FEUILLE N°2 *Ensembles et dictionnaires*

Exercice 1. Questions de cours

1.1. Par chacune des structures de données suivantes, préciser son type, sa taille et le type de ses éléments.

```
a = {"Abo", "Carapuce", "Rattata", "Abo"}
b = ["Serpent", "Minitortue", "Souris"]
c = [("Abo", 9), ("Carapuce", 52), ("Rattata", 25)]
d = {"Abo": True, "Carapuce": False, "Rattata": True, "Draco": True}
```

1.2. Par chacune des variables suivantes, préciser si elle est mutable ou non mutable.

```
a = "Abo"
b = {"Abo", "Carapuce", "Rattata", "Abo"}
c = ("Abo", 9)
d = [("Carapuce", 52), ("Rattata", 25)]
e = {"Abo": True, "Carapuce": False, "Rattata": True}
```

1.3. Chacune des lignes suivantes comporte une erreur : trouver laquelle

```
1 a = "Abo
2 b = {"Abo", "Carapuce", "Rattata", "Abo"]
3 c = {[1, 6], [4, 5, 7]}
4 d = {"Abo": True, "Carapuce": False, "Rattata": True, "Draco"}
```

Exercice 2. Questions de cours encore

On modélise le catalogue d'un magasin par un dictionnaire dont les clefs sont les articles (str) et la valeur leur prix en euros. Et on modélise l'ensemble de ses employés par un ensemble de str.

```
catalogue_magasin = {'stylo': 1.5, 'gomme': 0.5}
employes_magasin = {'Anna', 'Viktor', 'Rafael'}
```

2.1. Quelle instruction permet d'ajouter au catalogue des ciseaux coûtant 3,50 € ?

2.2. Quelle instruction permet de modifier le prix du stylo pour le passer à 2 € ?

2.3. Quelle instruction permet d'ajouter `"Tom"` comme employé du magasin ?

Exercice 3. Représentation de la mémoire et lecture de code

3.1. Donner une représentation de la mémoire juste avant de quitter la ligne 10.
Rappel : on entre dans une fonction uniquement quand elle est appelée

```
1  def ajoute_pika(un_pokedex):
2      """ ajoute Pikachu (6 kg) au pokedex passé en paramètre
3          resultat : Rien, aucun résultat ici : cette fonction ne renvoie rien
4      """
5      un_pokedex["Pikachu"] = 6 # ICI question 3.2
6
7  mon_pokedex = {"Abo":6.9, "Onix":21}
8  mon_pokedex["Onix"] = 210
9  mon_pokedex["Paras"] = 5.4  # ICI question 3.1
10 ajoute_pika(mon_pokedex)
11 print(mon_pokedex) # à préciser
```

3.2. Représenter l'état de la mémoire juste avant de quitter la ligne 6 dans le script précédent. Préciser l'affichage provoqué par la dernière ligne.

3.3. Représenter l'état de la mémoire au premier passage à la ligne indiquée (juste avant de quitter la ligne) , puis préciser l'affichage provoqué par la dernière ligne.

```
1  def ajoute_pika(pokedex):
2      """
3      ajoute Pikachu (poids 6 kg) au pokedex passé en paramètre si la limite
4      de poids n'est pas dépassée
5      resultat : True si l'ajout s'est fait, False sinon
6      """
7      ajout = False
8      poids = poids_total(pokedex)
9      if poids <= 500:
10         pokedex["Pikachu"] = 6
11         ajout = True
12     return ajout
13
14 def poids_total(dico):
15     somme = 0
16     for poids in dico.values():
17         somme = somme + poids  # ICI  question 3.3
18     return somme
19
20 mon_pokedex = {"Abo":6.9, "Stari":34.5}
21 ok = ajoute_pika(mon_pokedex)
22 print(ok, mon_pokedex) # à préciser
```

Exercice 4. Perrette et son troupeau

 L'objectif de cet exercice est de vous faire écrire quelques micro-fonctions manipulant des dictionnaires en faisant attention à faire le **bon** choix de boucle.
Récupérer le fichier `feuille2.py` disponible sur Célène.

