CORRIGÉ SUJET N°5

- EMPREINTE CARBONE -

Le Dossier ZIP de l'épreuve pratique N°5 se situe à l'adresse suivante :
https://sujets.examens-concours.gouv.fr/delos/api/file/public/69c4f059cb85140736350711

Ce corrigé est tiré du site suivant :
https://clem2429.github.io/Correction-Sujets-NSI-2026/

La version du fichier Python corrigé est accessible à l'adresse suivante :
https://clem2429.github.io/Correction-Sujets-NSI-2026/corrige/empreinte_corrige.py

- Ce sujet est le N°5 de la banque des sujets pratiques de NSI 2026;
- Ce sujet (corrigé) comporte 4 questions ;
- Chaque réponse à une question se trouve sur une page distincte.

QUESTION 1

Compléter le corps de la fonction `total_simple`. Ajouter un test permettant d'afficher l'empreinte carbone totale d'Ada, en utilisant les fonctions fournies pour accéder au fichier `empreinte_ada_aggr.json`.

```
def total_simple(empreinte):
    """Fonction qui renvoie l'empreinte carbone totale d'un
dictionnaire associant
    une empreinte carbone à des noms de catégories"""
    s = 0
    for key in empreinte:
        s += empreinte[key]
    return s
```

Afin d'afficher l'empreinte carbone totale d'Ada, on peut utiliser le code suivant : (Et tester sa valeur, au cas où...)

```
def test_total_simple():
    e_ada = chargement_json("empreinte_ada_agr.json")
    if est_dictionnaire(e_ada):
        # On a calculé depuis le fichier, on doit obtenir 7252
donc on prévient
        # l'erreur avant le return.
        assert total_simple(e_ada) == 7252, "Echec du calcul"
        return total_simple(e_ada)
```

QUESTION 2

En utilisant la fonction est_dictionnaire qui teste la nature d'une valeur, écrire le corps de la fonction récursive `total_rec` qui calcule la somme des valeurs numériques présentes dans des dictionnaires imbriqués. Des tests sont fournis dans la fonction `test_total_rec`, on pourra les compléter par un cas plus proche de la structure présente dans `empreinte_ada.json`.

```
def total_rec(empreinte):
    '''Fonction récursive qui renvoie l'empreinte carbone
totale représentée
    par un dictionnaire dont les valeurs peuvent aussi être des
dictionnaires'''
    s = 0
    for key in empreinte:
        if not est_dictionnaire(empreinte[key]):
            s += empreinte[key]
        else:
            s += total_rec(empreinte[key])
    return s
```

Afin d'ajouter un test pertinent pour cette fonction, on utilisera le fichier `empreinte_ada.json` qui contient des dictionnaires imbriqués.
Le test peut être rédigé de la manière suivante :

```
def test_total_rec():
    test_dico1 = {"a": 1, "d": 2}
    assert total_rec(test_dico1) == 3
    test_dico2 = {"a": {"b": 1, "c": 2}, "d": {"e": 3}}
    assert total_rec(test_dico2) == 6
    # Test ajouté avec l'empreinte totale de Ada (= 7252) :
    e_ada = chargement_json("empreinte_ada.json")
    assert total_rec(e_ada) == 7252, "Echec du Test pour
données complètes"
```

QUESTION 3

Lorsqu'on exécute la fonction alerte_valeur_aberrante sur le dictionnaire complet d'Ada avec une limite fixée à 1000, elle ne détecte aucune valeur aberrante, alors que le poste "Chauffage individuel" s'élève à 1500.
Expliquer précisément l'origine de cette erreur de conception et proposer une version corrigée de la fonction.

