Corrigé — Examen Python & IA

Durée : 1h30 — Barème indicatif

Partie I — Questions de cours (6 points)

Q1 (1 pt)

Différence : Une `list` est ordonnée, indexée par entiers, accepte les doublons. Un `set` est non ordonné, sans doublons, et ne peut pas être indexé.
Cas d'usage : liste → classement d'étudiants ; set → ensemble des pays visités (élimine les doublons).

(0.5 pt diff. claire, 0.25 pt × 2 exemples pertinents — accepter tout exemple cohérent)

Q2 (1.5 pt)

Mémoïsation : technique qui stocke le résultat d'une fonction coûteuse dans un dictionnaire pour éviter de le recalculer si la même entrée est re-sollicitée.
Exemple : Fibonacci récursif — sans mémoïsation, `fib(40)` explose ; avec un dict de cache, chaque valeur n'est calculée qu'une fois.

```
cache = {}
def fib(n):
    if n in cache: return cache[n]
    if n <= 1: return n
    cache[n] = fib(n-1) + fib(n-2)
    return cache[n]
```

(0.5 pt principe, 0.5 pt utilité, 0.5 pt exemple — accepter factorielle avec cache ou tout autre exemple)

Q3 (1.5 pt)

Bag of Words : représente un texte par un dictionnaire {mot: fréquence} en ignorant la grammaire et l'ordre des mots.
Perte : l'ordre des mots, la syntaxe, la structure de la phrase. « Le chat mange la souris » et « La souris mange le chat » ont le même sac de mots mais des sens opposés.

(0.5 pt principe, 0.5 pt perte identifiée, 0.5 pt exemple illustratif)

Q4 (1 pt)

Le dictionnaire contient **2 entrées** : `"pomme"` et `"poire"`.
`d["pomme"]` vaut **5** car la deuxième affectation `d["pomme"] = 5` écrase la valeur précédente (3).

Justification : les clés d'un dictionnaire sont uniques. Si on réaffecte une clé existante, l'ancienne valeur est remplacée.

(0.5 pt nb entrées, 0.5 pt valeur + justif)

Q5 (1 pt)

Vrai. J(A, A) = |A ∩ A| / |A ∪ A| = |A| / |A| = 1.

(0.5 pt vrai/faux, 0.5 pt justification par la formule — accepter aussi « l'intersection = l'union = l'ensemble lui-même, donc rapport = 1 »)

Partie II — Écrire du code (8 points)

Q6 (2 pt)

```
def comptage_mots(texte):
    d = {}
    for mot in texte.split():
        if mot in d:
            d[mot] += 1
        else:
            d[mot] = 1
    return d
```

Variante acceptée avec `.get()` : `d[mot] = d.get(mot, 0) + 1`

(1 pt boucle + split, 0.5 pt mise à jour correcte, 0.5 pt retour — enlever 0.5 si pas de return. Acceptez toute syntaxe correcte.)

Q7 (2 pt)

```
def jaccard(s1, s2):
    inter = len(s1 & s2)   # ou len(s1.intersection(s2))
    union = len(s1 | s2)   # ou len(s1.union(s2))
    if union == 0:
        return 0
    return inter / union
```

(1 pt intersection, 0.5 pt union, 0.5 pt division + gestion union=0 — tolérer sans gestion de union=0 si tests avec ensembles non vides)

Q8 (2 pt)

```
def factorielle(n):
    if n == 0:
```

```
        return 1
    return n * factorielle(n - 1)
```

(1 pt cas de base n=0, 0.5 pt appel récursif, 0.5 pt return correct — si récursif mais pas de cas de base → infini → 0 pt. Enlever 0.5 si la condition est n≤1 au lieu de n==0 tant que ça marche pour n≥0.)

Q9 (2 pt)

```
def normaliser(vecteur):
    norme = sum(x**2 for x in vecteur) ** 0.5
    return [x / norme for x in vecteur]
```

Variante avec `math.sqrt` : `import math; norme = math.sqrt(sum(x**2 for x in vecteur))`

(0.5 pt calcul norme, 0.5 pt compréhension de liste, 0.5 pt division correcte, 0.5 pt return — si pas de compréhension de liste : -0.5)

Partie III — Analyse de code (4 points)

Q10 (2 pt)

Sortie exacte :

```
 Moyenne : 15.0
Alice : 15 ★
Bob : 12
Charlie : 18 ★
```

(1 pt pour la ligne "Moyenne : 15.0", 0.5 pt pour les trois lignes noms, 0.5 pt pour les ★ au bon endroit — pénaliser -0.5 si "Moyenne : 15" sans .0)

Q11 (2 pt)

Réponse : Cette fonction applique un **chiffrement par décalage de César** (ou ROT-n) au texte : elle décale chaque lettre de `k` positions dans l'alphabet, en conservant la casse et les caractères non alphabétiques inchangés.

(1 pt pour « décalage » ou « César », 0.5 pt pour « conserve casse/caractères », 0.5 pt pour le paramètre k — accepter « chiffrement », « cryptage par substitution mono-alphabétique », etc.)

Partie IV — Petit problème (2 points)

Q12 (2 pt)

```
 notes_alice = {"maths": 15, "francais": 12, "anglais": 18}
notes_bob   = {"maths": 8,  "francais": 14, "anglais": 10}

moy_alice = sum(notes_alice.values()) / len(notes_alice)
moy_bob   = sum(notes_bob.values())   / len(notes_bob)

print("Moyenne d'Alice :", moy_alice)
print("Moyenne de Bob  :", moy_bob)

if moy_alice > moy_bob:
    print("Alice a la meilleure moyenne")
elif moy_bob > moy_alice:
    print("Bob a la meilleure moyenne")
else:
    print("Égalité !")
```

(0.5 pt sum/len pour Alice, 0.5 pt sum/len pour Bob, 0.5 pt affichage des moyennes, 0.5 pt comparaison + résultat final. Accepter `sum(...)/3` en dur au lieu de `len(...)` si cohérent avec 3 matières. Enlever 0.5 si le code ne gère pas le cas où les moyennes sont égales.)

Bonus — Pour les courageux·ses (1 pt)

B1 (0.5 pt)

Erreur : on essaie d'utiliser une liste `["nom", "prenom"]` comme clé de dictionnaire. Les listes ne sont pas **hachables** (mutable → hash non constant).
Correction : utiliser un tuple à la place : `d[("nom", "prenom")] = "Alice"`.

(0.25 pt identifier l'erreur « liste comme clé », 0.25 pt correction par tuple)

B2 (0.5 pt)

« Le XOR est son propre **inverse** ».

(Accepter « réciproque » ou « opposite » à la rigueur. « Inverse » est le terme exact.)

Récapitulatif du barème

Question	Points	Thème
Q1	1	list vs set
Q2	1.5	mémoïsation
Q3	1.5	Bag of Words

Q4	1	dictionnaire (clés)
Q5	1	Jaccard
Q6	2	comptage mots
Q7	2	Jaccard code
Q8	2	factorielle récursive
Q9	2	normalisation + compréhension
Q10	2	analyse de code (sortie)
Q11	2	analyse de code (César)
Q12	2	problème dictionnaires
B1–B2	1	bonus
Total	**20+1**	