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}
```

4.1. Décrire les deux troupeaux suivants :

```
troupeau_de_jean = {'vache':12, 'cochon':17, 'veau':3}
troupeau_vide = dict()
```

4.2. Compléter le code de la fonction `total_animaux` :

```
def total_animaux(troupeau):
    """
    Résultat : renvoie le nombre total (int) d'animaux
    dans le troupeau
    """
    pass

assert total_animaux(troupeau_de_perrette) == 63
assert total_animaux(troupeau_de_jean) == 32
assert total_animaux(troupeau_vide) == 0
```

4.3. Compléter le code de la fonction `tous_les_animaux` :

```
def tous_les_animaux(troupeau):
    """
    Résultat : renvoie l'ensemble des noms des animaux
    du troupeau
    """
    pass

assert tous_les_animaux(perrette) == {'veau', 'vache', 'poule'}
assert tous_les_animaux(jean) == {'veau', 'vache', 'cochon'}
assert tous_les_animaux(vide) == set()
```

4.4. Compléter le code de la fonction `specialisé` :

```
def specialisé(troupeau):
    """
    Résultat : vérifie si le troupeau contient 30 individus
    ou plus d'un même type d'animal

    """
    pass

assert specialisé(troupeau_de_perrette)
assert not specialisé(troupeau_de_jean)
assert not specialisé(troupeau_vide)
```

4.5. Compléter le code de la fonction `le_plus_representé` :

```
def le_plus_representé(troupeau):
    """
    Résultat : renvoie le nom de l'animal qui a le plus d'individus
    dans le troupeau et renvoie None si le troupeau est vide

    """
    pass

assert le_plus_representé(troupeau_de_perrette) == "poule"
assert le_plus_representé(troupeau_de_jean) == "cochon"
assert le_plus_representé(troupeau_vide) is None
```

4.6. Compléter le code de la fonction `reunion_troupeau` :

```
def quantite_suffisante(troupeau):
    """
    Résultat : vérifie si chaque type d'animal présent dans le troupeau
    contient au moins 5 individus
    """
    pass

assert quantite_suffisante(troupeau_de_perrette)
assert not quantite_suffisante(troupeau_de_jean)
assert quantite_suffisante(troupeau_vide)
```

Exercice 5. Qu'est ce qu'on mange ce soir ?

Je veux écrire des fonctions pour m'aider à gérer les repas de la semaine.
Voici ma modélisation :

recette
Non temps de préparation et cuisson difficulté

```
lundi = {'Carottes rapées': (15, 1),
         'Nouilles au beurre':(20, 2),
         'Mousse au chocolat':(30, 7)}
```

```
mardi = {'Soupe': (40, 3), 'Steak':(20, 4), 'Fromage':(1, 0)  }
mercredi = {'Pizza': (60, 7), 'Omelette':(15, 3), 'Pomme':(1, 0)}
```

5.1. Compléter le code de la fonction `recette_la_plus_facile` :

```
def recette_la_plus_facile(menu):
    """
    Résultat : ...

    """
    difficulte_min = None
    recette_facile = None
    for (recette, ( _ , difficulte))  in menu.items():
        if difficulte_min is None or difficulte_min > difficulte:
            difficulte_min = difficulte
            recette_facile = recette
    return recette_facile


assert recette_la_plus_facile(lundi) ...
assert recette_la_plus_facile(mardi) ...
assert recette_la_plus_facile(jeudi) ...
```

5.2. Compléter le code de la fonction `temps_total_de_preparation` :

```
def temps_total_de_preparation(menu):
    """
    Résultat : renvoie le temps total de préparation des recettes du menu
    """
    pass

assert temps_total_de_preparation(lundi) == 65
assert temps_total_de_preparation(mardi) == 61
assert temps_total_de_preparation(jeudi) == 0
```

5.3. Compléter le code de la fonction `menu_de_chef` :

