Exercice 1 : Passage d'un fichier en majuscules

1) Écrire un programme qui affiche un fichier texte en majuscules.
Indication : on peut utiliser la méthode ch.upper() qui retourne une copie de la chaîne ch en majusucles

2) Écrire un programme qui transforme un fichier texte en majuscules et stocke le résultat dans un nouveau fichier. Le nom du nouveau fichier sera construit en ajoutant *"_maj"* à la fin du nom du fichier initial.

Par exemple, si le fichier initial s'appelle donnees.txt le fichier en majuscules s'appellera donnees_maj.txt

Exercice 2 : Accès indexé dans un fichier

Ecrire une fonction qui retourne la ligne numéro n d'un fichier texte dont on passe en paramètre le nom. Que se passe-t-il avec votre programme s'il y a moins de n lignes dans votre fichier ?

Exercice 3 : Ensemble de points

a) Reprendre la définition de la classe Point définie dans un exercice précédent.
b) Écrire un programme qui, à partir d'un fichier de points (disponible sur Arche), représentés par une abscisse suivie d'une ordonnée sur chaque ligne, crée une liste d'instances de la classe Point
c) Créer la liste des points distants de plus de epsilon de l'origine (0,0).
d) Stocker cette liste de points distants dans un nouveau fichier

Exercice 4 : Un nain de jardin qui voyage

Capture d'écran du site internet de l'expérience, gnomeexperiment.com

Pour appuyer leur théorie sur la gravité, une équipe de physiciens finance le tour du monde de Kern, un nain de jardin qui est pesé à différents endroits du globe.
Ainsi, à Bombay, en Inde, le nain affiche 307,56 grammes alors qu'il atteint 309,82 g en Antarctique.
Le fichier *pesees.txt* recense les lieux des différentes pesées effectuées et leurs valeurs respectives.

1) Écrire un **programme** qui **saisit et stocke** le lieu et la valeur de la dernière pesée effectuée lors du périple du nain. Le programme **demande la** valeur à l'utilisateur et **complète** le fichier qui recense les valeurs déjà relevées. Des exemples de valeurs relevées sont fournis sur Arche.

2) Écrire un **second programme** qui calcule la moyenne des valeurs stockées dans le fichier, recherche la valeur la plus petite, la plus grande et calcule la différence entre les deux.

Exercice 5 : Extraction d'adresses mail

On dispose d'un fichier contenant diverses informations textuelles, dont des adresses mail.

- si une ligne contient une adresse mail, elle ne contient que cela
- une adresse mail est repérée par la présence du caractère "@" qui n'apparaît dans aucune autre ligne
- vous disposez d'un fichier de test sur Arche

1) Définir une classe *AdresseMail* dont chaque instance (objet) a comme attribut un **identifiant** et un **nom de serveur** :

   La méthode constructeur *__init__* de la classe *AdresseMail* reçoit en paramètre une **adresse mail fournie sous forme d'une chaîne de caractères** et en extrait l'identifiant et le nom du serveur.

Par exemple, une instance de la classe AdresseMail pourra être construite à partir de la chaîne :

peter.pan@monde.imaginaire.fr

peter.pan est l'identifiant, *monde.imaginaire.fr est* le serveur

2) Ecrire les méthodes accesseurs *getIdentifiant* et *getServeur* permettant aux classes externes d'accéder aux attributs associés.

3) Ecrire les méthodes mutateurs *setIdentifiant* et *setServeur* permettant aux classes externes de modifier les attributs associés.

4) Ecrire la méthode *__repr__* qui à partir d'une instance de la classe *AdresseMail* construit de façon inverse une chaîne de caractères correspondant à l'adresse mail en entier.

5) Ecrire un **programme** qui filtre les adresses mails d'un fichier dont le nom est fourni et crée une liste d'instances de la classe *AdresseMail* correspondant à ces adresses.

6) Compléter le **programme** précédent pour que les adresses de la liste en *univ-lorraine.fr* soient modifiées en *ensg.univ-lorraine.fr*.

7) Compléter le **programme** précédent pour qu'à partir de la liste d'instances de la classe *AdresseMail* ainsi modifiées il construise un fichier contenant les adresses mail complètes des personnes.

