R1.01 Initiation au développement
Feuille de TP n°9
Ensembles et dictionnaires

Objectifs de la feuille

- Consolider la maîtrise des ensembles et dictionnaires
- Construire des dictionnaires de fréquences
- Coder une API permettant de manipuler des matrices

Exercice 1 *Petites bêtes*

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

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

1.1 Écrire une fonction `toutes_les_familles(pokedex)` qui prend en paramètre une telle liste et qui renvoie l'ensemble de toutes les familles présentes dans la liste. Préciser la complexité de cette fonction.

```
assert toutes_les_familles(mon_pokedex) == {"Plante", "Poison"}
```

1.2 Écrire une fonction `nombre_pokemons(pokedex, famille)` qui prend en paramètre une telle liste et le nom d'une famille et qui renvoie le nombre de pokemons de cette famille dans la liste. Préciser la complexité de cette fonction.

```
assert nombre_pokemons(mon_pokedex, "Poison") == 2
assert nombre_pokemons(mon_pokedex, "Insecte") == 0
```

1.3 Écrire une fonction `frequences_famille(pokedex)` qui prend en paramètre une telle liste et qui retourne un dictionnaire dont les clefs sont les familles et chaque valeur le nombre de pokémons de cette famille dans la liste. Préciser la complexité de cette fonction.

```
assert frequences_famille(mon_pokedex) == {"Plante": 1, "Poison": 2}
```

1.4 Écrire une fonction `dico_par_famille(pokedex)` qui prend en paramètre une telle liste et qui retourne un dictionnaire dont les clefs sont les familles et chaque valeur l'ensemble des pokémons de cette famille. Préciser la complexité de cette fonction.

```
assert dico_par_famille(mon_pokedex) == {
    "Plante": {"Bulbizarre"}, "Poison": {"Aeromite", "Abo"} }
```

1.5 Écrire une fonction `famille_la_plus_représentée` qui prend en paramètre une telle liste et qui retourne la famille de pokemon qui est la plus représentée dans la liste. Préciser la complexité de cette fonction.

```
assert famille_la_plus_représentée(mon_pokedex) == "Poison"
```

[1] Pour les fans de Pokemons, ne râlez pas tout de suite : une modélisation plus "réaliste" est proposée dans un autre exercice

Exercice 2 *Première API pour manipuler les matrices*

Une API[2] sur les matrices est un ensemble de fonctionnalités qui permettent de faciliter la création d'applications qui manipulent des matrices. Il s'agit ici de se doter de briques "de bases" qui seront réutilisées pour créer une application plus complexe.

Dans cet exercice, on reprend la modélisation 1 vue la semaine dernière (TD 8B)

$$matrice1 = \begin{pmatrix} 10 & 11 & 12 & 13 \\ 14 & 15 & 16 & 17 \\ 18 & 19 & 20 & 21 \end{pmatrice} \qquad matrice2 = \begin{pmatrix} A & B & C \\ D & E & F \end{pmatrice} \qquad matrice3 = \begin{pmatrix} 2 & 7 & 6 \\ 9 & 5 & 1 \\ 4 & 3 & 8 \end{pmatrice}$$

```
matrice1 = (3, 4, [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21])

matrice2 = ...

matrice3 = ...
```

Vous devez compléter les fichiers `test_API_matrice.py` et `API_matrice1.py` en complétant le code des fonctions suivantes :

2.1 `matrice(nb_lignes, nb_colonnes, valeur_par_defaut)` qui crée une nouvelle matrice en mettant la valeur par défaut dans chacune de ses cases.

2.2 `get_nb_lignes(matrice)` permet de connaître le nombre de lignes d'une matrice

2.3 `get_nb_colonnes(matrice)` permet de connaître le nombre de colonnes d'une matrice

2.4 `get_val(matrice, ligne, colonne)` permet de connaître la valeur de l'élément de la matrice dont on connaît le numéro de ligne et le numéro de colonne.