```
def menu_de_chef(menu):
    """
    Résultat : vérifie si le menu contient au moins une recette
    difficile (difficulté > 5)

    """
    pass

assert menu_de_chef(lundi)
assert not menu_de_chef(mardi)
assert not menu_de_chef(jeudi)
```

5.4. Compléter le code de la fonction `recette_la_plus_longue` :

```
def recette_la_plus_longue(menu):
    """
    Résultat : le nom de la recette qui demande le plus de temps
    de préparation et cuisson

    """
    pass


assert recette_la_plus_longue(lundi) == 'Mousse au chocolat'
assert recette_la_plus_longue(mardi) == 'Soupe'
assert recette_la_plus_longue(jeudi) is None
```

📎 Rappel de méthodologie pour écrire le code d'une fonction

1. Je choisis des noms appropriés pour la fonction et ses paramètres. Je commence à écrire sa documentation. Je ne mets pas d'autre code que **pass**
2. J'écris des tests à partir d'exemples biens choisis
3. Je vérifie que les tests que j'ai écrits échouent
4. Je complète le code de la fonction jusqu'à ce que tous mes tests passent avec succès.

Exercice 6. Fan-club J'ai réalisé un sondage auprès de mes amis leur demandant quel est leur groupe de musique favori. J'ai modélisé les résultats par un dictionnaire :

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

6.1. Donner un exemple de dictionnaire qui pourrait modéliser les résultats d'un sondage auprès de vos amis.

6.2. Écrire une fonction `nombre_de_fans_de` qui, à partir d'un tel dictionnaire et d'un nom de groupe, renvoie le nombre de fans de ce groupe.

Par exemple, `nombre_de_fans_de(mon_sondage, 'Trust')` doit donner `2`

6.3. Écrire une fonction `fans_de` qui, à partir d'un tel dictionnaire et d'un nom de groupe, renvoie l'ensemble des fans de ce groupe.

Par exemple, `fans_de(mon_sondage, 'Trust')` doit donner `{'Florent', 'Caroline'}`

6.4. Écrire une fonction `tous_les_groupes` qui, à partir d'un tel dictionnaire renvoie tous les groupes de musique préférés qui apparaissent dans le sondage (je vous laisse choisir la structure de données adaptée).

Par exemple, `tous_les_groupes(mon_sondage)` doit donner `'Trust'`, `'SuperBus'` et `'ACDC'`.

6.5. Écrire [1] une fonction `top_groupe` qui renvoie le nom du groupe qui a le plus de fans.

Par exemple, `top_groupe(mon_sondage)` doit donner `'ACDC'`

1. Il s'agit, dans un premier temps, d'écrire un code "propre", sans redondance de code. Ensuite, vous pouvez essayer de trouver la "meilleure" fonction en terme de performance. Nous verrons en cours, la semaine prochaine, comment écrire cette fonction de manière plus efficace. (Quel Teaser !)

Exercice 7. Des supers héros

On dispose de dictionnaires dont les clefs sont le nom de personnages, et les valeurs des tuples contenant la force (int), l'intelligence (int) et la description (str) de ce personnage.

Voici un extrait d'un exemple d'un tel dictionnaire :