8) Ecrire une fonction qui permet d'ajouter une adresse mail à la fin du fichier des adresses mail préalablement construit. Le prénom et le nom de l'utilisateur, ainsi que le nom du serveur sont fournis en paramètres de la fonction. L'identifiant est construit sous la forme *prenom.nom*

Exercice 6 : Dosages

Des dosages d'une substance X ont été effectués sur plusieurs échantillons.

1) La classe ***Echantillon***

a) Les instances de la classe *Echantillon* ont comme attributs un nom, le poids de l'échantillon, le poids de la substance X dosée dans cet échantillon.

Définir le constructeur *__init__* qui permet de créer une instance en connaissant son nom, le poids de l'échantillon et le poids de la substance X.

b) Définir la méthode *__repr__* pour cette classe.

c) Définir la méthode *getTeneur* qui calcule le pourcentage de la substance observée dans l'échantillon courant

2) La classe ***CampagneEchantillon***

a) Définir une classe *CampagneEchantillon*.

Le constructeur *__init__* reçoit en paramètre le nom d'un fichier de données. Dans ce fichier on trouvera le nom de la substance mesurée puis sur chaque ligne suivante le nom d'un échantillon, le poids de l'échantillon et le poids de la substance mesurée dans l'échantillon. Le constructeur crée les attributs *nom* et *lechantillons* à partir du fichier passé en paramètre. La liste des échantillons est une liste d'instances de la classe Echantillon.

Vous trouverez sur Arche un exemple de fichier contenant les dosages de calcium dans diverses eaux minérales.

b) Définir la méthode *__repr__* pour cette classe.

c) Définir une méthode *moyenne* qui calcule la moyenne des pourcentages des teneurs sur l'ensemble des échantillons.

d) Définir une méthode *filtre* qui filtre les échantillons de la campagne d'échantillonnage qui dépasse un pourcentage fourni en paramètre et les stocke dans un fichier dont le nom est également fourni en paramètre. (Par exemple, les eaux qui ont plus de 12% sont considérées comme des eaux riches en calcium.)

Exercice 7 : Filtre MNT

On souhaite réaliser un modèle d'une zone géographique imposée (exemple la Lorraine).

Pour cela, on récupère le modèle numérique de terrain de la zone (MNT ou DEM Digital Elevation Model en anglais) (cf. le cours de SIG !).

L'IGN fournit ces données gratuitement sur son site sous forme d'un fichier texte organisé en 3 colonnes X Y Z avec au choix, un pas de 250, 500 ou 1000 m mais pour toute la métropole française (y compris la Corse).

Extrait d'un tel fichier : 1010000 6401000 2690
1011000 6401000 2518
1012000 6401000 2565
1013000 6401000 2662
1014000 6401000 2726
358000 6400000 0

Le logiciel (Gocad par exemple) dans lequel les données vont être traitées ne permet pas de faire facilement de la ségrégation des points utiles, l'idéal est donc de "filtrer" ce fichier avant d'en faire l'importation.

Pour cela, on considère que la zone à étudier est rectangulaire et on donne les coordonnées "extrêmes" de cette zone (Xmin, Xmax) et (Ymin, Ymax). L'opération de filtrage consiste à sélectionner les lignes pour lesquelles X et Y sont comprises entre ces bornes.

Par exemple, la zone représentant la Lorraine pourrait être encadrée par :

Xmin = 836900 *Xmax = 1040000*
Ymin = 6751600 *Ymax = 6950000*

1) Définir une classe *PointMaillage* dont les éléments ont comme attributs X, Y, Z

2) Définir une méthode *filtre* qui retourne *True* si le point courant est compris entre les bornes définies par les 4 valeurs *Xmin*, *Xmax*, *Ymin* et *Ymax* et *False* sinon.

3) Écrire un programme qui :

- A partir d'un fichier fourni par l'utilisateur **construit** un **nouveau** fichier contenant les points compris entre les coordonnées extrêmes données par l'utilisateur. Pour ce faire, on créera pour chaque point une instance de la classe *PointMaillage* et on utilisera la méthode *filtre*.