2.5 `set_val(matrice, ligne, colonne, nouvelle_valeur)` permet de modifier la valeur de l'élément qui se trouve à la ligne et à la colonne spécifiées. Cet élément prend alors la valeur `nouvelle_valeur` .

2.6 `charge_matrice_str(nom_fichier)` permet créer une matrice de `str` à partir d'un fichier CSV.

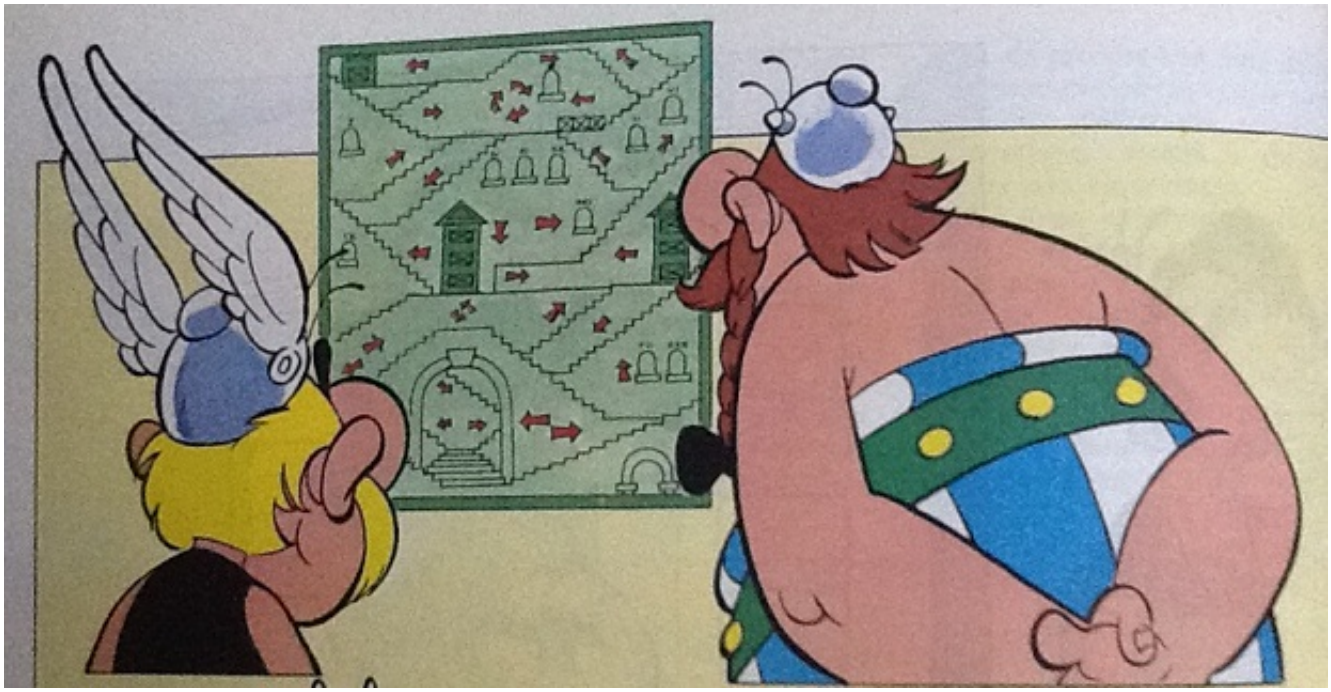
2.7 `sauve_matrice(matrice, nom_fichier)` permet sauvegarder une matrice dans un fichier CSV.

Remarque On vous donne la fonction `affiche(matrice, taille_cellule=4)` . Elle permet d'afficher une matrice dont les cellules ont une largeur `taille_cellule` .
Il est possible d'appeler cette fonction avec deux paramètres(`matrice` et `taille_cellule`) ou avec un seul paramètre (`matrice`). Dans ce cas, le paramètre `taille_cellule` sera remplacé par la valeur `4` .

[2] API est un acronyme anglais qui signifie « Application Programming Interface », que l'on traduit par interface de programmation d'application.

