FEUILLE n°7 *Algorithmes de tri*

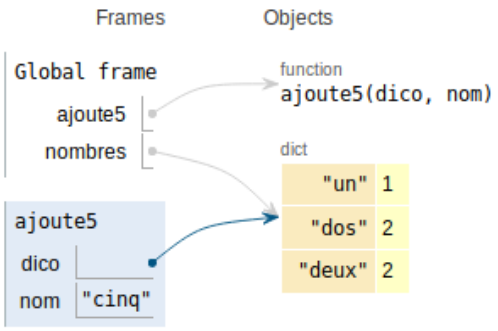
Exercice 1. Question de cours

```
def mystere(dictionnaire):
    """
    paramètre : un dictionnaire dont les clés sont des str et
    les valeurs des nombres.
    renvoie ???
    """
    def get_valeur(truc):
        return dictionnaire[truc]
    return sorted(dictionnaire, key = get_valeur)

assert mystere({'A': 5, 'C': 4, 'Z': 7, 'Y': 3}) == ???
```

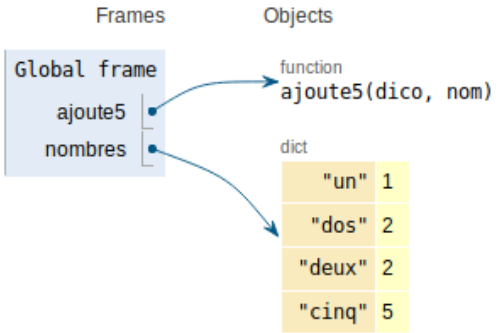
Quel est le type du paramètre `truc` ? Compléter le code de la fonction

Exercice 2. Représentation de la mémoire

2.1. Proposer un script qui provoque la représentation de la mémoire ci-contre. Préciser à quel endroit dans le script on se trouve.



2.2. Proposer un code pour la fonction `ajoute5` sachant qu'à la sortie de la fonction, l'état de la mémoire est le suivant.



Exercice 3. Tri par sélection du minimum

```
def tri_selection (liste):
    """
    param: une liste de nombres
    renvoie une liste triée contenant tous les éléments de 'liste'
    """
    liste_copy = liste.copy()
    res = []
    while len(liste_copy)!=0:
        # ICI
        element = min(liste_copy)
        res.append(element)
        liste_copy.remove(element)
    return res
```

3.1. On exécute `tri_selection([4, 5, 1, 3, 1])`. Donner la valeur de `res` à chaque passage par la ligne indiquée.

	valeurs de res
tour 0	
tour 1	
tour 2	
tour 3	
tour 4	

3.2. Compléter l'invariant de boucle :

```
Au début du tour de boucle numéro ??, res est ??
qui contient les ?? plus petits éléments de liste.
```

3.3. Quelle est la complexité de la fonction `tri_selection` ?

Exercice 4. Tri fusion

Petit rappel de l'algorithme de tri fusion : On découpe la liste de départ en sous-listes de taille 1 (qui sont donc triées). On obtient ainsi une liste de listes triées. On va ensuite fusionner ces sous-listes deux à deux jusqu'à ce qu'il n'y en ait plus qu'une seule, qui sera notre résultat.

Voici un exemple d'implémentation en python :

```
def tri_fusion (liste):
    if liste == []:
        return []
    sous_listes = divise_en_singletons(liste)
    while len(sous_listes) > 1:
        # ICI
        sous_listes = fusionne_sous_listes(sous_listes)
    return sous_listes[0]
```

4.1. On exécute `tri_fusion([4,7,1,3,9,2,3,3])`. Donner la valeur de `sous_listes` à chaque passage par la ligne indiquée.

	valeurs de `sous_listes`
avant la boucle	
tour 0	
tour 1	
tour ?	
tour ?	
après la boucle	

4.2. Compléter le code de la fonction `divise_en_singletons`

```
def divise_en_singletons(liste):
    """
    param : une liste non vide de nombres
    renvoie une liste de listes à un élément, contenant chaque élément de liste
    """
    pass

assert divise_en_singletons([6, 4, 2, 7, 1]) == [[6], [4], [2], [7], [1]]
```

