FEUILLE n°4 *Dictionnaire de fréquences et notion de complexité*

Exercice 1. Un petit bout de cours - La méthode get dans un dictionnaire Facultatif

En python, les dictionnaires possèdent une méthode `get` qui peut prendre deux paramètres. Elle fonctionne de la façon suivante :

`dico.get(cle, valeur_autre)` renvoie la valeur associée à `cle` si `cle` existe comme clé du dictionnaire, sinon elle renvoie `valeur_autre`

Par exemple :

```
dico = {'Aatrox':12 'Braum':7, 'Nunu':3, 'Zed':17}
>>> dico.get('Braum', 42)
7
>>> dico.get('Teemo', 42)
42
```

1.1. Préciser la valeurs des variables `a` , `b` et `dico` à l'issue du script suivant :

```
dico = {'Poison': 2, 'Eau': 3}
a = dico.get('Dragon', 1)
b = dico.get('Eau', 1)
dico['Eau'] = dico.get('Eau', 0) + 1
dico['Dragon'] = dico.get('Dragon', 0) + 1
```

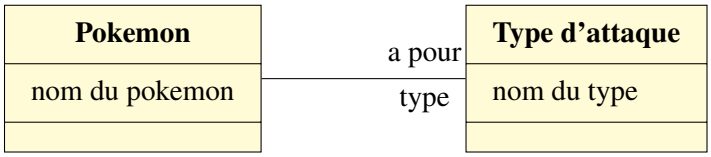
1.2. Dans cette question, un pokedex est modélisé par une liste de couples `(nom_pokemon, type_attaque)` . En utilisant la méthode get, compléter le code de la fonction `frequences_types` .

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

def frequences_types(pokedex):
    """ renvoie le dictionnaire de fréquences des types d'attaques des pokemons
    présents dans le pokedex"""
    dico = dict()
    for (_, type_attaque) in pokedex:
        ????
    return dico

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

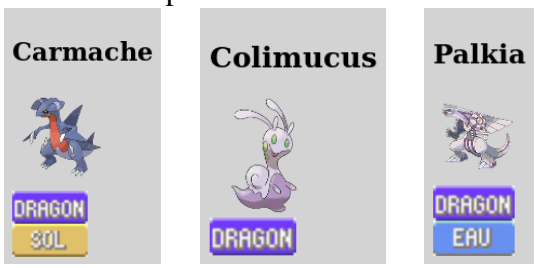
Exercice 2. Choix de modélisation et complexité — Obligatoire

Récupérer le fichier feuille4.py sur Célène

Voici le MCD d'un pokedex



Le pokedex de Anakin



Le pokedex de Romain

Anakin a trouvé trois façons différentes de modéliser les données en python.

2.1. Préciser comment serait implémenté le pokedex de Romain dans chacune des trois versions.

2.2. Choisir une version et compléter le code des quatre fonctions dont on donne le profil à la fin de l'exercice. Vous préciserez la complexité de chacune de vos fonctions.

```python
# version 1
pokedex_anakin_v1 = {
    ('Carmache', 'Dragon'), ('Carmache', 'Sol'),
    ('Colimucus', 'Dragon'), ('Palkia', 'Dragon'),
    ('Palkia', 'Eau')}

# version 2
pokedex_anakin_v2 = {
    'Carmache': {'Dragon','Sol'},
    'Colimucus': {'Dragon'},
    'Palkia': {'Dragon', 'Eau'}}

# version 3
pokedex_anakin_v3 = {
    'Dragon': {'Carmache','Colimucus', 'Palkia'},
    'Sol': {'Carmache'},
    'Eau': {'Palkia'}}
```

2.3. Choisir une autre version et écrire à nouveau les quatre fonctions demandées, toujours en précisant leur complexité.

2.4. **facultatif** Terminer le travail avec la dernière version

Complexités	`appartient()`	`toutes_les_attaques()`	`nombre_de()`	`ajoute()`
Avec la version 1				
Avec la version 2				
Avec la version 3				

```python
def appartient(pokemon, pokedex): # O(??)
    """ renvoie True si pokemon (str) est présent dans le pokedex """
    pass
assert not appartient("Rocaillou", pokedex_anakin)

def toutes_les_attaques(pokemon, pokedex): # O(??)
    """
    param: un pokedex et le nom d'un pokemon (str) qui appartient au pokedex
    resultat: renvoie l'ensemble des types d'attaque du pokemon passé en paramètre
    """
    pass
assert toutes_les_attaques("Palkia", pokedex_anakin) == {'Eau', 'Dragon'}

def nombre_de(attaque, pokedex): # O(??)
    """
    param: un pokedex et un type d'attaque (str)
    resultat: renvoie le nombre de pokemons de ce type d'attaque
    dans le pokedex
    """
    pass
assert nombre_de("Dragon", pokedex_anakin) == 3

def ajoute(pokemon, attaque, pokedex):  # O(??)
    """
    Ajoute au pokedex le pokemon (str) qui a pour un seul type 'attaque' (str)
    dans le pokedex
    resultat : rien
    """
    pass
copie = copy.deepcopy(pokedex_anakin)
ajoute("Sancoki", "Eau", copie)
assert copie == ???
```