- Et affiche :
    - le nombre de points contenus dans le fichier initial
    - et le nombre de points filtrés

On fournit un exemple de fichier récupéré gratuitement sur le site IGN avec un pas de 1000 m et sa note descriptive qui permet de comprendre comment il est constitué.

Avec ce fichier et les bornes données en exemple plus haut, on obtient 35714 points filtrés sur 571485 points au départ.

On peut trouver par ailleurs sur Geoportail : http://www.geoportail.fr/ (Menu "voir/Afficher en 2D") les coordonnées Lambert 93 des points limites de la zone d'étude.

Exercice 8 : Décompression d'une carte

Une carte constituée d'un ensemble de régions est stockée dans un fichier sous forme compressée, chaque ligne ayant la structure suivante :

Une ligne = $nom_1\ nb_1\ nom_2\ nb_2\ nom_3\ nb_3$ ...

où nom_i représente un nom de région et nb_i représente le nombre de points de la région nom_i sur la ligne considérée.

Ainsi par exemple, la ligne : *G 1 H 4 A 3 B 1 I 1*

décrit la ligne de la carte *:* *G H H H H A A A B I*

1) Définir une classe ***Zone*** dont les instances sont définies par deux attributs : *nom_* (chaîne de caractères) et *repet_* (entier)

a) Définir la méthode *__repr__* pour cette classe.

b) Définir une méthode *decompresse* qui retourne pour une instance de la classe *Zone* une liste dans laquelle *nom_* est répété *repet_* fois

Par exemple, si nom_ = 'A' et repet_ = 5,
la méthode decompresse retourne la liste ['A', 'A', 'A', 'A', 'A']

2) Définir une classe ***Ligne*** dont les instances sont définies par un attribut *lzones_* qui contient une liste d'instances de la classe *Zone.*

Le constructeur *__init__* de cette classe reçoit en argument une chaîne de caractères de la forme "*G 1 H 4 A 3 B 1 I 1*" et construit l'attribut *lzones_* comme une liste **d'instances de la classe Zone.**

a) Définir la méthode *__repr__* pour cette classe.

b) Définir une méthode *decompresse* qui retourne la ligne décompressée (c'est à dire la concaténation de chaque zone décompressée de *lzones_*), par exemple ['G', 'H', 'H', 'H', 'H', 'A', 'A', 'A', 'B', 'I'].

Cette méthode utilisera bien sûr la méthode *decompresse* de la classe *Zone*.

3) Définir une classe ***Carte***. Son constructeur *__init__* reçoit en paramètre un **nom de fichier** dans lequel la carte est stockée sous forme compressée. Une instance de la classe Carte a pour attributs :

a) *nom_fichier* qui contient le nom du fichier fourni en paramètre

b) *matrice*_carte_ correspondant à la matrice de la carte telle que *matrice_carte_[i][j] = region de la case i,j*

La matrice *matrice_carte* est construite à partir des lignes du fichier dont le nom est fourni en paramètre. Pour ce faire, à partir du fichier, on peut procéder ainsi :

- on crée pour chaque ligne un objet de la classe *Ligne*
- on décompresse chaque ligne au fur et à mesure
- on stocke chaque ligne décompressée dans la matrice

c) Définir la méthode *__repr__* pour cette classe.

d) Ecrire une méthode *sauvegarde* qui sauvegarde la carte décompressée dans un fichier.

Chaque ligne de ce fichier aura la forme G H H H H A A A B I

On donnera au nouveau fichier le même nom que le fichier initial suivi de *"_decompresse*".

Exercice 9 : Rendre la monnaie

Un distributeur de tickets de stationnement peut rendre la monnaie, relativement à un prix de stationnement choisi par l'utilisateur et relativement à une somme versée par ce dernier.

Ce programme se voulant international, la liste des valeurs des pièces disponibles pour le pays traité est stockée dans un fichier *Monnaie*. Pour faciliter les traitements ultérieurs, les valeurs des pièces sont enregistrées en centimes.

Exemple : Si le fichier Monnaie contient : 200 100 50 20
le système rend la monnaie avec 4 types de pièces de valeur : 2 – 1 – 0.5 – 0.2€

Le nombre de pièces de chaque sorte disponibles est stocké dans un fichier *Caisse*.

