BACCALAURÉAT

SESSION 2024

Épreuve de l'enseignement de spécialité

NUMÉRIQUE et SCIENCES INFORMATIQUES

Partie pratique

Classe Terminale de la voie générale

Sujet n°40

DURÉE DE L'ÉPREUVE : 1 heure

Le sujet comporte 4 pages numérotées de 1 / 4 à 4 / 4
Dès que le sujet vous est remis, assurez-vous qu'il est complet.

Le candidat doit traiter les 2 exercices.

EXERCICE 1 (10 points)

On considère des tables, c'est-à-dire des tableaux de dictionnaires ayant tous les mêmes clés, qui contiennent des enregistrements relatifs à des animaux hébergés dans un refuge.

Les attributs des enregistrements sont `'nom'`, `'espece'`, `'age'`, `'enclos'`.

Voici un exemple d'une telle table :

```
animaux = [ {'nom':'Medor', 'espece':'chien', 'age':5, 'enclos':2},
            {'nom':'Titine', 'espece':'chat', 'age':2, 'enclos':5},
            {'nom':'Tom', 'espece':'chat', 'age':7, 'enclos':4},
            {'nom':'Belle', 'espece':'chien', 'age':6, 'enclos':3},
            {'nom':'Mirza', 'espece':'chat', 'age':6, 'enclos':5}]
```

Programmer une fonction `selection_enclos` qui :

- prend en paramètres :
  - une table `animaux` contenant des enregistrements relatifs à des animaux (comme dans l'exemple ci-dessus),
  - un numéro d'enclos `num_enclos` ;
- renvoie une table contenant les enregistrements de `animaux` dont l'attribut `'enclos'` est `num_enclos`.

Exemples avec la table `animaux` ci-dessus :

```
>>> selection_enclos(animaux, 5)
[{'nom':'Titine', 'espece':'chat', 'age':2, 'enclos':5},
 {'nom':'Mirza', 'espece':'chat', 'age':6, 'enclos':5}]

>>> selection_enclos(animaux, 2)
[{'nom':'Medor', 'espece':'chien', 'age':5, 'enclos':2}]

>>> selection_enclos(animaux, 7)
[]
```

EXERCICE 2 (10 points)

On considère des tableaux de nombres dont tous les éléments sont présents exactement trois fois à la suite, sauf un élément qui est présent une unique fois et que l'on appelle « l'intrus ». Voici quelques exemples :

```
tab_a = [3, 3, 3, 9, 9, 9, 1, 1, 1, 7, 2, 2, 2, 4, 4, 4, 8, 8, 8]
#l'intrus est 7

tab_b = [8, 5, 5, 5, 9, 9, 9, 18, 18, 18, 3, 3, 3]
#l'intrus est 8

tab_c = [5, 5, 5, 1, 1, 1, 0, 0, 0, 6, 6, 6, 3, 8, 8, 8]
#l'intrus est 3
```

On remarque qu'avec de tels tableaux :

- pour les indices multiples de 3 situés strictement avant l'intrus, l'élément correspondant et son voisin de droite sont égaux,
- pour les indices multiples de 3 situés après l'intrus, l'élément correspondant et son voisin de droite - s'il existe - sont différents.

Ce que l'on peut observer ci-dessous en observant les valeurs des paires de voisins marquées par des caractères `^` :

```
[3, 3, 3, 9, 9, 9, 1, 1, 1, 7, 2, 2, 2, 4, 4, 4, 8, 8, 8]
 ^  ^     ^  ^     ^  ^     ^  ^     ^  ^     ^  ^
 0        3        6        9        12       15
```

Dans des tableaux comme celui ci-dessus, un algorithme récursif pour trouver l'intrus consiste alors à choisir un indice i multiple de 3 situé approximativement au milieu des indices parmi lesquels se trouve l'intrus.

Puis, en fonction des valeurs de l'élément d'indice i et de son voisin de droite, à appliquer récursivement l'algorithme à la moitié droite ou à la moitié gauche des indices parmi lesquels se trouve l'intrus.

Par exemple, si on s'intéresse à l'indice 12, on voit les valeurs 2 et 4 qui sont différentes : l'intrus est donc à gauche de l'indice 12 (indice 12 compris)

En revanche, si on s'intéresse à l'indice 3, on voit les valeurs 9 et 9 qui sont identiques : l'intrus est donc à droite des indices 3-4-5, donc à partir de l'indice 6.

Compléter la fonction récursive `trouver_intrus` proposée page suivante qui met en œuvre cet algorithme.

```
def trouver_intrus(tab, g, d):
    """Renvoie la valeur de l'intrus situé entre les indices g et d
    ↪
    dans le tableau tab où :
    tab vérifie les conditions de l'exercice,
    g et d sont des multiples de 3."""
    if g == d:
        return ...
    else:
        nombre_de_triplets = (d - g) // ...
        indice = g + 3 * (nombre_de_triplets // 2)
        if ...:
            return ...
        else:
            return ...
```

Exemples :

```
>>> trouver_intrus([3, 3, 3, 9, 9, 9, 1, 1, 1, 7,
        2, 2, 2, 4, 4, 4, 8, 8, 8], 0, 18)
7

>>> trouver_intrus([8, 5, 5, 5, 9, 9, 9, 18, 18, 18, 3, 3, 3],
                    0, 12)
8

>>> trouver_intrus([5, 5, 5, 1, 1, 1, 0, 0, 0,
        6, 6, 6, 3, 8, 8, 8], 0, 15)
3
```