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

```
def fusionne(liste1, liste2):
    """
    param: deux listes triées dans l'ordre croissant
    renvoie la liste triée composée de tous les éléments de liste1 et liste2
    """
    pass

assert fusionne([5, 8, 9, 10], [1, 2, 3, 5, 17, 18]) == [1, 2, 3, 5,
    5, 8, 9, 10, 17, 18]
```

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

```
def fusionne_sous_listes(liste_de_listes):
    """
    param: une liste de listes de listes. Les listes internes sont triées dans
    l'ordre croissant.
    résultat : une nouvelle liste de listes, obtenues en fusionnant
    liste_de_listes[0] avec     liste_de_listes[1] pour la première,
    liste_de_listes[2] avec liste_de_listes[3] pour la deuxième, etc.
    Si len(liste_de_listes) est impair, alors la dernière sous-liste est copiée
    dans le résultat en dernière position.
    """
    pass

assert fusionne_sous_listes([[4, 6], [2], [5, 7],
    [1, 3], [0, 10, 16]]) == [[2, 4, 6], [1, 3, 5, 7], [0, 10, 16]]
```

4.5. On exécute `tri_fusion` sur une liste de taille 16. Combien d'appels à `fusionne_sous_listes` ont lieu ? Avec une liste de taille 32 ? Avec une liste de taille 1000 ?

Exercice 5. Pour réviser

On dispose d'un dictionnaire dont les clés sont les noms d'éléments atomiques et la valeur associée est leur état à température et pression ambiantes. Par exemple :

```
exemple = {'neon ': 'Gaz', 'fer ': 'Solide', 'helium': 'Gaz'}
```

5.1. Écrire le code d'une fonction `frequences` qui prend un tel dictionnaire en paramètre, et qui renvoie un nouveau dictionnaire dont les clés sont les états et les valeurs, le nombre d'éléments atomiques correspondants.

```
def frequences(dico_elements):
    pass

assert frequences(exemple) == {'Gaz': 2, 'Solide': 1}
```

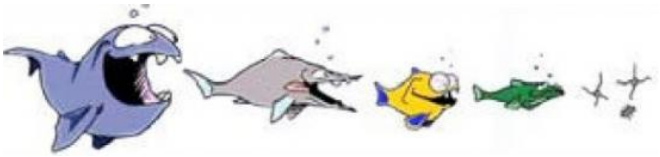
5.2. Écrire le code d'une fonction `etat_min` qui prend un tel dictionnaire en paramètre, et qui renvoie l'état présent dans le dictionnaire qui est le moins représenté.

```
def etat_min(dico_elements):
    pass

assert etat_min(exemple) == 'Solide'
```

Exercice 6. S'entrainer pour l'évaluation

6.1. Compléter le code de la fonction `inverse(dictionnaire)` qui prend en paramètre un dictionnaire et inverse les clés et les valeurs.

```
def inverse(dictionnaire):
    """
    param : un dictionnaire dont les valeurs sont des ensembles
    revoie un dictionnaire dont les clés et les valeurs (éléments
    dans les ensembles) sont inversées
    """
    pass

dico1 = {1: {'a', 'b', 'd'}, 2: {'b', 'z'}, 3: {'b', 'd'}}
dico2 = {'b': {1, 2, 3}, 'a': {1}, 'd': {1, 3}, 'z': {2}}
inverse(dico1) == dico2
inverse(dico2) == dico1
```



Exercice 7. Ecosystème

Dans cet exercice, un écosystème est représenté par un dictionnaire dont les clefs sont le nom des espèces présentes et les valeurs le nom de l'espèce qui constitue leur alimentation.

Par exemple, dans l'écosystème 1, le lion a besoin de lapin pour survivre alors que l'Herbe n'a besoin de rien.

```
ecosysteme_1 = { 'Loup': 'Mouton', 'Mouton':'Herbe', 'Dragon':'Lion',
    'Lion':'Lapin', 'Herbe':None, 'Lapin':'Carotte', 'Requin':'Surfer'}

ecosysteme_2 = { 'Renard':'Poule', 'Poule':'Ver de terre',
    'Ver de terre':'Renard', 'Ours':'Renard' }
```

7.1. Écrire une fonction `extinction_immédiate(ecosysteme)` qui prend en paramètre un écosystème et qui renvoie l'ensemble des espèces de l'écosystème qui ne peuvent pas survivre dans cet ecosystème.

