Centres étrangers – 2025 – sujet2 - Correction

Exercice 2 (6 points)

Question 1

E = 0x45, W = 0x57 → somme = 0x45 + 0x57 = 0x9C

👉 Commentaire : *On additionne simplement les codes ASCII (en hexadécimal) des deux lettres.*

Question 2

Les deux mots contiennent les mêmes lettres dans un ordre différent, donc même somme ASCII, donc même clé.

👉 Commentaire : *La fonction de hachage est insensible à l'ordre des lettres puisqu'elle ne fait qu'une somme.*

Question 3

```
somme = 0
for caractere in mot:
    somme += ord(lettre)
```

👉 *Commentaire* : On initialise la somme puis on ajoute les codes ASCII un à un.

Question 4

Cela conserve uniquement l'octet de poids faible (valeur modulo 256). Les clés vont donc de 0 à 255.

👉 Commentaire : Utile pour s'assurer que la clé tient sur un octet (comme précisé dans l'énoncé).

Question 5

O(n), avec n le nombre de mots dans la liste.

👉 Commentaire : *Dans le pire cas, le mot à ajouter est plus grand que tous les autres*

Question 6

```
c in dico
```

👉 Commentaire : *C'est la syntaxe standard en Python pour tester la présence d'une clé.*

Question 7

```
def ajouter_mot_dict(dict_mots, mot):
    cle = code_hachage(mot)
    if cle in dict_mots:
        dict_mots[cle] = ajouter_mot_liste(dict_mots[cle], mot)
    else:
        dict_mots[cle] = [mot]
```

👉 Commentaire : *On vérifie si la clé existe déjà. Si oui, on ajoute dans la liste existante ; sinon, on crée une nouvelle entrée.*

Question 8

Appel 1 : debut=0, fin=5 → milieu=2 → liste[2]='PYTHON' → 'NSI' < 'PYTHON' → fin=2
Appel 2 : debut=0, fin=2 → milieu=1 → liste[1]='NSI' → égalité → retour True

👉 Commentaire : *On observe deux appels récursifs : le mot est trouvé au deuxième appel.*

Question 9

Elle divise l'espace de recherche par deux à chaque étape.

👉 Commentaire : *Cela permet d'éviter de tester chaque élément un à un, comme le ferait une recherche linéaire.*

Question 10

O(log n), c´est un logarithme de base 2.

👉 Commentaire : Une comparaison divise l'espace par 2, donc logarithmique en base 2.

Question 11

```
def mot_present(dict_mots, mot):
    cle = code_hachage(mot)
    if cle in dict_mots:
        liste = dict_mots[cle]
        return est_present(liste, mot, 0, len(liste))
    else:
        return False
```

👉 Commentaire : *On commence par calculer la clé, puis on utilise* `est_present` *si la clé est présente dans le dictionnaire.*
