FEUILLE n°3 *Dictionnaire de fréquences et notion de complexité*

Exercice 1. Question de cours

Dans cet exercice, `liste` est une liste, `ensemble` est un ensemble et `dico` est un dictionnaire. Ces trois structures de données sont de taille N. Par exemple :

```
liste = ['Aatrox', 'Braum', 'Nunu', 'Braum', ... ]
ensemble = {'Aatrox', 'Braum', 'Nunu', 'Zed', ... }
dico = {'Aatrox':12 'Braum':7, 'Nunu':3, 'Zed':17, ... }
```

Préciser la complexité de chacune des instructions suivantes :

1.1. `champion = liste[2]`

1.2. `zed = max(ensemble)`

1.3. `mushroom = 'Teemo' in liste`

1.4. `liste.append('Teemo')`

1.5.
```
if 'Teemo' in dico.keys():
    print('ok')
```

1.6.
```
if 42 in dico.values():
    print('ok')
```

1.7.
```
for champ in dico.keys():
    print(champ)
```

1.8. `ensemble.add('Teemo')`

1.9. `longueur = len(ensemble)`

1.10. `copie = dico.copy()`

1.11. `dico['Teemo'] = 15`

1.12. `mushroom = 'Teemo' in ensemble`

1.13. `new_set = set(liste)` construit un nouvel ensemble à partir des éléments de `liste`

1.14. `new_set == ensemble`

1.15. `new_set = ensemble`

Exercice 2. Lecture de code et complexité

2.1. Compléter les tests dans le script ci-dessous

2.2. Déterminer la complexité de chacune des fonctions

2.3. Associer chaque fonction à sa description

```python
mes_notes = [17, 14, 10, 17, 19, 10, 17, 17, 14]

def fonction1(notes): # Complexité = ??
    res = notes[0]
    for note in notes:
        if res < note:
            res = note
    return res
assert fonction1(mes_notes) == ???

def fonction2(notes): # Complexité = ??
    res = dict()
    for note in notes:
        if note in res.keys():
            res[note]+= 1
        else:
            res[note] = 1
    return res
assert fonction2(mes_notes) == ???

def fonction3(notes): # Complexité = ??
    res = 0
    for note in notes:
        if note > 15:
            res+= 1
    return res
assert fonction3(mes_notes) == ???

def fonction4(notes): # Complexité = ??
    for i in range(len(notes)):
        for j in range(len(notes)):
            if i!=j and notes[i] == notes[j]:
                return True
    return False
assert fonction4(mes_notes) == ???
```

```
""" doc A
renvoie le nombre de notes
dans la liste
"""

""" doc B
renvoie le dictionnaire
de fréquences des notes
"""

""" doc C
vérifie si la liste
contient au moins deux
notes consécutives
identiques
"""

""" doc D
compte le nombre de bonnes
notes (notes > 15)
"""

""" doc E
renvoie la note la plus
grande de la liste
"""

""" doc F
renvoie la note la plus
petite de la liste
"""

""" doc G
vérifie si la liste
contient au moins deux
notes identiques
"""
```

Exercice 3. Représentation de la mémoire

```
def consonnes(chaine):
    les_voyelles = {'a', 'e', 'u', 'i', 'o', 'y'}
    res = []
    for lettre in chaine:
        if lettre not in les_voyelles :
            res.append(lettre)
            # Question 4.3
    return res

def nombre_repetitions(liste_lettres):
    nombre = 0
    lettre_precedente = None
    for lettre in liste_lettres :
        if lettre == lettre_precedente:
            nombre += 1
        lettre_precedente = lettre
    return nombre

def nombre_consonnes_repetees(mot):
    # Question 4.2
    les_consonnes = consonnes(mot)
    # Question 4.4
    return nombre_repetitions(les_consonnes)

chaine = 'accueillies '
# Question 4.1
n = nombre_consonnes_repetees (chaine)
print(n)
```

3.1. Donner la représentation de la mémoire (pile et tas) au premier passage à l'endroit indiqué `# Question 4.1`

3.2. Donner la représentation de la mémoire (pile et tas) au premier passage à l'endroit indiqué `# Question 4.2`

3.3. Donner la représentation de la mémoire (pile et tas) au premier passage à l'endroit indiqué `# Question 4.3`

3.4. Donner la représentation de la mémoire (pile et tas) au premier passage à l'endroit indiqué `# Question 4.4`

3.5. Déterminer la complexité des trois fonctions de ce script (on notera N la longueur de la structure de données passée en paramètre) et décrire, en une phrase, ce qu'elles font.

Récupérer le fichier `feuille3.py` disponible sur Célène.