Par exemple, dans l'écosystème1, Le lapin et le requin ne peuvent pas survivre car il n'y a ni carotte ni surfer à se mettre sous la dent. En revanche, dans l'écosystème2, toutes les espèces survivent : elles ont toutes de quoi se sustenter.

```
>>> extinction_immédiate(ecosysteme_1)
{'Lapin', 'Requin'}
>>> extinction_immédiate(ecosysteme_2)
set()
```

7.2. Dans l'écosystème1, comme le lapin ne peut pas survivre, le lion disparaît également par manque de lapin. Et comme le lion disparaît, le dragon disparaît à son tour.

Écrire une fonction `especes_en_voie_disparition(ecosysteme)` qui prend en paramètre un écosystème et qui renvoie l'ensemble des espèces de l'écosystème qui sont en voie de disparition.

Exemples

```
>>> especes_en_voie_disparition(ecosysteme_1)
{'Lapin', 'Requin', 'Lion', 'Dragon'}
>>> especes_en_voie_disparition(ecosysteme_2)
set() # Toutes les espèces survivent : elles ont toutes de quoi se manger
```

Exercice 8. Naturalistes en herbe

On dispose d'un dictionnaire `regime_alimentaire` dont les clés représentent des animaux, et les valeurs l'ensemble des choses dont ils peuvent se nourrir.

Un autre dictionnaire `nourriture_disponible` indique pour un certain nombre d'endroits, l'ensemble des aliments qu'on y trouve.

Par exemple :

```
regime_alimentaire_exemple = {
        'Requin': {'Nageur', 'Sac Plastique', 'Poisson'},
        'Nageur': {'Poisson', 'Noisette', 'Pizza'},
        'Lion': {'Gazelle'},
        'Ecureuil': {'Noisette'},
        'Prof': {'Poisson', 'Pizza', 'Noisette'}
}
nourriture_disponible_exemple = {
        'Ocean': {'Poisson', 'Sac Plastique', 'Nageur'},
        'Savane': {'Gazelle'},
        'Jardin avec piscine': {'Nageur', 'Noisette', 'Pizza'},
        'Orléans': {'Pizza', 'Sac Plastique'},
}
```

Avec les données de cet exemple, le nageur ne peut pas survivre dans la savane, mais il le peut dans un jardin avec piscine.

8.1. Avec les données de l'exemple, quels sont les quels sont tous les animaux capables de survivre dans l'Océan ?

8.2. Écrire une fonction `peutSurvivre(regime_alimentaire,nourriture_disponible)` qui renvoie un dictionnaire dont les clés sont les lieux et les valeurs sont les animaux qui peuvent survivre à cet endroit.

Exercice 9. (facultatif) Un problème de sac à dos

Avant de partir en randonnée, on doit remplir son sac à dos, en particulier avec de la nourriture, qui doit être bon marché, légère à transporter et nourrissante. On représente les aliments à notre disposition par un dictionnaire dont les clés sont des noms d'aliments, et les valeurs sont des tuples indiquant :

- le prix (en euros pour 100g)
- l'apport en énergie (en kCal pour 100g)
- la quantité disponible (en g)

Par exemple :

```
nourriture_disponible = {
        'Abricots secs': (1.5 , 153, 50),
        'Crème de marron': (2.5 , 200, 100) ,
        'Pain': (0.3 , 65, 1000)
}
```

9.1. Écrire une fonction qui renvoie le nom de l'aliment le moins cher à notre disposition

9.2. On veut écrire une fonction `calories_max_pour_10_euros(aliments)` qui déterminer le nombre de calories maximal que l'on pourrait emporter pour 10 euros, si on n'était pas limité par la quantité.

Compléter le test suivant :

Écrire le code de cette fonction.

9.3. On veut écrire une fonction `sac_leger_avec_2000kCal(aliments)` qui indique comment obtenir 2000 kCal avec les aliments les plus légers possibles (en fonction de ce qui est disponible).

a) Compléter le test suivant :

```
assert sac_leger_avec_2000kCal(nourriture_disponible) == ...
```

b) Écrire le code de cette fonction.