correspondent aux valeurs des variables après la ligne 20. Les noms ont été raccourcis pour simplifier les réponses.

nom	a	b	c	reste
	None	**None**	**None**	{}
'Tom'	('Tom', 8)	**None**	**None**	{}
'Lamb'	('Tom', 8)	('Lamb', 4)	**None**	{}
'Carl'	('Tom', 8)	('Lamb', 4)	('Carl', 3)	{}
'Kurt'	('Tom', 8)	('Kurt', 6)	('Lamb', 4)	{'Carl': 3}
'Ayet'	('Tom', 8)	('Kurt', 6)	('Lamb', 4)	{'Carl': 3, 'Ayet': 4}

b) Expliquer, en français, 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.

Solution : Cette fonction détermine les 3 meilleurs notes dans le dictionnaire et renvoie des couples de la forme `(nom, note)`. La quatrième valeur correspond au dictionnaires des notes qui ne font pas partie des 3 meilleures.

c) Quelles sont les valeurs de `c` et de `reste` renvoyées par `enigme` si le dictionnaire a strictement moins de 3 entrées?

Solution : Avec moins de 3 entrées, `c` vaut **None** et `reste` est un dictionnaire vide.

d) Compléter la fonction `classement` qui prend 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 `(nom, note)` des candidats classés par notes décroissantes.

```
>>> classement({'Tom Matt': 8, 'Lambert Ginne': 4, 'Carl Roth': 3,
                'Kurt Jett': 6, 'Ayet Finzerb': 4})
[('Tom Matt', 8), ('Kurt Jett', 6), ('Lambert Ginne', 4),
 ('Ayet Finzerb', 4), ('Carl Roth', 3)]
```

```
def classement(notes):
    cl = []
    reste = notes
    while len(reste) > 0:
        a, b, c, reste = enigme(reste)
        for v in [a, b, c]:
            if v != None:
                cl.append(v)
    return cl
```

Partie C :

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. Cette fois, il n'enregistre pas les réponses des élèves mais uniquement si la réponse est bonne ou pas. On obtient des listes de booléens avec **True** si la réponse est bonne et **False** sinon.
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. Voici la première :

```
def renote_express(copcorr) :
    c = 0
    while copcorr[c] == True:
        c = c + 1
    return c
```

La seconde est incomplète :

```
def renote_express2(copcorr) :
    gauche = 0
    droite = len(copcorr)
    while droite - gauche > 1:
        milieu = (gauche + droite)//2
        if copcorr[milieu] == True:
            gauche = milieu
        else:
            droite = milieu
    if copcorr[gauche] == True:
        return droite
    else:
        return gauche
```

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

b) Expliquer pourquoi la fonction `renote_express2` va particulièrement plus vite que la fonction `renote_express` pour les longues évaluations.

Solution : Cette fonction utilise la dichotomie. Il faut donc regarder nettement moins de valeurs à regarder pour pouvoir répondre.

c) (Bonus) 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 prendre en paramètre la liste de booléens correspondant à la copie corrigée, mais la copie et le corrigé. Il ne faut pas construire la liste de booléens.

Solution : Il faut remplacer tous les tests cherchant à savoir si `copcorr[i]` est vrai par `cop[i] == corr[i]`.

Rappels sur les dictionnaires en Python :

Action	Instruction et syntaxe
Créer un dictionnaire vide	`dico={}`
Obtenir un élément d'un dictionnaire existant à partir de sa clé et renvoie une erreur si cle n'existe pas dans le dictionnaire	`dico[cle]`
Modifier la valeur d'un élément d'un dictionnaire à partir de sa clé	`dico[cle]=nouvelle_valeur`
Ajouter un élément dans un dictionnaire existant	`dico[nouvelle_cle]=valeur`
Tester l'appartenance d'un élément à un dictionnaire (renvoie un booléen)	`cle in dico`
Afficher les associations `cle:valeur` du dictionnaire `dico`	`for cle in dico:` `  print(cle, dico[cle])`

Exercice 2 : (13pt) *Cet exercice porte sur la programmation Python et la cryptographie.*
Le chiffrement Playfair, popularisé par Lord Playfair et utilisé par l'armée britannique durant les guerres du XXème siècle, est basé sur le chiffrement de paires de lettres (appelées **digrammes**).

