Amérique du Nord – 2025 – sujet2 - Correction

Exercice 1 (6 points)

Partie A – Redondance par réplication

1. Écriture binaire du caractère 'a'

Le code ASCII du caractère 'a' est 97. Son écriture binaire sur 8 bits est : 01100001.

---

2. Fonction replique

Chaque bit de la liste est répété trois fois consécutivement.

Résultat de l'appel replique([0, 0, 1, 0, 1]) : `[0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1]`

---

3. Fonction nb_occurrences

```
def nb_occurrences(tab, i):
    d = {}
    for elt in tab:
        if elt in d:
            d[elt] = d[elt] + 1
        else:
            d[elt] = 1
    return d
```

👉*Commentaire : on utilise un dictionnaire pour compter le nombre d'occurrences.*

---

4. Fonction majorite

```
def majorite(d):
    cle_max = None
    val_max = -1
    for cle in d:
        if d[cle] > val_max:
            val_max = d[cle]
            cle_max = cle
    return cle_max
```

👉*Commentaire : on sélectionne la clé associée à la valeur maximale.*

Partie B – Bits de parité

5. Détection de l'erreur

La ligne 2 et la colonne 2 présentent une parité impaire : l'erreur se situe à leur intersection.

6. Fonction erreur_colonne

```
def erreur_colonne(mat):
    for j in range(len(mat[0])):
        somme = 0
        for i in range(len(mat)):
            somme += mat[i][j]
        if somme % 2 == 1:
            return j
```

👉*Commentaire : la colonne erronée est celle dont la somme des bits est impaire.*

Partie C – Code de Hamming (4,7)

7. Mot initial correspondant au code reçu

Le code reçu 1010000 diffère d'un seul bit du code 1110000 : le mot initial est donc 1000.

8. Fonction corriger_erreur

```
def corriger_erreur(code):
    for mot in table_hamming:
        diff = 0
        for i in range(7):
            if code[i] != table_hamming[mot][i]:
                diff += 1
        if diff <= 1:
            return table_hamming[mot]
```

👉*Commentaire : on compare le code reçu à tous les codes valides.*

---

9. Nombre de feuilles de l'arbre décodeur

L'arbre décodeur complet comporte 2^7 = 128 feuilles.

---

10. Fonction récursive decode

```
def decode(arbre, code, i):
    if arbre.gauche is None and arbre.droit is None:
        return arbre.etiquette
    if code[i] == 0:
        return decode(arbre.gauche, code, i + 1)
    else:
        return decode(arbre.droit, code, i + 1)
```

👉*Commentaire : on descend récursivement dans l'arbre en suivant les bits du code.*