```
avengers = {
    'Spiderman': (5, 5, 'araignée à quatre pattes'),
    'Hulk': (7, 4, "Grand homme vert"),
    'Agent 13': (2, 3, 'agent 13'),
    'M Becker': (2, 6, 'expert en graphe'), ...
    }
```

7.1. Sans utiliser d'ordinateur, compléter le tableau suivant :

description	expression	valeur
Force de Hulk	`avenger['Hulk'][0]`	7
		`'araignée a quatre pattes'`
	`avenger['Drax'][2]`	`'Benêt tatoué'`
Intelligence de Gamora		4
		`(2, 3, 'agent 13')`

Pour les questions suivantes, une bonne méthodologie et des bonnes pratiques sont de rigueur.

7.2. Écrivez une fonction `intelligence_moyenne` qui prend en paramètre un tel dictionnaire et qui renvoie la valeur moyenne de l'intelligence de tous les personnages.

Remarque et conseil

Pour écrire vos tests, utilisez un jeu de données limité dans lequel vous vous attardez uniquement sur ce qui est important à ce moment là.. Par exemple, ici, la description importe peu.

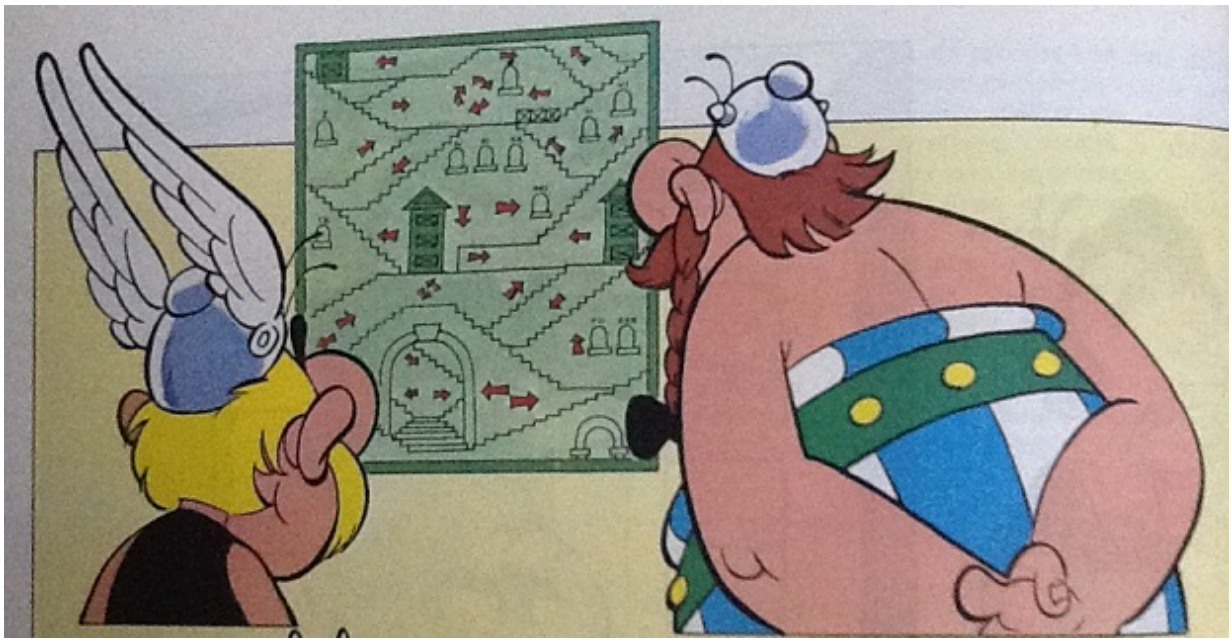
```
# dictionnaires bidons pour faire des tests :
exemple2 = {'a':(1,1,'a'), 'b':(3,9,'b'), 'c':(7,2,'c')}
assert intelligence_moyenne(exemple2) == 4
exemple3 = {'a':(1,1,'a'), 'b':(3,9,'b'), 'd':(4,4,'d')}
assert abs(intelligence_moyenne(exemple3)-14/3) <= 0.01
```

7.3. Écrivez une fonction `kikelplusfort` qui prend en paramètre un tel dictionnaire et qui renvoie le nom du personnage le plus fort.

7.4. Écrivez une fonction `combienDeCretins` qui prend en paramètre un tel dictionnaire et qui renvoie le nombre de personnages dont l'intelligence est strictement inférieure à la moyenne.

Exercice 8. La maison qui rend fou

Dans *les douze Travaux d'Astérix*, le huitième des douze travaux consiste à obtenir le laissez-passer A-38 dans la « maison qui rend fou » (ou mqrf). La maison qui rend fou est un bâtiment bureaucratique de plusieurs étages organisé en dépit de toute logique, où le personnel (incluant quelques fous), redirige Astérix et Obélix d'un guichet à l'autre afin de réunir la totalité des formulaires nécessaires pour obtenir le laissez-passer A-38, *dixit wikipedia*.



Nous allons représenter cette maison qui rend fou par un dictionnaire `mqrf`, dont les clefs sont les noms des guichetièr·es, et les clefs sont soit un nom de guichetièr·e, soit `None`.

Ainsi, chaque valeur `mqrf[guichet_courant]` de ce dictionnaire correspond à la réponse qu'on obtient au guichet_courant de la maison qui rend fou. Si `mqrf[guichet_courant]` est `None`, victoire ! le guichet visité délivre le formulaire A-38. Si `mqrf[guichet_courant]` est une chaîne `aller_voir`, alors le guichet visité nous renvoie vers le guichet `aller_voir`.

Exemples de "maison qui rend fou" :

