Exercice 1

Graphes et parcours de graphe (extrait AN1 J1 Ex 2)

Le dictionnaire suivant est une implémentation d'un graphe :

```
graphe = {'G':['J','N'],
          'J':['G','Y','E','L'],
          'Y':['J','E','N'],
          'E':['J','Y','N'],
          'N':['G','Y','E'],
          'M':['A'],
          'A':['M'],
          'L':['J']}
```

1. Donner la definition de *graphe*
2. Quelles sont les clés du dictionnaire ?
3. Comment, à partir de ce dictionnaire parvient-on à atteindre la liste `['G','Y','E','L']` ?
4. Représenter ce graphe.

On donne les 2 algorithmes de parcours en profondeur d'un graphe :

```
def DFS(d,s,visited=[]):
    visited.append(s)
    for v in d[s]:
        if v not in visited:
            DFS(d,v)
    return visited
```

```
def DFS_ite(d,s,visited=[], stack=[]):
        stack.append(s)
        while stack:
            v = stack.pop()
            if v not in visited:
                visited.append(v)
                unvisited = [n for n in d[v] if n not in visited]
                stack.extend(inverse(unvisited))
        return visited

def inverse(L):
    return [L[i] for i in range(len(L)-1,-1,-1)]
```

1. Expliquer ce qu'est un parcours en profondeur.
2. Laquelle de ces fonctions est de type itératif ? Récursif ? Pourquoi ?
3. Ecrire dans chaque cas l'instruction qui appelle la fonction, afin de faire le parcours en profondeur du graphe, à partir du sommet L.
4. Quelles seront les listes retournées par chacune de ces fonctions ?

On donne la fonction de parcours en largeur d'un graphe :

```
from collections import deque

def BFS_ite(d,s,visited=[], queue=deque()):
        queue.append(s)
        while queue:
            v = queue.popleft()
            if v not in visited:
                visited.append(v)
                unvisited = [n for n in d[v] if n not in visited]
                queue.extend(unvisited)
        return visited
```

Remarque :

La librairie *collection* apporte la structure de donnée dequeue qui se comporte comme une *File*, avec la methode de classe `popleft()` qui permet de retirer le PREMIER element de la File, en temps constant :

```
from collections import deque
# Declaring deque
queue = deque(['name','age','DOB'])
print(queue.popleft())
# affiche name
print(queue)
# affiche deque(['age', 'DOB'])
```

1. Quelles sont les différences avec la fonction de parcours en profondeur ?
2. Quelles sera la liste retournée par cette fonction ?

Exercice 2

Listes chainées (extrait Am Nord J2 2024 Ex 3)

Un réseau PaP est utilisé pour effectuer des transactions financières en monnaie numérique *nsicoin* entre les trois utilisateurs. Pour cela, on crée la classe Transaction ci-dessous :

```
class Transaction:
    def __init__(self, expediteur, destinataire, montant):
        self.expediteur = expediteur
        self.destinataire = destinataire
        self.montant = montant
```

2. Dans cycle de transactions, Alice envoie dix *nsicoin* à Charlie puis Bob envoie cinq nsicoin à Alice. Écrire la liste Python correspondante à ces transactions. Les transactions réalisées pendant cet intervalle de temps sont regroupées par ordre d'apparition dans une liste Python.

Pour garder une trace de toutes les transactions effectuées, on utilise une liste chaînée de blocs (ou blockchain) dont le code Python est fourni ci-dessous. Toutes les dix minutes un nouveau bloc contenant les nouvelles transactions est créé et ajouté à la blockchain.

```python
class Bloc:
    def __init__(self, liste_transactions, bloc_precedent):
        self.liste_transactions = liste_transactions
        self.bloc_precedent = bloc_precedent # de type Bloc

class Blockchain:
    def __init__(self):
        self.tete = self.creer_bloc_0()

    def creer_bloc_0(self):
        """
        Crée le premier bloc qui distribue 100 nsicoin à tous les
    utilisateurs
        (un pseudo-utilisateur Genesis est utilisé comme
        expéditeur)
        """
        liste_transactions = [
            Transaction("Genesis", "Alice", 100),
            Transaction("Genesis", "Bob", 100),
            Transaction("Genesis", "Charlie", 100)
            ]
        return Bloc(liste_transactions, None)
```

3. La figure 1 représente les trois premiers blocs d'une Blockchain. Expliquer pourquoi la valeur de l'attribut `bloc_precedent` du bloc0 est None.

