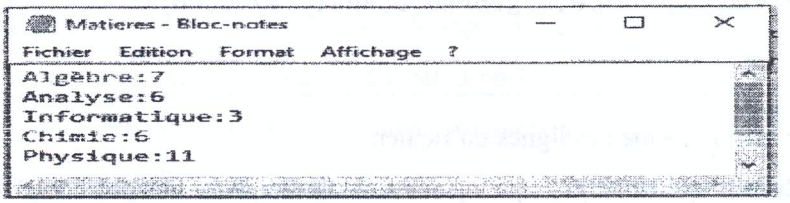
DEVOIR SEMESTRE 1

Matière : INFORMATIQUE
Classes : 2ème Année ***Durée :*** 1 h
Préparation : MP-PC-PT

Problème 1 (10 points)

Soit un fichier texte intitulé 'Matieres.txt' dont chaque ligne contient le nom d'une matière suivi de son coefficient, séparés par le caractère ':'.



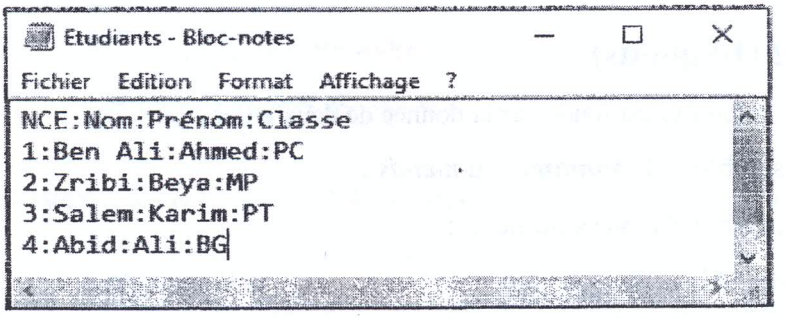
Travail demandé : Écrire un programme Python contenant les fonctions suivantes :

1. Saisie(): qui permet de saisir et de retourner un entier 0<N<=20.

2. Saisie_Etudiants(N) : qui permet de saisir le NCE (de type entier), le nom, le prénom et la classe de **N** étudiants séparés par le caractère **':'** et les enregistrer dans un fichier texte nommé "Etudiants.txt".

Exemple :

>>> Saisie_Etudiants(4) retourne le fichier 'Etudiants.txt' contenant 5 lignes (une entête et quatre lignes contenant les informations des quatre étudiants)



NB. Respecter bien la forme des lignes du fichier.

3. Construire_Dic(fich) : qui à partir d'un fichier **fich** stockant les données relatives des matières, retourne un dictionnaire composé de clefs qui sont les noms des matières et de valeurs qui sont les coefficients correspondants.

Exemple:

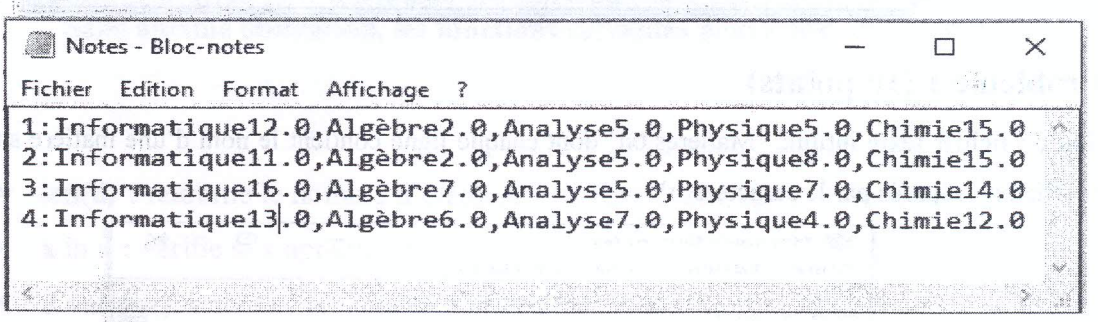
>>> Construire_Dic('Matieres.txt') retourne {'Informatique': 3, 'Analyse': 6, 'Chimie': 6, 'Algèbre': 7, 'Physique': 11}

4. Saisie_Notes(fich,N) : qui à partir d'un fichier **fich** des matières et **N** le nombre des étudiants, stocke dans un fichier **'Notes.txt'** les notes d'un étudiant.

Chaque ligne est formée par NCE:matière1note1,matière2note2,matière3note3, etc…

Exemple :

>>> Saisie_Notes("Matieres.txt",4) retourne le fichier 'Notes.txt'



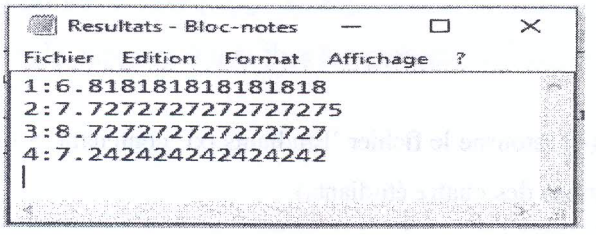
NB. Respecter bien la forme des lignes du fichier.

5. Calculer_Resultat(fich1,fich2): qui à partir d'un fichier **fich1** des matières et **fich2** des notes, stocke dans un fichier 'Resultats.txt' la moyenne de chaque étudiant.

NB. La moyenne = (note de chaque matière × son coefficient)/total des coefficients.