Partie A : la clef de chiffrement

Ce chiffrement utilise un tableau de 5×5 lettres contenant un mot clef. On remplit le tableau avec les lettres du mot clef (sans doublons), puis on le complète avec les lettres restantes de l'alphabet (sans la lettre W) dans leur ordre alphabétique. Une lettre n'apparait qu'une seule fois dans le tableau.

Par exemple, si on choisit comme clef le mot `PLAYFAIR`, le carré de chiffrement obtenu est celui présenté dans la Figure 1.
On commence par les lettres de la clef (cases blanches) sans les doublons (ici le A) puis on complète le tableau (cases grisées) avec les lettres restantes de l'alphabet, dans l'ordre alphabétique.

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

Figure 1.

1) Représenter ci-dessous le carré de chiffrement obtenu avec la clef `CONTROLEDENSI`. Les lettres de l'alphabet à utiliser sont :
`A B C D E F G H I J K L M N O P Q R S T U V X Y Z`

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

On donne ci-dessous le code Python de la fonction `creer_liste_clef` qui prend en paramètre la clef de chiffrement et renvoie une liste contenant 25 lettres ordonnées de la façon suivante : d'abord les lettres de la clef choisie (sans doublon) puis les lettres de l'alphabet restantes (classées par ordre alphabétique).

```
1  def creer_liste_clef(clef):
2      """ hypothèse : la clef ne contient pas la lettre W """
3      assert ...
4      deja_utilises = []
5      # alphabet sans la lettre W:
6      alphabet = 'ABCDEFGHIJKLMNOPQRSTUVXYZ'
7      for i in range(len(clef)):
8          if not (clef[i] in deja_utilises):
9              deja_utilises.append(clef[i])
10     for lettre in alphabet:
11         if not lettre in deja_utilises:
12             deja_utilises.append(lettre)
13     return deja_utilises
```

```
>>> creer_liste_clef('PLAYFAIR')
['P', 'L', 'A', 'Y', 'F', 'I', 'R', 'B', 'C', 'D', 'E', 'G',
'H', 'J', 'K', 'M', 'N', 'O', 'Q', 'S', 'T', 'U', 'V', 'X', 'Z']
```

2) Compléter l'assertion au début de la fonction `creer_liste_clef` afin de s'assurer que l'hypothèse sur la clef soit respectée.
Solution : `assert 'W' not in clef`

On donne ci-dessous le code incomplet de la fonction `creer_carre` qui prend en paramètre la liste créée par la fonction `creer_liste_clef` et renvoie le carré de chiffrement.

```
def creer_carre(liste_clef):
    carre = [[0 for j in range(5)] for i in range(5)]
    for i in range(25):
        carre[............][............] = liste_clef[i]
    return carre
```

```
>>> creer_carre(creer_liste_clef('PLAYFAIR'))
[['P', 'L', 'A', 'Y', 'F'], ['I', 'R', 'B', 'C', 'D'], ['E', 'G', 'H', 'J', 'K'],
['M', 'N', 'O', 'Q', 'S'], ['T', 'U', 'V', 'X', 'Z']]
```

3) Compléter le code de la fonction `creer_carre`, en utilisant les opérateurs `%` (reste de la division entière) et `//` (division entière).
Solution : `carre[i//5][i%5] = liste_clef[i]`

Partie B : chiffrer un message Le chiffrement d'un message se fait en deux étapes :

- 1ère étape : on découpe le message en digrammes (paires de lettres) ;
- 2ème étape : on chiffre chacun des digrammes avec le carré de chiffrement.

