Université de Tunis **A.U : 2017- 2018**
Institut Préparatoire aux Etudes d'Ingénieurs de Tunis

Examen Informatique N°2

Classes : 2ème Année SM & SP
Durée : 2 h ***Documents non autorisés***

Exercice (8pts):

L'Instance Supérieure Indépendante pour les élections gère les informations concernant les différentes municipalités tunisiennes à travers un fichier texte "munip.txt". Ce fichier contient pour chaque municipalité son nom, son gouvernorat (un nombre de 1 à 24) et sa population. Ainsi chaque ligne du fichier est représentée sous cette forme : **#nom;gouvernorat;population**

Exemple du fichier 'munip.txt'

#Ariana;02 ;114 486
#Sidi Hassine;01 ;109 690
#Mohamedia;03 ;106 167
#El Mourouj;03 ;104 586
#Gafsa;04 ;95 242
#Raoued;02 ;94 691
…

Travail demandé :

1. Écrire une fonction python **liste_munip**, qui à partir du fichier "munip.txt" crée et retourne une liste de chaines de caractères où chaque chaine correspond aux différentes informations d'une municipalité. Il est à noter que les chaines de caractères ne contiennent pas les symboles '#' et '\n'.
2. Écrire une fonction python **tuple_munip**, qui à partir du fichier "munip.txt", crée et retourne un tuple où chaque élément est aussi un tuple contenant le nom, le gouvernorat, et la population. (avec nom de type chaine et population et gouvernorat de type entier).
3. Écrire une fonction python **lc_munip**, qui à partir du fichier "munip.txt" retourne la municipalité ayant le nom le plus long et la municipalité ayant le nom le plus court.

4. Écrire une fonction python, **seuil__munip**, qui à partir du fichier "munip.txt" retourne la liste des municipalités ayant un nombre d'habitant inferieur à une valeur N. Cette liste comprendra des couples de la forme (nom, gouvernorat).
5. Écrire une fonction python **gouv_munip**, qui à partir du fichier "munip.txt" retourne un dictionnaire contenant le nombre de municipalités et la population de chaque gouvernorat. Ainsi, le nom du gouvernorat représentera la clé du dictionnaire et le tuple (nombre_ de_ municipalités, population) représentera la valeur de la clé.
6. Écrire une fonction python **peup_gouv**, qui à partir du fichier "munip.txt" retourne le gouvernorat le plus peuplé ainsi que le moins peuplé.

Problème (12pts):

Dans cet exercice, on se propose de définir la classe **Intervalle** ci-dessous permettant de modéliser des intervalles mathématiques **fermés de réels strictement positifs** et de les manipuler.

Class Intervalle:

```
def __init__(self, a, b):
        #Réponse à la Question 1
def __repr__(self):
        #Réponse à la question 2
def __contains__(self, x) :
        #Réponse à la question 3
def __add__(self, autre) :
        #Réponse à la question 4
def __mul__(self, autre) :
        #Réponse à la question 5
def inverse(self) :
        #Réponse à la question 6
def __truediv__(self, autre) :
        #Réponse à la question 7
def __and__(self, autre) :
        #Réponse à la question 8
def __or__(self, autre) :
        #Réponse à la question 9
```

Travail demandé :

1. Ecrire la méthode d'initialisation de la classe Intervalle **__init__(self, a, b)** qui permet d'initialiser la borne inférieure et la borne supérieure d'un intervalle de réels strictement positifs avec les valeurs a et b. La borne inférieure sera initialisée avec la plus petite valeur de a et b tandis que la borne supérieure prendra la plus grande valeur.
2. Ecrire la méthode de représentation de l'intervalle **__repr__(self)** qui permet de représenter l'objet intervalle sous la forme suivante [borne_inf, borne_sup].

   Exemple : a=Intervalle(2,8) donne [2, 8].

3. Ecrire la méthode, nommée **__contains__(self,x)** qui permet de vérifier l'appartenance d'un réel x à l'intervalle.

   Exemple : 4 in Intervalle(2.0, 5.0) donne True.

4. Ecrire la méthode **__add__(self, autre**) qui permet d'additionner deux intervalles.

   Exemple : Intervalle(2.0, 5.0) + Intervalle(3.0, 4.0) donne [5.0, 9.0].

5. Ecrire la méthode **__mul__(self,autre)** qui retourne le produit de deux intervalles.

   Exemple : Intervalle(2.0, 5.0)*Intervalle (3.0, 4.0)donne [6.0, 20.0].

6. Ecrire la méthode **inverse(self)** qui retourne l'inverse d'un intervalle. L'inverse d'un intervalle a pour borne inférieure (respectivement supérieure) l'inverse de la borne supérieure (respectivement l'inverse de la borne inférieure).

   Exemple : inverse(Intervalle(2.0, 4.0)) donne [0.25, 0.5].

7. Ecrire la méthode **__truediv__(self,autre)** qui retourne le quotient de deux intervalles.
   La borne inférieure de l'intervalle quotient est obtenue en faisant le quotient de la borne inférieure du premier intervalle par la borne supérieure du deuxième intervalle. Par contre, la borne supérieure de l'intervalle quotient est égale au quotient de la borne supérieure du premier intervalle par la borne inférieure du deuxième intervalle.

   Exemple : Intervalle(3.0, 4.0) I/ntervalle(2.0, 5.0]) donne [0.6, 2.0].

8. Ecrire la méthode **__and__(self,autre)** qui permet de retourner le résultat de l'intersection de deux intervalles. La borne inférieure de l'intersection est la plus grande des deux bornes inférieures tandis que la borne supérieure de l'intersection est la plus petite des deux bornes supérieures. La méthode retourne None si l'intersection est vide.

   Exemple : Intervalle(1.0, 3.0)&Intervalle(2.0,4.0) donne [2.0, 3.0].

9. Ecrire la méthode **__or__(self, autre)** qui permet de retourner l'intervalle réunion et None si leur intersection est vide.

   Exemple : Intervalle(1.0,3.0) | Intervalle(2.0,4.0) donne [1.0, 4.0].

Application

Soient deux résistances R1 (1200 Ω à ± 0, 1%) et R2 (1800 Ω à ± 0, 1%) placées en parallèle.

a. Exprimez les valeurs des résistances R1 et R2 sous formes d'intervalles.
b. Ecrire une fonction python **resistance_equivalente**(R1,R2) qui permet de calculer et de retourner la résistance équivalente des deux résistances en parallèle en utilisant la formule suivante: $R_{eq} = \frac{R_1 \times R_2}{R_1 + R_2}$.

Bon travail