```
mqrf1 = {"Abribus":"Astus", "Jeancloddus":"Abribus",
    "Plexus":"Gugus", "Astus":None, "Gugus":"Plexus",
    "Saudepus":None }

mqrf2 = {"Abribus":"Astus", "Jeancloddus":None,
        "Plexus":"Saudepus", "Astus":"Gugus",
        "Gugus":"Plexus", "Saudepus":None }
```

8.1. Dans la `mqrf1`, de quel guichet on obtient le formulaire A-38 si l'on s'adresse d'abord au guichet de `"Abribus"` ? Dans la `mqrf2`, de quel guichet on obtient le formulaire A-38 si l'on s'adresse d'abord au guichet de `"Abribus"` ?

8.2. Écrire une fonction qui prend en entrée une maison qui rend fou et un `guichet_de_depart`, et qui indique de quel guichet finira par donner le formulaire si l'on commence par s'adresser au guichet `guichet_de_depart`.

```
def quel_guichet(mqrf, guichet_de_depart):
    """
    paramètres :
      - mqrf est une maison qui rend fou
      - guichet_de_depart est le nom d'un guichet de la mqrf
    résultat : le nom du guichet qui finit par donner le formulaire
    """
    pass

assert quel_guichet(mqrf1, "Saudepus") == "Saudepus"
assert quel_guichet(mqrf3, "Abribus") == "Jeancloddus"
```

8.3. Modifier la fonction précédente de façon à ce qu'elle renvoie un tuple contenant le nom du guichet qui finira par donner le formulaire ainsi que le nombre de guichets visités pour y parvenir.

```
assert quel_guichet(mqrf1, "Saudepus") == ("Saudepus", 1)
assert quel_guichet(mqrf3, "Abribus") == ("Jeancloddus", 5)
```

8.4. Que se passe-t-il si je m'adresse abord au guichet de `"Abribus"` dan la `mqrf3` ?

```
mqrf3 = {"Abribus":"Astus", "Jeancloddus":None,
         "Plexus":"Jeancloddus", "Astus":"Gugus",
         "Gugus":"Plexus", "Saudepus":"Bielorus"}
```

8.5. Panoramix a remarqué quelque chose : si l'on tourne en rond plus longtemps que **len**`(mqrf)`, c'est qu'on ne trouvera jamais la sortie (autant abandonner).
Écrire une fonction `quel_guichet_v2` qui prend en argument un dictionnnaire `mqrf` représentant la maison qui rend fou et un nom de `guichet_de_depart`, et renvoie un booléen indiquant s'il est possible d'obtenir le formulaire A-38 dans la maison `mqrf` en commençant par s'adresser au `guichet_de_depart`.