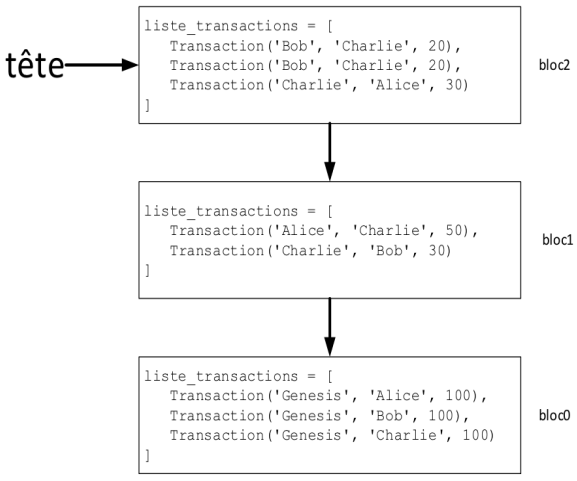

FIGURE 1 – Blockchain

4. À l'aide des classes Bloc et Blockchain, écrire le code Python permettant de créer un objet `ma_blockchain` de type Blockchain représenté par la figure 1.

5. Donner le solde en nsicoin de Bob à l'issue du bloc2.

6. On souhaite doter la classe Blockchain d'une méthode `ajouter_bloc` qui prend en paramètre la liste des transactions du dernier cycle et l'ajoute dans un nouveau bloc. Écrire le code Python de cette méthode ci-dessous.

```python
def ajouter_bloc(self, liste_transactions):
    # A compléter
```

9. On souhaite doter la classe Bloc d'une nouvelle méthode `calculer_solde` permettant de renvoyer le solde à l'issue de ce bloc. Recopier et compléter sur votre copie le code Python de cette méthode :

```python
def calculer_solde(self, utilisateur):
    if self.precedent is None: # cas de base
        solde = 0
    else:
        solde = ... # appel récursif : calcul du solde au bloc précé
   dent
        for transaction in bloc.liste_transactions
            if ... == utilisateur:
                solde = solde - ....
            elif ... :
                ...
        return solde
```

10. Écrire l'appel à la fonction `calculer_solde` permettant de calculer le solde actuel de Alice

Exercice 3

Dictionnaires (extraits 2024 ANJ1 Ex3 et sujet 0B Ex 3)

3.1 Import et traitement csv

Voici un exemple d'utilisation de DictReader du module *csv* :

```
fichier exemple.csv

champ1, champ2
a,7
b,8
c,9
```

code Python

```python
import csv
with open('exemple.csv','r') as fichier:
    donnees = list(csv.DictReader(fichier,delimiter=','))
print(donnees)
```

Affichage :

```
[{'champ1': 'a', 'champ2': '7'},
 {'champ1': 'b', 'champ2': '8'},
 {'champ1': 'c', 'champ2': '9'}]
```

Voici un extrait du fichier `flashcards.csv` :

```
discipline;chapitre;question;reponse
histoire;crise de 1929; jeudi noir - date;24 octobre 1929
histoire;crise de 1929; jeudi noir - quoi;krach boursier
histoire;2GM;l'Axe;Allemagne, Italie, Japon
histoire;2GM;les Alliés;Chine, Etats-Unis, France, Royaume-Uni, URSS
philosophie;travail;Marx;alienation de l'ouvrier
```

1. Ecrire une fonction `charger` qui pend `nom_fichier` comme paramètre et charge le contenu d'un fichier pour retourner un tableau (liste de dict) comme dans l'exemple ci-dessus.
2. Ecrire l'instruction qui charge `flashcards.csv` dans la variable `flashcard`, de type tableau.
3. Ecrire une fonction `choix_discipline` qui prend pour parametres `donnees`, un tableau, et `disc`, la discipline choisie. Cette fonction retourne un tableau à partir de la selection sur les disciplines egales à `disc`.
4. Ecrire une fonction `question_reponse` qui prend pour paramètres `donnees`, `disc` et qui retourne une liste de listes constituées de couples `(question, reponse)` relatifs à la discipline choisie. Utiliser la fonction `choix_discipline`.

3.2 Livres et notes maxi, recherche dans une table

Dans cette première partie, on utilise un dictionnaire Python. On considère le programme suivant :

```
dico_livres = {'id' : [1, 2, 14, 4, 5, 8, 7, 15, 9, 10],
               'titre' : ['1984', 'Dune', 'Fondation', 'Ubik', 'Blade
    Runner', 'Les Robots', 'Ravage', 'Chroniques martiennes', 'Dragon dé
    chu', 'Fahrenheit 451'],
               'auteur' : ['Orwell', 'Herbert', 'Asimov', 'K.Dick', 'K.
    Dick', 'Asimov', 'Barjavel', 'Bradbury', 'Hamilton', 'Bradbury'],
               'ann_pub' : [1949, 1965, 1951, 1953,1968, 1950, 1943, 1950,
     2003, 1953],
               'note' : [10, 8, 9, 9, 8, 10, 6, 7, 8, 8]
               }
a = dico_livres['note']
b = dico_livres['titre'][2]
```