Pour découper le message en digrammes, on prend les lettres deux par deux en tenant compte de deux cas particuliers :

- si les deux lettres sont identiques, on ajoute un `'X'` après la première lettre et on poursuit le découpage deux à deux à partir de la deuxième lettre ;
- s'il ne reste qu'une seule lettre, on forme une dernière paire en lui ajoutant la lettre un `'X'`.

Par exemple, le découpage de `'BACCALAUREAT'` donnera `'BA'`, `'CX'`, `'CA'`, `'LA'`, `'UR'`, `'EA'`, `'TX'`.
Le chiffrement d'un message se fait ensuite en chiffrant chaque digramme (paire de lettres), de la manière suivante :

- si les lettres du digramme se trouvent sur la même ligne du carré de chiffrement, il faut les remplacer par celles se trouvant immédiatement à leur droite (en bouclant sur la gauche si le bord est atteint) ;
- si les lettres apparaissent sur la même colonne du carré de chiffrement, les remplacer par celles qui sont juste en dessous (en bouclant par le haut si le bas de la table est atteint) ;
- sinon, remplacer les lettres par celles se trouvant sur la même ligne du carré de chiffrement, mais dans le coin opposé du rectangle défini par la paire originale.

Par exemple, si le message est `'VIVELANSI'`, les digrammes sont : `'VI'`, `'VE'`, `'LA'`, `'NS'`, `'IX'` et leurs codages avec la clef `PLAYFAIR` sont :

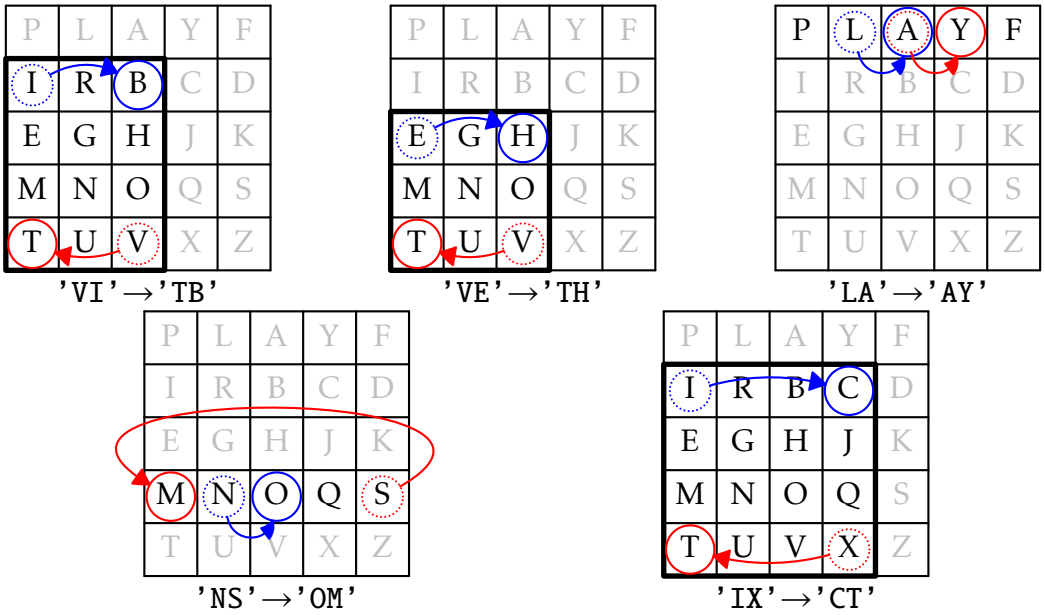

Figure 2. Chiffrement de quelques digrammes

On donne ci-dessous le code incomplet de la fonction `couper_en_digrammes` qui prend en paramètre une chaîne de caractères et renvoie la liste des digrammes la constituant :

```
1  def couper_en_digrammes(message):
2      digrammes = []
3      i = 0
4      while i < len(message) - 1:
5          if message[i] == message[i+1]:
6              digrammes.append(message[i] + 'X')
7              i = i + 1
8          else:
9              ...
10             i = i + 2
11     if i == len(message) - 1:   # il reste une lettre isolée
12         digrammes.append(message[i] + 'X')
13     return digrammes
```

