Initiation à Python
Devoir et TD 2

Justin Cano
École Polytechnique de Montréal
Département de Génie Électrique
Email: nom.prénom@polymtl.ca

Résumé

Ce document comporte deux parties : le devoir et le TD proposé à l'issue de ma présentation.
Sujets abordés : Notions de programmation objet, de bibliothèques et de gestions de fichiers.

I. Travaux dirigés

A. Définition d'une classe

On veut créer un programme de gestion de chats (*miaou mais pas clavardage, je précise.*) pour un vétérinaire. Sauf qu'un chat n'est pas un type connu dans Python.

Un `Chat` comporte les attributs suivants :

— `nom`;
— `anneeDeNaissance`;
— `poids`;
— `race`;
— `vaccins`.

évidemment le nom et la race sont des *strings* tandis que son année de naissance et son poids sont des respectivement un *entier* et un *flottant*. Pour les vaccins c'est plus délicat, il s'agit d'un autre objet de classe `Vaccin`, qui comporte les attributs suivants :

— `anneeInjection`;
— `pathologie`.

là aussi, il y a un string et un entier.

Note : tout ce qui suit sera dans un fichier `Vet.py`.

1) Coder la classe `Vaccin` sans aucune autre méthode que le constructeur générique (qui initialise les attributs à 0 et respectivement `""`, des *setters/getters* et une méthode d'affichage `print()` qui affiche tous les attributs ;
2) Créez à la main deux vaccins «`Tetanos`», «`Rage`» administrés en 2015 et en 2016 et imprimez-les pour tester votre classe ;
3) Coder les premières fonctions de la classe chat : constructeur ayant en argument le nom, l'année de naissance, le poids et la race. Un setter pour le poids et aussi demandé (oui un chat peut grossir). L'attribut `Vaccins` est une liste vide [] pour l'instant.

   Une méthode `print()` affichera tous les attributs. Utiliser une boucle `for` pour les vaccins et utilisant la méthode `print()` de cette classe.
4) Coder une méthode `inoculate_tetanos(year)`, prenant en argument une année et qui ajoute dans votre «chat» sa vaccination pour le «`Tetanos`» ;
5) Faire de même pour `inoculate_rage(year)` avec «`Rage`» remarquer qu'on peut créer `inoculate_vaccine(name,year)` pour mutualiser les codes.

6) Coder une routine qui calculera l'age de votre chat (nouvel attribut `age` à stocker à l'intérieur de votre objet `Chat`) avec l'année en cours en argument ;

7) Coder une routine qui vérifiera si le tétanos a bien été administré à un chat qui a plus de deux ans elle doit retourner "`vaccin Tétanos ok`" ou "`vaccin Tétanos à faire`" ;

8) Vérifier le logiciel précédent : on crée le `Chat` ≪Agathe≫, qui pèse 2kgs, est née en 2017, qui est de race Européenne, qui a eu le vaccin contre le Tétanos en 2018. On exécute la routine codée en 7) et on vérifie le résultat. Ré-exécuter le code pour le `Chat` ≪Jerry≫, né en 2011, Siamois, vacciné pour la Rage en 2013 (mais pas pour le Tétanos). Afficher les attributs de chacun des chats durant l'exécution.

B. Import de librairie externe

On reprend le code de la section précédente.

1) Dans un même répertoire créer un fichier `pharmacieVeterinaire.py` externe qui importe l'élément Vaccin de la librairie `Vet.py`.

2) Créer alors une liste de vaccins de trois éléments (libre à vous) et l'afficher. Cela peut être utilisé concrètement pour afficher des données de vaccinations sans pour autant dévoiler le nom du ≪patient≫, très utile en ces temps pandémiques.

C. Gestion de fichiers

On va essayer d'effectuer des opérations automatisées de tri sur des bases de données en `.csv` pour s'exercer.

1) Créer une routine qui importe le fichier `demographie_villes.csv` dans des listes `villes` et `nombre_habitants` ;

2) Créer une routine qui ira classer par ordre croissant d'habitants ces villes. Attention, il ne faut pas perdre les indices : on pourra alors retrouver l'indice du maximum d'habitants, retirer de la liste `nombre_habitants` et `villes` la case jusqu'à ce qu'elle soit vide ;

3) Créer une routine qui crée `demographie_villes_triee.csv`, version triée de la liste donnée dans le désordre précédemment.

II. Devoir 2 : faire de la concurrence (modeste) à Pronote

A. Contexte

Vous êtes un professeur de prépa désireux de produire des bulletins de notes automatisés à la fin de l'année de *Math Sup* : les élèves sont soit couverts d'éloge (vont en classe étoile) soit bannis du lycée en fonction de leur classement. Également, pour le recrutement, vous êtes intéressé par la performance des élèves en fonction de leur lycée d'origine.

B. Entrées, structures de données et sorties requises

On vous donne une base de données d'élèves `notes_eleves.csv` qui contient les noms, prénoms, et les notes obtenues durant le trimestre (avec en tête de coefficient et de matière) et lycée d'origine.

Il faudra, en utilisant `Python` charger cette dernière, de manière totalement automatisée. Ensuite, créer des objets `Evaluation` et `Eleve` avec des attributs à déterminer. Un `Eleve` contient ses notes dans un tableau dont chacune des classe correspond à une évaluation. La classe de *Math Sup* est composée minimalement de deux listes : une d'élèves et une d'évaluations. Ce groupe peut être modélisé par une classe `ClasseDePrepa`[1].

Tout ceci, pour obtenir un rapport de fin d'année tenant dans un fichier texte à envoyer aux élèves en fin d'année. Les statistiques sont affichées sur la console Python, c'est juste par curiosité.

C. Fonctionnalités demandées

1) Statistique de moyenne par lycée d'origine des élèves écrite sur le terminal ;
2) Comme on est en prépa, il faudra créer un rapport automatique pour le groupe (fichier `.txt`) :
   a) Désignant le major (premier) dans chacune des matières et en moyenne générale ;
   b) Spécifiant les trois premiers qui iront en classe étoile et leurs moyennes générales ;
   c) Spécifiant les six derniers au classement qui doivent quitter la prépa et leur moyennes générales.

D. Contraintes

Le langage orienté objet **doit être utilisé** et le code ne devra être qu'en `Python`.

Gestion de base de donnée : l'ajout/le retrait d'un élève/évaluation dans la base de donnée ne doit pas perturber le calcul (à condition que toutes les données soient saisies correctement un code exhaustivement *stupid proofed* n'est pas requis pour ce projet) ;

Flexibilité du code : L'usager peut également modifier durant l'exécution du code le coefficient d'une évaluation qu'il juge trop sévère ou trop peu (on est en prépa !) pour voir ce que cela donne (il ne modifie pas la base de donnée mais "expérimente" en bon prof) ;

Normalisation : l'usager peut également faire en sorte que le premier d'une évaluation aie toujours $19/20$ (pour que les moyennes soient significatives[2]) et que les autres aient un ≪coefficient de réajustement≫ appliqué :

$$\text{Note ajustee}_{\text{evaluation}}(eleve) = \frac{19.0}{\max\{\text{Note brute}_{\text{evaluation}}(i)\}_{i\in\text{Classe}}}\text{Note brute}_{\text{evaluation}}(eleve),$$

les notes après ≪réajustement≫ seront celles du bulletin.

1. Je la nomme ainsi car il n'est point classe de nommer une classe encapsulant des classes `Classe`, un piège classique.
2. Basé sur une histoire vraie, un certain Justin C. a déjà majoré avec un $12/20$...