2.5. Compléter les codes des deux fonctions suivantes qui permettent de transformer une version en une autre

```python
def v1_to_v2(pokedex_v1):
    """ param: prend en paramètre un pokedex version 1
        renvoie le même pokedex mais en version 2 """
    pass
assert v1_to_v2(pokedex_anakin_v1) == pokedex_anakin_v2

def v2_to_v3(pokedex_v2):
    """ param: prend en paramètre un pokedex version2
        renvoie le même pokedex mais en version3 """
    pass
assert v2_to_v3(pokedex_anakin_v2) == pokedex_anakin_v3
```

Exercice 3. Représentation de la mémoire — Obligatoire

3.1. Donner une représentation de la mémoire (pile et tas) à la fin de la ligne passage à l'endroit indiqué et préciser l'affichage obtenu à l'issue du script.

```
def ajoute_42(liste):
    liste.append(42)
    # ICI
    return liste

liste1 = [7, 1, 5]
liste2 = ajoute_1(liste1)
print(liste1) # A PRECISER
print(liste2) # A PRECISER
```

3.2. Donner une représentation de la mémoire (pile et tas) au passage à l'endroit indiqué et préciser l'affichage obtenu à l'issue du script.

```
def ajoute_42(liste):
    res = liste
    res.append(42)
    # ICI
    return res

liste1 = [7, 1, 5]
liste2 = ajoute_42(liste1)
print(liste1) # A PRECISER
print(liste2) # A PRECISER
```

3.3. Donner une représentation de la mémoire (pile et tas) au passage à l'endroit indiqué et préciser l'affichage obtenu à l'issue du script.

```
def ajoute_42(liste):
    liste.append(42)
    res = liste
    liste.remove(42)
    # ICI
    return res

liste1 = [7, 1, 5]
liste2 = ajoute_42(liste1)
print(liste1) # A PRECISER
print(liste2) # A PRECISER
```

3.4. Donner une représentation de la mémoire (pile et tas) au passage à l'endroit indiqué et préciser l'affichage obtenu à l'issue du script.

```
def ajoute_42(liste):
    res = liste.copy()
    res.append(42)
    # ICI
    return liste

liste1 = [7, 1, 5]
liste2 = ajoute_42(liste1)
print(liste1) # A PRECISER
print(liste2) # A PRECISER
```

Exercice 4. Lecture de code et complexité - Caractères en double **Facultatif**

Anakin doit écrire une fonction `caracteres_en_double` qui prend une chaîne de caractères en paramètre et qui renvoie l'ensemble des caractères qui apparaissent au moins 2 fois dans la chaîne. Il est très fier de ce qu'il a fait : cette fois ci, il n'a pas oublié la documentation, il a pensé à écrire des tests, et il a même découpé son code pour que ce soit plus lisible !

```
#Version 1
def nombre_apparition(chaine, caractere):
    """ renvoie le nombre d'occurrence du caractère dans la chaine """
    cpt=0
    for char in chaine:
        if caractere == char:
            cpt=cpt+1
    return cpt
assert nombre_apparition("yololo", "o") == 3
assert nombre_apparition("yololo", "a") == 0

def caracteres_en_double_v1(chaine):
    """ renvoie tous les caractères de la chaine qui apparaissent au moins 2 fois """
    caracteresRepetees = set()
    for caractere in chaine:
        if nombre_apparition(chaine, caractere) > 1:
            caracteresRepetees.add(caractere)
    return caracteresRepetees
assert caracteres_en_double_v1("May the force be with you") == {'o','y',' ','h','e','t'}
```

Et pourtant, Obiwan lui a dit qu'il pouvait faire mieux. Aidons Anakin à améliorer son programme.

4.1. On note N la taille de la chaîne de caractères. Déterminer la complexité de la fonction `caracteres_en_double_v1` proposée par Anakin.

4.2. Après plusieurs heures de travail, Anakin propose finalement deux autres programmes, mais il ne sait pas lequel choisir. Pouvez-vous l'aider ? Argumentez votre réponse.

```
#Version 2
def dico_freq(chaine):
    """ renvoie le dictionnaire des de fréquence des caractères de chaine """
    dico = dict()
    for char in chaine:
        if char in dico.keys():
            dico[char]+=1
        else:
            dico[char]=1
    return dico
assert dico_freq("yololo") == {'y':1, 'o':3, 'l':2}
```

```python
def caracteres_en_double_v2(chaine):
    """ renvoie tous les caractères de la chaine qui apparaissent au moins 2 fois """
    caracteresRepetees = set()
    dictionnaire_freq = dico_freq(chaine)
    for (caractere, frequence) in dictionnaire_freq.items():
        if frequence > 1:
            caracteresRepetees.add(caractere)
    return caracteresRepetees
assert caracteres_en_double_v2("May the force be with you") == {'o','y',' ','h','e','t'}
```

```python
#Version 3
def caracteres_en_double_v3(chaine):
    """ renvoie tous les caractères de la chaine qui apparaissent au moins 2 fois """
    dejaVu = set()
    caracteres_repetes = set()
    for caractere in chaine:
        if caractere not in dejaVu:
            dejaVu.add(caractere)
        else:
            caracteres_repetes.add(caractere)
    return caracteres_repetes
assert caracteres_en_double_v3("May the force be with you") == {'o','y',' ','h','e','t'}
```

