BACCALAUREAT

SESSION 2022

Épreuve de l'enseignement de spécialité

NUMERIQUE et SCIENCES INFORMATIQUES

Partie pratique

Classe Terminale de la voie générale

Sujet n°25

DUREE DE L'EPREUVE : 1 heure

Le sujet comporte 2 pages numérotées de 1 / 3 à 3 / 3
Dès que le sujet vous est remis, assurez-vous qu'il est complet.

Le candidat doit traiter les 2 exercices.

EXERCICE 1 (4 points)

On considère des tables (des tableaux de dictionnaires) 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 `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 `table_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 (4 points)

On considère des tableaux de nombres dont tous les éléments sont présents exactement trois fois et à suivre, 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, 5, 5, 5]
#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, 5, 5, 5]
 ^  ^     ^  ^     ^  ^     ^  ^     ^  ^     ^  ^     ^  ^     ^
 0        3        6        9        12       15       18       21
```

Dans des listes comme celles 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.

Compléter la fonction ci-dessous 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 la liste 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, 5, 5, 5], 0, 21)
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
```

```
"""
Author: Pascal Padilla
Source: correction de l'exercice 1 du sujet 25 des épreuves pratiques NSI 2022
"""


def selection_enclos(dic_animaux: dict, num_enclos: int) -> dict:
    """
    Sélection dans un dictionnaire de données en fonction
    de l'identifiant de l'enclos

    Args:
        dic_animaux (dict): données avec au moins la clé 'enclos'
        num_enclos (int): identifiant de l'enclos à sélectionner

    Returns:
        dict: dictionnaire de données filtré par num_enclos

    Tests et exemples:
    >>> 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}]
    >>> selection_enclos(animaux, 5)
    [{'nom': 'Titine', 'espece': 'chat', 'age': 2, 'enclos': 5}, {'nom': 'Mirza', 'e
spece': 'chat', 'age': 6, 'enclos': 5}]
    >>> selection_enclos(animaux, 2)
    [{'nom': 'Medor', 'espece': 'chien', 'age': 5, 'enclos': 2}]
    >>> selection_enclos(animaux, 7)
    []
    """
    # resultat qui sera renvoyé en sortie
    # (tableau de dictionnaire(s))
    resultat_requete = []

    # parcourir tout le dictionnaire dans une boucle
    for entite in dic_animaux:

        # ajouter l'entité courante si l'identifiant correspond
        if entite['enclos'] == num_enclos:
            resultat_requete.append(entite)

    # renvoyer le résultat de la sélection
    return resultat_requete



# tests avec des affichages
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}]

print(selection_enclos(animaux, 5))
print(selection_enclos(animaux, 2))
print(selection_enclos(animaux, 7))


# tests avec des assertions
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}]
assert selection_enclos(animaux, 5) == [{'nom': 'Titine', 'espece': 'chat', 'age': 2
, 'enclos': 5}, {'nom': 'Mirza', 'espece': 'chat', 'age': 6, 'enclos': 5}]
assert selection_enclos(animaux, 2) == [{'nom': 'Medor', 'espece': 'chien', 'age': 5
, 'enclos': 2}]
assert selection_enclos(animaux, 7) == []
```

```
# tests avec doctest
from doctest import testmod
testmod()
```

```
"""
Author: Pascal Padilla
Source: correction de l'exercice 2 du sujet 25 des épreuves pratiques NSI 2022

Remarque:
    algorithme qui fonctionne sur diviser pour régner
    rapide car la taille de la zone à explorer est divisée par deux à chaque appel
    ressemble à la dichotomie
"""


def trouver_intrus(tab, g, d):
    '''
    Renvoie la valeur de l'intrus situé entre les indices g et d
    dans la liste tab où
    tab vérifie les conditions de l'exercice,
        g et d sont des multiples de 3.
    '''
    # affichage à décommenter pour 'voir' l'algo fonctionner
    # print(f'recherche: {g}..{d}\n  sous-tableau: {tab[g:d+1]}')

    # cas où la zone de recherche est ramenée à un seul élément
    #  -> l'intrus est identifié
    if g == d:
        return tab[g] # ou (c'est équivalent): return tab[d]

    else:
        # il y a autant de triplets que la longueur de la zone de
        # recherche divisée par 3
        nombre_de_triplets = (d - g) // 3
        # positionnement à l'indice situé au milieu ET
        #  correspondant au début d'un triplet
        indice = g + 3 * (nombre_de_triplets // 2)

        # cas de la zone inférieure à l'intrus
        # (l'élément courant et l'élément juste à droite sont égaux)
        if tab[indice] == tab[indice + 1] :
            # appel récursif avec recherche dans la zone après indice
            #  comme tab[indice] n'est pas l'intrus, on l'exclu de la
            #  zone de recherche à venir: g <- indice + 3
            return trouver_intrus(tab, indice + 3, d)
        else:
            # appel récursif avec recherche dans la zone avant indice
            #  comme tab[indice] est PEUT ÊTRE l'intrus, on NE l'exclu PAS
            #  de la zone de recherche à venir: d <- indice
            return trouver_intrus(tab, g, indice)


# ajout de cette condition pour des raisons de tests automatisés
if __name__ == '__main__':
    assert trouver_intrus([3, 3, 3, 9, 9, 9, 1, 1, 1, 7, 2, 2, 2, 4, 4, 4, 8, 8, 8,
5, 5, 5], 0, 21) == 7
    assert trouver_intrus([8, 5, 5, 5, 9, 9, 9, 18, 18, 18, 3, 3, 3], 0, 12) == 8
    assert trouver_intrus([5, 5, 5, 1, 1, 1, 0, 0, 0, 6, 6, 6, 3, 8, 8, 8], 0, 15) =
= 3
```