Exemple :

>>>Calculer_Resultat("Matieres.txt","Notes.txt") créé le fichier 'Resultats.txt'



Problème 2 (10 points)

Un graphe non orienté G est fondé sur la donnée de deux ensembles :

- un ensemble *S* de ***sommets*** ou *nœuds* ;
- un ensemble *A* d'**arcs** ou liens ;

Si on note l'ensemble S de n sommets S= $\{s_0, s_1, \ldots s_n\}$ et l'ensemble A de p arrêtes A=$\{a_0, a_1, .., a_{p-1}\}$, alors toute arrête a_i est définie par un couple non ordonné de sommets $(s_j,s_k) \in S^2$.

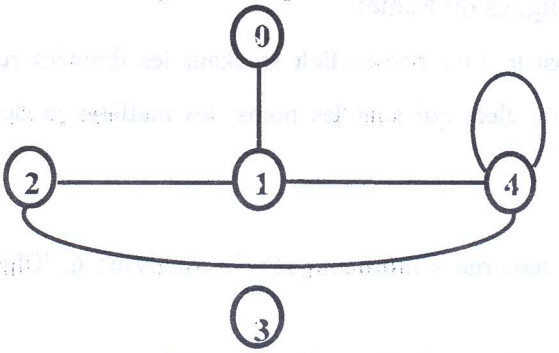


Figure 1 : Exemple de graphe non orienté

Soit un exemple de graphe non orienté représenté par la figure 1. Les **n** sommets de ce graphe sont numérotés de 0 à n-1.

Chaque arrête est désignée par un couple (i,j) où i et j appartiennent nécessairement à l'intervalle [0,n-1].

Il existe plusieurs méthodes pour représenter un graphe. La plus naturelle consiste à utiliser le tuple (n,L) où **n** est le nombre de sommets et **L** la liste de ses arrêtes énumérées dans n'importe quel ordre.

On appelle **degré** du sommet **k** et on note **deg(k)** le nombre d'arrêtes dont ce sommet est l'une des extrémités.

Question 1. Donner le tuple (n,L) où **n** est le nombre de sommet du graphe de la figure1 et **L** la liste de ses arrêtes.

Question 2. Ecrire une fonction **degre(liste_arretes,sommet)** prenant en arguments la liste des arrêtes d'un graphe et le numéro d'un sommet et retournant le degré de ce dernier.

Exemple : >>>degre([(0,1),(1,2),(1,4),(2,4),(4,4)],1) retourne 3

Question 3. Ecrire une fonction **non_isole(nombre_sommets,liste_arretes)** prenant en arguments le nombre de sommets d'un graphe et la liste de ses arrêtes et retournant la liste des sommets non isolés, c'est-à-dire intervenant dans au moins une arrête.

Exemple : >>>non_isole(5, [(0,1),(1,2),(1,4),(2,4),(4,4)]) retourne [0, 1, 2, 4]

Une méthode alternative de représentation d'un graphe, plus performante, consiste à utiliser la notion de liste d'adjacence. On appelle liste d'adjacence d'un graphe à n sommets la liste de longueur n dont l'élément courant d'indice **k** est lui-même une liste classant, dans un ordre quelconque, l'ensemble des sommets reliés par une arrête au sommet k. La donnée de la liste d'adjacence suffit pour caractériser un graphe dans son intégralité. Soit la liste d'adjacence du graphe de la figure 1 liste_adjacence=[[1],[0,2,4],[1,4],[],[1,2,4]]. Sommet 1 est lié uniquement à 0,2 et 4. Sommet 4 est lié uniquement à 1,2 et 4.

Question 4. Ecrire une fonction **adjacence(nombre_sommets,liste_aretes)** prenant en arguments le nombre de sommets d'un graphe et la liste de ses arrêtes et retournant la liste d'adjacence de cette dernière.

Exemple : >>>adjacence(5, [(0,1),(1,2),(1,4),(2,4),(4,4)]) retourne [[1, 0, 2, 1, 4, 1, 4, 2, 4], [1, 0, 2, 1, 4, 1, 4, 2, 4], [1, 0, 2, 1, 4, 1, 4, 2, 4], [1, 0, 2, 1, 4, 1, 4, 2, 4], [1, 0, 2, 1, 4, 1, 4, 2, 4]]

Question 5. Ecrire une fonction **max_degre(liste_adjacence)** prenant en arguments la liste_adjacence d'un graphe et retournant le sommet dont l'indice est le plus grand.

Exemple : >>>max_degre([[1],[0,2,4],[1,4],[],[1,2,4]]) retourne 4

car la longueur de [1] est 1, la longueur de [0,2,4] est 3 , etc.

ANNEXE – Quelques méthodes Python de la classe dict et de la classe str

Sans aucune obligation, les fonctions suivantes pourraient vous être utiles.

- **len(d)** : retourne le nombre d'éléments du dictionnaire d.
- **x in d** : vérifie si x appartient à d.
- **d.values()**: retourne un itérable formé par les valeurs du dictionnaire d.
- **d.keys()**: retourne un itérable formé par les clés du dictionnaire d.
- **ch.isalpha()**: renvoie True si la chaîne ch n'est formé que par des caractères alphabétiques, False si non.
- **ch.split(sep)**: permet de retourner la liste formée par les sous-chaînes de la chaîne de caractères ch séparées par le caractère sep donné en argument (par exemple "2 :deux".split(':') donne la liste ['2', 'deux']).
- **ch.strip()** : permet de supprimer à la fois les espaces de fin et de début de la chaine ch.