Corrigé — Examen Python & IA

Sujet B — Durée : 1h30 — Barème indicatif

Partie I — Questions de cours (6 points)

Q1 (1 pt)

Différence : un `tuple` est **immuable** (on ne peut pas modifier, ajouter ou supprimer d'éléments après création), une `list` est **muable**.
Cas d'usage tuple : clé de dictionnaire, coordonnées fixes (x, y), retour de fonction avec plusieurs valeurs.

(0.5 pt pour immuable vs muable, 0.5 pt pour un exemple cohérent de tuple — accepter « clé de dict », « protection contre les modifications involontaires », etc.)

Q2 (1.5 pt)

Une **table de hachage** est une structure de données qui associe des **clés** à des **valeurs** en utilisant une fonction de hachage (`hash()`) qui transforme la clé en un index (entier) pour un accès direct en mémoire. En Python, les dictionnaires (`dict`) sont implémentés avec des tables de hachage : la clé est hachée, le résultat détermine l'emplacement où stocker la valeur. Cela donne un accès en **O(1)** en moyenne.

(0.5 pt principe, 0.5 pt fonction de hachage, 0.5 pt lien avec dict Python + O(1) — accepter « rapide », « constant » pour O(1))

Q3 (1.5 pt)

Le **one-hot encoding** représente chaque catégorie par un vecteur binaire de taille égale au nombre de catégories, avec un 1 à la position correspondante et des 0 ailleurs.
Exemple avec ["rouge", "vert", "bleu"] :
rouge → [1, 0, 0], vert → [0, 1, 0], bleu → [0, 0, 1].

(0.5 pt principe, 0.5 pt exemple correct, 0.5 pt précision « vecteur binaire » ou « un 1, des 0 » — attention : un encodage {rouge: 1, vert: 2, bleu: 3} n'est PAS du one-hot)

Q4 (1 pt)

`resultat` = `{"abc": 2, "def": 1}`
(deux occurrences de "abc", une de "def")

(0.5 pt pour les bonnes clés, 0.5 pt pour les bons comptes — accepter toute notation correcte du dict)

Q5 (1 pt)

Vrai. Une compréhension de liste peut inclure un filtre `if` :

```
[x**2 for x in range(10) if x % 2 == 0]  # → [0, 4, 16, 36, 64]
```

(0.5 pt vrai/faux, 0.5 pt exemple correct — accepter tout exemple valide avec `if`)

Partie II — Écrire du code (8 points)

Q6 (2 pts)

```
def freq_caracteres(texte):
    d = {}
    for c in texte:
        if c == ' ':
            continue
        d[c] = d.get(c, 0) + 1
    return d
```

Variante sans `continue` : `if c != ' ': d[c] = d.get(c, 0) + 1`

(0.5 pt boucle sur les caractères, 0.5 pt exclusion des espaces, 0.5 pt mise à jour du compteur, 0.5 pt return — si pas d'exclusion des espaces : -0.5)

Q7 (2 pts)

```
def intersection(s1, s2):
    res = set()
    for e in s1:
        if e in s2:
            res.add(e)
    return res
```

(0.5 pt création d'un set, 0.5 pt boucle sur s1, 0.5 pt test d'appartenance, 0.5 pt add + return — si retourne une liste au lieu d'un set : -0.25)

Q8 (2 pts)

```
def somme_chiffres(n):
    if n < 10:
        return n
    return n % 10 + somme_chiffres(n // 10)
```

(0.5 pt cas de base n < 10, 1 pt appel récursif n//10, 0.5 pt addition avec n%10 — accepter n==0 comme cas de base si géré correctement)

Q9 (2 pts)

```
def pairs_au_carre(liste):
    return [x**2 for x in liste if x % 2 == 0]
```

(0.5 pt compréhension de liste, 0.5 pt condition `if x % 2 == 0`, 0.5 pt `x**2`, 0.5 pt return — si pas de compréhension : -0.5)

Partie III — Analyse de code (4 points)

Q10 (2 pts)

Sortie exacte :

```
Intersection : [3, 4]
Union : [1, 2, 3, 4, 5, 6]
Dans a mais pas dans b : [1, 2]
```

(0.5 pt chaque ligne correcte, 0.5 pt pour l'ordre croissant dû à `sorted()` — attention : enlever 0.5 si les crochets sont absents ou si l'ordre est faux)

Q11 (2 pts)

Cette fonction calcule la **transposée** d'une matrice (inverser lignes et colonnes).

(1.5 pt pour « transposée » ou « transpose la matrice », 0.5 pt pour « échange lignes/colonnes » — accepter « retourne la matrice », « pivot » si l'idée est claire. Refuser « tri » ou « copie ».)

Partie IV — Petit problème (2 points)

Q12 (2 pts)

```
notes = {
    "maths":    [12, 15, 8, 17],
    "francais": [10, 14, 13, 11],
    "anglais":  [16, 9, 14, 12]
}

moyennes = {}
for matiere, liste in notes.items():
    moyennes[matiere] = sum(liste) / len(liste)
    print(matiere, ":", moyennes[matiere])
```

```
meilleure = max(moyennes, key=moyennes.get)
print("Meilleure moyenne :", meilleure)
```

(0.5 pt boucle sur items, 0.5 pt calcul correct de la moyenne, 0.5 pt stockage/affichage, 0.5 pt détermination de la meilleure matière. Accepter une approche sans `max()` : `if moy > max_moy: ...`. Si tout est hardcodé (sans boucle) : max 1 pt.)

Bonus — Pour les courageux·ses (1 pt)

B1 (0.5 pt)

Erreur : on utilise une liste `[1, 2]` comme clé de dictionnaire. Les listes sont muables → non hachables.
Correction : remplacer par un tuple : `d = {(1, 2): "a"}`.

(0.25 pt identifier l'erreur, 0.25 pt correction par tuple)

B2 (0.5 pt)

« Le XOR est involutif : la même opération appliquée deux fois **annule** (ou **inverse**) le message original. »

(Accepter « annule », « inverse », « restaure l'original ». Refuser « chiffre ».)

Récapitulatif du barème

Question	Points	Thème
Q1	1	tuple vs list
Q2	1.5	table de hachage
Q3	1.5	one-hot encoding
Q4	1	dictionnaire (get)
Q5	1	compréhension + if
Q6	2	fréquence caractères
Q7	2	intersection d'ensembles
Q8	2	somme chiffres récursive
Q9	2	compréhension avec filtre
Q10	2	analyse (ensembles)
Q11	2	analyse (transposée)

Q12	2	problème dictionnaire de listes
B1–B2	1	bonus
Total	**20+1**	