Exercice 5. Résolution d'un problème algorithmique — Facultatif

On est à la fin d'un week-end entre amis, et on cherche à faire les comptes suite aux dépenses de chacun.

Par exemple, plusieurs amis se sont retrouvés pour un week-end de mai. Ce week-end là, Pierre a acheté du pain pour 12 euros, Paul a dépensé 100 euros pour les pizzas, Pierre a payé l'essence et en a eu pour 70 euros, Marie a acheté du vin pour 15 euros, Paul a aussi acheté du vin et en a eu pour 10 euros, Anna n'a quant à elle rien acheté.

Un peu plus tard au mois de juin, les amis ont organisé un nouveau week-end auquel Paul n'a pas pu participer. En revanche Béatrice et Sasha se sont jointes au groupe. Pour les comptes, Pierre a acheté du fromage pour 15 euros et du pain pour 12 euros. Marie a acheté le vin pour 20 euros et les glaces pour 34 euros. Anna a payé les pizza et en a eu pour 52 euros. Béatrice a payé 8 euros de pistaches pour l'apéro. Et Pierre a finacé la location du film pour 8 euros et les pop corn pour 3 euros.

5.1. Dans le tableau suivant, on a récapitulé les dépenses de chacun pour le week-end de mai. Indiquez combien chacun devra verser ou recevoir.

Pierre	Paul	Marie	Anna
12 70 10	100	15	

5.2. Même question pour le week-end du mois de juin.

Pierre	Marie	Anna	Béatrice	Sasha
15 12 8 3	20 34	52	8	

5.3. Écrire une fonction `afficheBilan(sdd)` qui prend en paramètre une structure de données qui représente un week-end, et qui, pour chaque personne qui a participé au week-end, affiche "X doit payer Y euros" ou "X doit recevoir Y euros" selon le cas.

Avant de vous lancer dans l'écriture du code, réfléchissez ! Comment résoudre ce problème "à la main" ? Quelles structures de données je vais utiliser ? Quelles micro-fonctions me seront nécessaires ? Et n'oubliez pas les bonnes pratiques et utilisez une "bonne" méthodologie.

Exercice 6. Nom de code A005150 **Facultatif** 

6.1. Compléter le code de la fonction suivante :

```
def suivante(liste):
    """
    param: une liste de nombres entiers positifs qui modélise un terme
    d'une suite suite audioactive
    renvoie une nouvelle liste qui modélise le terme suivant de la suite audioactive
    """
    nouvelleListe = []
    current_element = None
    for nombre in liste:
        if nombre == current_element:
            nouvelleListe[-2]+=1
        else:
            current_element = nombre
            nouvelleListe.extend([1,nombre])
    return nouvelleListe

assert  suivante([1]) == [1, 1]
assert  suivante([4, 2]) == ???
assert  suivante([[1, 1, 1]) == ???
assert  suivante([2, 5, 5, 5, 5]) == ???
assert  suivante([[3, 1, 1, 1, 1, 2, 2, 2,]) == ???
```

La suite de Conway est une suite "audioactive" dont le premier terme est 1. Ainsi, à partir de la liste `[1]`, on peut appliquer la fonction précédente (donc la fonction `suivante`) pour obtenir la séquence de Conway, où chaque ligne se déduit de la précédente par `suivante` :

etape 1	1
etape 2	1 1
etape 3	2 1
etape 4	1 2 1 1
etape 5	1 1 1 2 2 1
etape 6	3 1 2 2 1 1

6.2. Pour cette question, un ordinateur est inutile, mais quelques neurones seront sans doute nécessaires

a) Lisez à voix haute cette séquence.

b) Donnez les termes de la suite de Conway aux étapes 7 et 8.

c) Un chiffre 4 peut-il apparaître dans cette séquence ?

d) Peut-on avoir le motif 333 dans une ligne de cette séquence ?

Pour les questions suivantes, imaginez un algorithme et implémentez-le en python pour répondre aux questions suivantes :

6.3. Existe-t-il une ligne de plus de 131 124 nombres dans cette séquence ? Si oui, à partir de quelle étape ?

6.4. Le motif 1322112111 3122113121113 apparaît-il dans cette séquence ? Si oui, à quelle étape ?