1. Déterminer les valeurs des variables a et b après l'exécution de ce programme.

La fonction `titre_livre` prend en paramètre un dictionnaire (de même structure que `dico_livres`) et un identifiant, et renvoie le titre du livre qui correspond à cet identifiant. Dans le cas où l'identifiant passé en paramètre n'est pas présent dans le dictionnaire, la fonction renvoie `None`.

```
def titre_livre(dico, id_livre):
    for i in range(len(dico['id'])):
        if dico['id'][i] == ... :
            return dico['titre'][...]
    return ...
```

2. Recopier et compléter les lignes 3, 4 et 5 de la fonction `titre_livre`.

3. Écrire une fonction `note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la note maximale.

4. Écrire une fonction `livres_note` qui prend en paramètre un dictionnaire dico (de même structure que `dico_livres`) et une note n, et qui renvoie la liste des titres des livres ayant obtenu la note n (on rappelle que t.append(a) permet de rajouter l'élément a à la fin de la liste t).

5. Écrire une fonction `livre_note_maxi` qui prend en paramètre un dictionnaire dico (de même structure que dico_livres) et qui renvoie la liste des titres des livres ayant obtenu la meilleure note sous la forme d'une liste Python.

Exercice 4

Arbres, parcours BFS (extrait 2024 centres etrangers J2, Ex 3)

4.1 Partie B

On considère l'arborescence de fichiers de la figure suivante :

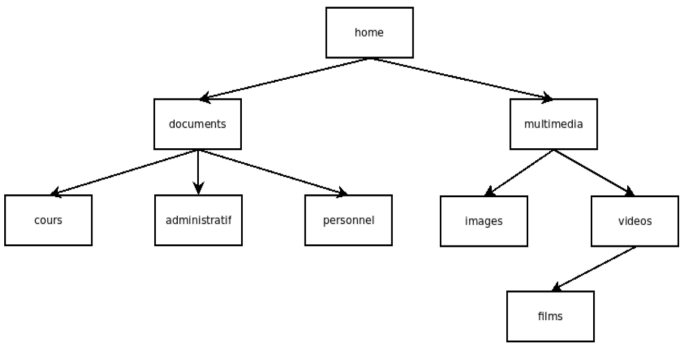

FIGURE 2 – arborescence fichiers

on propose l'implémentation suivante. L'attribut fils est une variable de type list contenant tous les dossiers fils. Cette liste est vide dans le cas où le dossier est vide.

```
class Dossier:
    def __init__(self, nom, liste):
        self.nom = nom
        self.fils = liste # liste d'objets de la classe Dossier
```

1. Écrire le code Python d'une méthode `est_vide` qui renvoie `True` lorsque le dossier est vide et `False` sinon.

2. Écrire le code Python permettant d'instancier une variable `var_multimedia` de la classe Dossier représentant le dossier multimedia de la figure précédente. Attention : cela nécessite d'instancier tous les nœuds du sous-arbre de racine multimedia.

3. Recopier et compléter sur votre copie le code Python de la méthode parcours suivante qui affiche les noms de tous les descendants d'un dossier en utilisant l'ordre préfixe.

```
def parcours(self):
    print(...)
```

```
3       for f in ...:
4           ...
```

4. Justifier que cette méthode parcours termine toujours sur une arborescence de fichiers.
5. Proposer une modification de la méthode parcours pour que celle-ci effectue plutôt un parcours suffixe (ou postfixe).
6. Expliquer la différence de comportement entre un appel à la méthode `parcours` de la classe Dossier et une exécution de la commande UNIX `ls`

On considère la variable `var_videos` de type Dossier représentant le dossier videos de la figure précédente. On souhaite que le code Python `var_videos.mkdir("documentaires")` crée un dossier documentaires vide dans le dossier `var_videos`.

7. Écrire le code Python de la méthode `mkdir`.
8. Écrire en Python une méthode `contient(self, nom_dossier)` qui renvoie `True` si l'arborescence de racine `self` contient au moins un dossier de nom `nom_dossier` et `False` sinon.
9. Avec l'implémentation de la classe Dossier de cette partie, expliquer comment il serait possible de déterminer le dossier parent d'un dossier donné dans une arborescence donnée. On attend ici l'idée principale de l'algorithme décrite en français. On ne demande pas d'implémenter cet algorithme en Python.
10. Proposer une modification dans la méthode `__init__` de la classe Dossier qui permettrait de répondre à la question précédente beaucoup plus efficacement et expliquer votre choix.