Exercice 4. Fan club - la suite

J'ai réalisé un sondage auprès de mes amis leur demandant quel est leur groupe de musique favori et j'ai modélisé les résultats par un dictionnaire :

```
mon_sondage ={'Florent':'Trust', 'Celine':'SuperBus', 'Julien':'ACDC',
              'Denys':'ACDC', 'Caroline':'Trust', 'Gérard':'ACDC'}
```

4.1. Ecrire une fonction `dico_freq` qui prend un tel sondage en paramètre et qui renvoie le dictionnaire de fréquences des groupes de musique.

```
assert dico_freq(mon_sondage) == {'Trust': 2, 'ACDC': 3, 'SuperBus': 1}
```

4.2. Déterminer la complexité des deux fonctions `nombre_de_fans_de` et `top_groupe`.

```
def nombre_de_fans_de(sondage, groupe): # Complexité = ??
    """renvoie le nombre de fans du groupe"""
    nb = 0
    for gpe in sondage.values():
        if gpe == groupe:
            nb+=1
    return nb

assert nombre_de_fans_de(mon_sondage, 'Trust') == 2

def top_groupe(sondage): # Complexité = ??
    """renvoie le nom du groupe qui a le plus de fans"""
    groupe_au_top = None
    nb_fan_au_top = 0
    for groupe in sondage.values():
        if nombre_de_fans_de(sondage, groupe) >= nb_fan_au_top:
            nb_fan_au_top = nombre_de_fans_de(sondage, groupe)
            groupe_au_top = groupe
    return groupe_au_top

assert top_groupe(mon_sondage) == 'ACDC'
```

4.3. Proposez un nouveau code de la fonction `top_groupe` qui permettrait d'en améliorer l'efficacité. Préciser la complexité de cette nouvelle fonction.

Exercice 5. Petites bêtes On donne une liste contenant des informations sur des pokemons sous la forme d'un tuple (nom, type). Par exemple [1] :

```
mon_pokedex = [('Bulbizarre', 'Plante'),  ('Aeromite', 'Poison'),  ('Abo', 'Poison')]
```

5.1. Écrire une fonction `frequences_types` qui prend en paramètre une telle liste et qui retourne un dictionnaire dont les clefs sont les types et chaque valeur le nombre de pokémons de ce type dans la liste. Préciser la complexité de cette fonction.

```
assert frequences_types(mon_pokedex) == {'Plante': 1, 'Poison': 2}
```

5.2. Écrire une fonction `dico_par_types` qui prend en paramètre une telle liste et qui retourne un dictionnaire dont les clefs sont les types et chaque valeur l'ensemble des pokémons de ce type. Préciser la complexité de cette fonction.

```
assert dico_par_types(mon_pokedex) == {
        'Plante': {'Bulbizarre'}, 'Poison': {'Aeromite', 'Abo'} }
```

5.3. Écrire une fonction `type_le_plus_représenté` qui prend en paramètre une telle liste et qui retourne le type de pokemon qui est le plus représenté dans la liste. Préciser la complexité de cette fonction.

```
assert type_le_plus_représenté(mon_pokedex) == 'Poison'
```

5.4. En réalité, les pokemons peuvent avoir plusieurs types. Ainsi Bulbizarre est de type Plante mais également de type Poison. Je modifie ma modélisation de la façon suivante :

```
mon_pokedex = [('Bulbizarre', {'Plante', 'Poison'}),
  ('Aeromite', {'Poison', 'Insecte'}), ('Abo', {'Poison'})]
```

Proposez une version2 des fonctions précédentes qui prennent en paramètre cette nouvelle modélisation.

```
assert frequences_types_v2(mon_pokedex) == {'Plante': 1, 'Poison': 3, 'Insecte':1}
assert dico_par_types_v2(mon_pokedex) == { 'Plante': {'Bulbizarre',},
        'Poison': {'Bulbizarre', 'Aeromite', 'Abo'}, 'Insecte': {'Aeromite'}}
assert type_le_plus_représenté_v2(mon_pokedex) == 'Poison'
```

1. Pour les fans de Pokemons, ne râlez pas tout de suite : une modélisation plus "réaliste" arrive à la dernière question

Exercice 6. Où l'on retrouve Perrette et son troupeau

Dans le monde de Perrette, on modélise les troupeaux sous la forme d'un dictionnaire dont les clés sont le nom des animaux et les clefs leur nombre dans le troupeau. Par exemple, si Perrette possède dans son troupeau 14 veaux, 7 vaches et 42 poules, son troupeau sera modélisé de la façon suivante :

```
troupeau_de_perrette={'veau':14, 'vache':7, 'poule':42}
```

6.1. Compléter le code de la fonction `ajoute_animaux`. Préciser sa complexité.

```
def ajoute_animaux(troupeau, animal, nombre):  # Complexité = ??
        """
        Cette fonction ajoute des animaux dans le troupeau.
        Par exemple 'ajoute_animaux(troupeau, "Zébu", 4)' ajoute 4 Zébus
        au troupeau passé en paramètre
        Résultat : None
        """
        pass

tdp = {'veau':14, 'vache':7, 'poule':42}
ajoute_animaux(tdp, 'vache', 23)
assert tdp == {'veau':14, 'vache':30, 'poule':42}
ajoute_animaux(tdp, 'girafe', 2)
assert tdp == {'veau':14, 'vache':30, 'poule':42, 'girafe':2}
```