Exemple : Si le fichier Caisse contient : 20 33 3 10

cela signifie qu'il reste en caisse 20 pièces de 2€, 33 pièces de 1€, 3 pièces de 0.5€ et 10 pièces de 0.2€.

- Définir une classe *Distributeur* et sa méthode *__init__* qui modélise un distributeur. Les éléments de cette classe sont construits à partir de deux attributs : le nom du fichier qui contient les types de pièces et le nom du fichier qui contient le stock de chaque pièce.
- Écrire les méthodes qui permettent de construire à partir des fichiers précédents, deux nouveaux attributs : la liste des types de pièces et la liste du stock de chaque pièce.
- Écrire une méthode qui, pour une somme à payer *sap* et une somme versée *sv* supérieure ou égale à *sd*, retourne la liste du nombre de pièces de chaque sorte à rendre en utilisant le moins de pièces possibles. Les sommes seront fournies en centimes également. Le nombre de pièces à rendre doit tenir compte du nombre de pièces en stock. Cette méthode mettra également à jour le fichier du stock.

Exercice 10 : Gestion de scolarité

Le service de scolarité d'une école veut informatiser la gestion des notes des élèves.
On suppose que ce service dispose de 2 fichiers :

- le fichier des noms des matières contenant sur chaque ligne :
  nom_matière coefficient
- le fichier des notes des élèves, contenant sur chaque ligne :
  nom_élève note1 note2 note3 ...
  Les notes sont fournies dans le même ordre que les matières.

Exemple :

Fichier Eleves.dta	*Fichier Matieres.dta*
Vermet 12 1 16 12	*chimie 2*
Lavest 3 12 18 14	*geol 5*
Durand 8 2 15 16	*math 4*
Bauer 10 8 12 11	*physique 3*

1) Définir une classe *Eleve* :

- Le constructeur *__init__* reçoit en paramètre le nom de l'élève et une liste de notes (attributs d'un élève)
- Écrire les méthodes assesseurs et mutateurs *getNom, getLNotes, setNom* et *setLNotes.*
- Écrire la méthode *calculeMoyenne* qui calcule la moyenne pondérée d'un élève à partir de la liste des coefficients passée en paramètre.

2) Définir une classe *Promo* :

- Le constructeur *__init__* reçoit en paramètre le nom de la promo, le nom du fichier qui contient le nom des élèves et leurs notes et le nom du fichier qui contient la liste des matières avec leurs coefficients et les définit comme attributs d'une promo.
- Écrire la méthode *creeListeEleves* qui crée les nouveaux attributs *listeEleves* et *nbeleves* à partir du fichier des élèves. *listeEleves* est constituée d'une liste **d'instances** de la classe *Eleve* et *nbeleves* contient le nombre d'élèves de la promo.
- Ajouter un appel à cette fonction dans le constructeur*__init__.*
- Écrire la méthode *creeListeMatieres* qui crée le nouvel attribut *listeMatieres* à partir du fichier des matières. *listeMatieres* est une liste composée de couples (*nomMatiere*, *coefficient*).
- Ajouter un appel à cette fonction dans le constructeur *__init__*
- Écrire une méthode *calculeLesMoyennes* qui calcule les moyennes de tous les élèves d'une promo et les stocke dans un nouveau fichier. Ce fichier contiendra sur chaque ligne le nom de l'élève et sa moyenne.

3) Écrire une méthode *filtreNotesMatiere* qui construit un nouveau fichier contenant les notes de tous les élèves dans une matière donnée. Le nom du fichier résultat sera imposé sous la forme : *notes_nomDeLaMatiere.txt*
Par exemple : "notes#anglais.txt"

4) Écrire une méthode *filtreElevesMoyens* qui affiche tous les élèves dont la note dans une matière est inférieure à la moyenne générale de l'ensemble des élèves dans cette matière.
Indication :

- *Il faut dans un premier temps parcourir le fichier des notes dans la matière pour calculer la moyenne*
- *avant de parcourir à nouveau ce fichier pour effectuer le filtrage demandé dont la moyenne. Pour revenir au début du fichier des élèves on peut utiliser la fonction : feleves.seek(0)*