Exercice 1

Exercices

1.1 Carré magique

exercice issu du site zonensi

Un carré magique d'ordre n est une matrice carrée n*n telle que que la somme des nombres sur chaque ligne, sur chaque colonne et sur chaque diagonale principale soient égales.

1. Vérifier que la matrice M est bien un carré magique. :

```
M = [(4,9,2), (3,5,7), (8,1,6)]
```

2. On se propose de construire une fonction vérifiant qu'une matrice de taille n*n est bien un carré magique :

- Créer une fonction `estCarre(M)` qui vérifie que la matrice est bien carrée (son nombre de ligne est égal à son nombre de colonne). Cette fonction renverra `True` dans ce cas, et `False` sinon.
- Créer une fonction `sommeLigne(M,i)` qui renvoie la somme des nombres de la ligne de la matrice M
- Créer une fonction `sommeColonne(M,i)` qui renvoie la somme des nombres de la colonne de la matrice M
- Créer une fonction `sommeDiagPrincipale(M)` qui renvoie la somme des nombres de la diagonale principale de M. (diagonale dont les éléments ont le même numéro de ligne et de colonne).
- Créer une fonction `sommeDiagSecondaire(M)` qui renvoie la somme des nombres de la diagonale non principale de M
- En utilisant les fonctions précédentes, créer une fonction `estMagique(M)` qui renvoie `True` si la matrice est magique, et `False` sinon.

3. Ajouter des tests avec le module `doctest`

Nous allons ajouter à nos fonctions des informations pour limiter les risques d'erreurs de programmation.

Ainsi, dans la fonction `sommeLigne(M,i)`, on ajoutera les lignes dans le docstring :

```
def sommeLigne(M,i):
    """
    >>> M = [(4,9,2),(3,5,7),(8,1,6)]
    >>> sommeLigne(M,0)
    15
    """
    # script de la fonction
```

Puis, à la fin du programme :

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Maintenant, lorsque vous executez le fichier, le module `doctest` va rechercher tous les tests simulés dans les docstrings, et vérifier les résultats. Un message affichera alors le nombre de tests reussis ou manqués.

Voir dans allophysique > Python avancé > 4. Programmer des tests avec Doctest

> Ajouter un test avec son resultat dans chacune des fonctions programmées plus haut.

1.2 Liste de dictionnaires

On considère les données des étudiants mises dans une liste de dictionnaires :

```
etudiants = [
    {"nom": "Alice", "age": 21, "ville": "Paris"},
    {"nom": "Bob", "age": 22, "ville": "Lyon"},
    {"nom": "Charlie", "age": 23, "ville": "Marseille"}
]
```

1. Compléter le script qui donne le nom de tous les étudiants

```
for etudiant in etudiants:
    print(etudiant…[
```

2. afficher l'age à l'aide d'une boucle for :

```
for etudiant in etudiants:
    …
```

3. Afficher le nom des étudiants qui habitent Lyon (s'il y en a) :

```
for e in etudiants:
    if e[   ..      ==    ..     :
        print(e[..
```

4. Affiche toutes les personnes avec leur âge. On utilisera l'expression formatée `print({} : {} ans.format(nom,age))`

```
...
...
...
...
```

5. Comment est modifiée la liste etudiants avec l'instruction suivante ?

```
> etudiants[0]["age"] = 22
```

6. Le script suivant permet de calculer l'âge moyen des étudiants. Complétez le :

```
total_age = ..
for e in etudiants:
    total_age += ..

moyenne = total_age / ..
print"(Moyenne d'âge :", moyenne)
```

Exercice 2

Correction

```
def estCarre(M):
    return len(M) == len(M[0])

def sommeLigne(M,i):
    """
    : example
    >>> M = [(4,9,2),(3,5,7),(8,1,6)]
    >>> sommeLigne(M,0)
    15
    """
    s = 0
    for x in M[i]:
        s += x
    return x

def sommeDiagPrincipale(M):
    s = 0
    for i in range(len(M)):
        s+=M[i][i]
    return s

def sommeDiagSecondaire(M):
    s = 0
    for i in range(len(M)):
        s+=M[len(M)-1-i][i]
    return s

def estMagique(M):
    test = True
    s = sommeLigne(M,0)
    for i in range(1,len(M)):
        if sommeLigne(M,i) != s:
            test = False
    if sommeDiagPrincipale(M)!=s or sommeDiagSecondaire(M)!=s:
        test = False
    return test

if __name__ == "__main__":
    import doctest
```

```
40     doctest.testmod()
```