6.2. Compléter le code de la fonction `reunit_troupeaux`. Préciser sa complexité.

```
def reunit_troupeaux(troupeau1, troupeau2): # Complexité = ??
        """
        Cette fonction réunit deux troupeaux d'animaux
        Résultat : un nouveau dictionnaire qui est la réunion des deux troupeaux
        """
        pass

perrette = {'veau':14, 'vache':7, 'poule':42}
jean = {'vache':12, 'cochon':17, 'veau':3}
assert reunit_troupeaux(perrette, jean) == {'veau':17, 'vache':19,
        'poule':42, 'cochon':17}
```

Exercice 7. Occupations

On décrit la journée d'un individu sous la forme d'une liste de couples (Activité, durée en heures). Par exemple, voici la journée de lundi de Léa :

```
lundi_lea = [('Sommeil', 7.5), ('Repas', 0.25), ('Douche', 0.1),
        ('Transport', 0.75), ('Travail', 4), ('Repas', 1),
        ('Sommeil',0.5), ('Travail',4.5)]
```

7.1. En utilisant la même modélisation (une liste de tuples), décrire votre journée d'hier.

7.2. Écrire une fonction qui calcule le temps qu'une personne a passé à travailler pendant la journée ?[2]

7.3. Écrire une fonction qui détermine l'activité à laquelle il/elle a consacré le plus de temps ?

Exercice 8. Qui a mon livre ?

J[3]'ai prêté un livre à Alex. Alex l'a prêté à son tour, et ainsi de suite. C'est comme ça que j'ai perdu mon livre. J'ai donc décidé, avant de prêter un autre livre à Alex, de créer un dictionnaire, comme suit :

```
dico_prets = {'M Becker':'Alex'}
```

Les clés de ce dictionnaire sont des personnes, et la valeur associée à une personne est celle à qui elle a prêté le livre. Par exemple, quand Alex prête mon livre à Boris, on exécute l'instruction suivante.

```
dico_prets['Alex'] = 'Boris'
```

8.1. Quelle est alors la nouvelle valeur de `dico_prets` ?

Après plusieurs étapes, voici la valeur de `dico_prets` :

```
dico_prets ={'M Becker':'Alice', 'Alice':'Boris',
                        'Charlotte':'Denis', 'Boris':'Charlotte' }
```

8.2. Qui a eu le livre en troisième ? Qui a maintenant mon livre ?

2. Bien sûr, vous n'oubliez pas vos bonnes pratiques et vous codez avec méthode !
3. Dans cet exercice "Je" = "Monsieur Becker"

8.3. Compléter le code de la fonction suivante, qui indique qui a prêté le livre à une personne donnée.

```
def qui_a_prete(dico_prets, emprunteur):
        """
        Paramètres:
           - dico_prets est un dictionnaire dont les clés sont des personnes
             qui ont eu le livre, et la valeur associée est le nom de la
             personne à qui chacune a confié le livre.
          - emprunteur, le nom d'une personne
        Résultat : le nom de la personne qui a prêté le livre à 'emprunteur '
        Note: on suppose qu'emprunteur a effectivement reçu le livre et que
        chaque personne n'a eu le livre qu'une seule fois
        """
        pass

assert qui_a_prete({'M Becker':'Alex', 'Alex':'Boris'}, 'Alex') == 'M Becker'
```

8.4. Compléter le code de la fonction suivante, qui indique qui a maintenant le livre en sa possession. Vous n'oublierez pas de coder au moins un tests avant de coder le corps de la fonction. Indication : la personne qui a actuellement le livre en sa possession est celle qui n'apparaît pas comme clé dans le dictionnaire, puisqu'elle n'a pas encore passé le livre à quelqu'un.

```
def qui_a_maintenant(dico_prets):
        """
        Paramètre: dico_prets est un dictionnaire dont les clés sont des
        personnes qui ont eu le livre, et la valeur associée est le nom de la
        personne à qui chacune a confié le livre.
        Résultat : le nom de la personne qui a maintenant le livre
        Note: on suppose que chaque personne n'a eu le livre qu'une seule fois
        """
        pass
```

8.5. Compléter le code de la fonction suivante, qui indique comment le livre est passé d'une personne à une autre.

```
def intermediaires(dico_prets, depart, arrivee):
        """
        Paramètres:
           - dico_prets est un dictionnaire dont les clés sont des personnes
             qui ont eu le livre, et la valeur associée est le nom de la
             personne à qui chacune a confié le livre.
          - 'depart' et 'arrivee' sont deux personnes
        Résultat : la liste des personnes qui ont eu le livre entre
        depart (compris) et arrivee (non compris), dans l'ordre où elles se
        sont passées le livre.
        Note: on suppose que chaque personne n'a eu le livre qu'une seule fois,
        et que 'depart' et 'arrivee' l'ont eu, et que 'depart' l'a eu avant 'arrivee'
```

```
        """
        pass

d = {'M Becker ':'Alice', 'Alice': 'Boris', 'Charlotte':'Denis', 'Boris':'Charlotte'}
assert intermediaires(d,'M Becker','Charlotte') == ['M Becker', 'Alice', 'Boris']
```