Exercice 3 *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 valeurs 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 }
```

3.1 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"` ?

3.2 Écrire une fonction `quel_guichet(mqrf, guichet_de_depart)` qui prend en entrées une maison qui rend fou et un `guichet_de_depart`, et qui indique quel guichet finira par donner le formulaire si l'on commence par s'adresser au guichet `guichet_de_depart`.

```
assert quel_guichet(mqrf1, "Saudepus") == "Saudepus"
assert quel_guichet(mqrf2, "Abribus") == "Saudepus"
```

3.3 En vous inspirant de la fonction précédente, écrire une fonction `quel_guichet_v2(mqrf, guichet_de_depart)` qui renvoie un tuple contenant le nom du guichet qui finira par donner le formulaire ainsi que le nombre de guichets visités pour y parvenir.

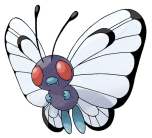
```
assert quel_guichet_v2(mqrf1, "Saudepus") == ("Saudepus", 1)
assert quel_guichet_v2(mqrf2, "Abribus") == ("Saudepus", 5)
```

3.4 Que se passe-t-il si je m'adresse d'abord au guichet de `"Abribus"` dans la `mqrf3` ?

```
mqrf3 = {"Abribus":"Astus", "Jeancloddus": "Astus",
    "Plexus":"Jeancloddus", "Astus":"Gugus",
    "Gugus":"Plexus", "Saudepus":"Bielorus"}
```

3.5 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_v3(mqrf, guichet_de_depart)` qui prend en paramètres une maison qui rend fou et un nom de guichet de depart et renvoie un tuple contenant :

- le nom du guichet qui finira par donner le formulaire ainsi que le nombre de guichets visités pour y parvenir si cela est possible
- None si on ne trouve jamais la sortie



Exercice 4 *Pokemon : Une modélisation plus réaliste*

Les pokemons peuvent avoir plusieurs familles.
Ainsi Bulbizarre appartient à la famille "Plante" mais également celle du "Poison".
On modifie la modélisation proposée dans l'exercice 1 de la façon suivante :

```
mon_pokedex = {"Bulbizarre":{"Plante", "Poison"}, "Aeromite":{"Poison", "
   Insecte"}, "Abo":{"Poison"}}
```

Proposez une v2 des fonctions précédentes qui prennent en paramètre cette nouvelle modélisation.

```
assert toutes_les_familles_v2(mon_pokedex) == {"Plante", "Poison", "
   Insecte"}
assert nombre_pokemons(mon_pokedex, "Poison") == 3
assert frequences_famille_v2(mon_pokedex) == {"Plante": 1, "Poison": 3, "
   Insecte":1}
assert dico_par_famille_v2(mon_pokedex) == { "Plante": {"Bulbizarre",},
    "Poison": {"Bulbizarre", "Aeromite", "Abo"}, "Insecte": {"Aeromite"}}
assert famille_la_plus_représentée_v2(mon_pokedex) == "Poison"
```

Exercice 5 *Matrices : on ajoute quelques fonctionnalités*

Compléter le fichier `test_API_matrice.py` et `API_matrice1.py` en intégrant le code des fonctions suivantes :

5.1 `get_ligne(matrice, ligne)` qui renvoie sous la forme d'une liste la ligne de la matrice dont le numéro est spécifié.

5.2 `get_colonne(matrice, colonne)` qui renvoie sous la forme d'une liste la colonne de la matrice dont le numéro est spécifié.

5.3 `get_diagonale_principale(matrice)` qui renvoie sous la forme d'une liste la diagonale principale d'une matrice carrée.

5.4 `get_diagonale_secondaire(matrice)` qui renvoie sous la forme d'une liste la diagonale secondaire d'une matrice carrée.

5.5 `transpose(matrice)` qui renvoie la transposée d'une matrice.

5.6 `is_triangulaire_inf(matrice)` qui indique si une matrice est triangulaire inférieure.

5.7 `bloc(matrice, ligne, colonne, hauteur, largeur)` qui renvoie la sous-matrice de la matrice commençant à la ligne et colonne indiquées et dont les dimensions sont hauteur et largeur.