On exécute dans la console :

```
e_ada = chargement_json("empreinte_ada.json")
alerte_valeur_aberrante(e_ada, 1000)
>> True
```

Attention : l'énoncé est fautif car la fonction renvoie bien True. Cependant, la fonction reste fautive car elle ne parcourt par l'entiereté d'un dictionnaire.
Voici une proposition de correction :
On a par ailleurs rajouté une sécurité qui affiche si le dictionnaire est vide mais renvoie tout de même False afin de s'assurer que l'utilisateur pourra tout de même s'en servir.

```
def alerte_valeur_aberrante(empreinte, limite):
    """
    Fonction censée déterminer si au moins une valeur du
dictionnaire
    dépasse strictement la limite donnée.
    """
    if empreinte == {}:
        return "Votre dictionnaire est vide."
    for valeur in empreinte.values():
        if est_dictionnaire(valeur):
            if alerte_valeur_aberrante(valeur, limite):
                return True
        else:
            if valeur > limite:
                return True
    return False
```

QUESTION 4

Afin de prévenir toute régression future sur la fonction `alerte_valeur_aberrante` corrigée, il convient de définir une stratégie de validation robuste.
Proposer un jeu de tests pertinent pour cette fonction. Pour chaque cas de test envisagé, préciser la structure du dictionnaire fourni en entrée, le résultat attendu et la particularité algorithmique que ce test permet de vérifier.

Voici 7 tests avec leur jeu de données personnalisés réalisables :

```
def tests_valeur_aberrante():
    dico_simple = {
        "valeur_a": 2,
        "valeur_b": 5,
        "valeur_c": 10,
        "valeur_d": 0,
        "valeur_e": 46
     }
    # T1 - On teste avec un dictionnaire simple avec une valeur
aberrante :
    assert alerte_valeur_aberrante(dico_simple, 45) ==
True,"Echec Test 1"

    #T2 - On teste avec un dictionnaire simple sans une valeur
aberrante :
    assert alerte_valeur_aberrante(dico_simple, 50) ==
False,"Echec Test 2"

    #T3 - On teste avec un dictionnaire simple dont la valeur
max est la limite :
    assert alerte_valeur_aberrante(dico_simple, 46) ==
False,"Echec Test 3"

    dico_complexe = {
        "valeur_a": 2,
        "valeur_b": {
            "valeur_ba": 3,
            "valeur_bb": 19028,
            "valeur_bc": 0,
            "valeur_bd": 1039
        },
        "valeur_c": 10,
        "valeur_d": {
            "valeur_da": 2,
            "valeur_db": 1892
        },
        "valeur_e": {
            "valeur_ea": {
                "valeur_eaa": 2,
                "valeur_eab": {
                    "valeur_eaba": 20001
                },
```

```
                "valeur_eac": 2900
            },
            "valeur_eb": 29
        }
     }
    # T4 - On teste avec un dictionnaire complexe avec une
valeur aberrante :
    assert alerte_valeur_aberrante(dico_complexe, 20000) ==
True, "Echec Test 4"

    # T5 - On teste avec un dictionnaire complexe sans une
valeur aberrante :
    assert alerte_valeur_aberrante(dico_complexe, 900000) ==
False, "Echec Test 5"

    dico_nul = {}
    # T6 - On teste avec un dictionnaire nul :
    assert alerte_valeur_aberrante(dico_nul, 0) == False,
"Echec Test 6"

    dico_negatif = {
        "valeur_a": -2,
        "valeur_b": {
            "valeur_ba": 3,
            "valeur_bb": -19028,
            "valeur_bc": 0,
            "valeur_bd": -1039
        },
        "valeur_c": -10,
        "valeur_d": {
            "valeur_da": 2,
            "valeur_db": -1892
        },
        "valeur_e": {
            "valeur_ea": {
                "valeur_eaa": 2,
                "valeur_eab": {
                    "valeur_eaba": 20001
                },
                "valeur_eac": -2900
            },
            "valeur_eb": -29
        }
```

```
    }

    # T7 - On teste avec un dictionnaire complexe negatif avec
une valeur aberrante :
    assert alerte_valeur_aberrante(dico_negatif, -2) == True,
"Echec Test 7"

    # T8 - On teste avec un dictionnaire complexe negatif sans
une valeur aberrante :
    assert alerte_valeur_aberrante(dico_negatif, 900000) ==
False, "Echec Test 8"
```