4) Compléter la ligne 9 de la fonction `couper_en_digrammes`.
Solution : `digrammes.append(message[i]+message[i+1])`

5) Donner le résultat de l'appel `couper_en_digrammes('PELLES')`.
Solution : `['PE', 'LX', 'LE', 'SX']`

6) Donner le chiffrement du message `PELLES` avec le carré de chiffrement `PLAYFAIR` donné en Figure 1.
Solution : `IMYUPGQZ`

La fonction `ligne_colonne` prend en paramètres une lettre et le carré de chiffrement créé par la fonction `creer_carre`, et renvoie les coordonnées de la lettre dans le carré de chiffrement. On suppose que la lettre donnée se trouve dans le carré.

```
1  def ligne_colonne(lettre, carre):
2      for i in range(.............):
3          for j in range(.............):
4              if ............................:
5                  return ...
```

Exemple (avec le carré de chiffrement de la Figure 1) :

```
>>> ligne_colonne('A', carre)
(0, 2)
>>> ligne_colonne('N', carre)
(3, 1)
```

7) Compléter le code de la fonction `ligne_colonne`.
Solution :

```
def ligne_colonne(lettre, carre):
    for i in range(5):
        for j in range(5):
            if lettre == carre[i][j]:
                return (i, j)
```

Voici le code incomplet de la fonction `chiffrer_digramme` qui prend en paramètres le carré de chiffrement et un digramme, et qui renvoie le digramme chiffré correspondant :

```python
def chiffrer_digramme(digramme, carre):
    lettre1 = digramme[0]
    lettre2 = digramme[1]
    i1, j1 = ligne_colonne(lettre1, carre)
    i2, j2 = ligne_colonne(lettre2, carre)
    if i1 == i2:  # même ligne
        c1 = carre[i1][(j1+1)%5]
        c2 = carre[i2][(j2+1)%5]
    elif ..............:  # même colonne
        c1 = ...
        c2 = ...
    else:  # autres cas
        c1 = ...
        c2 = ...
    digramme_chiffre = c1 + c2
    return digramme_chiffre
```

8) Compléter le code de la fonction `chiffrer_digramme`.

```python
def chiffrer_digramme(digramme, carre):
    lettre1 = digramme[0]
    lettre2 = digramme[1]
    i1, j1 = ligne_colonne(lettre1, carre)
    i2, j2 = ligne_colonne(lettre2, carre)
    if i1 == i2: # même ligne
        c1 = carre[i1][(j1+1)%5]
        c2 = carre[i2][(j2+1)%5]
    elif j1 == j2: # même colonne
        c1 = carre[(i1+1)%5][j1]
        c2 = carre[(i2+1)%5][j2]
    else: # autres cas
        c1 = carre[i1][j2]
        c2 = carre[i2][j1]
    digramme_chiffre = c1 + c2
    return digramme_chiffre
```

9) Écrire le code python de la fonction `chiffrer_playfair` qui prend en paramètres deux chaînes de caractères `message` et `clef` correspondant au message à chiffrer et au mot-clef choisi, et qui renvoie le message chiffré, en utilisant les fonctions déjà écrites précédemment.

Exemple :

```
>>> chiffrer_playfair('VIVELANSI', 'PLAYFAIR')
'TBTHAYOMCT'
```

```python
def chiffrer_playfair(message, clef):
    carre = creer_carre(clef)
    digrames = couper_en_digrammes(message)
    res = ""
    for d in digramme:
        res = res + chiffrer_digramme(d, carre)
    return res
```