TP : *CS 54 - Problem Solving 1*

TOP : Techniques and tOols for Programming – 1A

Ce premier TP de 4 heures a deux objectifs principaux et deux objectifs annexes. Les objectifs principaux sont :

1. réaliser quelques premiers *Problem Solving* sur des problèmes simples résolvables de plusieurs manières,
2. implémenter les solutions en Python ?

Les objectifs annexes sont :

— se familiariser individuellement avec les environnements utilisés à l'école : gitlab, venv, Visual Studio Code,

— s'échauffer pour le premier examen sur machine.

Les seules structures de données nécessaires dans ce TP sont les listes. Leur maîtrise est indispensable pour tous les TD/TP qui suivront et bien sûr, pour l'examen du 8 octobre pour lequel il faudra également maîtriser les bases de GIT et les entrées sorties (lire et écrire dans un fichier).

Propos liminaires

Vous disposez de 4 heures pour réaliser tout ou partie tu TP. Prenez le temps de lire les problèmes, de les comprendre, de réfléchir à des solutions. Il existe de multiples manières de résoudre les problèmes posés. Tentez, explorez. Aucune solution à un problème n'est mauvaise !.

Si la réflexion collaborative est essentielle, les TPs de CS54 sont également conçus pour que chaque élève puisse acquérir les compétences et connaissances nécessaires à la programmation et à l'utilisation des plateformes logicielles professionnelles sous-jacentes (git par exemple). Sachant que les évaluations sont individuelles, il est impératif que vous passiez individuellement le temps nécessaire pour maîtriser et pratiquer les outils et langages de programmation.

★ **Exercice 1: Préparation de l'environnement de travail**

▷ **Question 1:** Récupérez (par clonage) une version locale du projet git qui vous a été affecté pour ce TP.

À chaque étape du TP, nous vous demandons de déposer vos réalisations sur le git du projet que vous venez de récupérer. Il vous est également demandé de ne mettre qu'une seule fonction Python par fichier. Chaque fichier portera le nom de la fonction qu'il contient.

▷ **Question 2:** Référrez vous à la page Arche de CS54 pour les bases de Git et GitLab (`https://arche.univ-lorraine.fr/mod/page/view.php?id=1252876`).

▷ **Question 3:** Créez un environnement virtuel python `venv` dans le répertoire de votre projet. Vous trouverez la documentation sur l'environnement venv à l'adresse suivante : `https://docs.python.org/fr/3/library/venv.html`.

★ **Exercice 2:** Les nombres cools

Abe est très organisé. Dans sa bibliothèque, tous les livres sont classés du plus petit au plus grand (en nombre de pages), ses trophées sportifs sont alignés sur l'étagère de sa chambre du plus petit au plus grand, bref tout est parfait. Récemment, Abe à compté ses briques Lego (il en a beaucoup) et s'est rendu compte que certains nombres qu'il rencontrait avaient une caractéristique particulière : tous les chiffres qu'ils contiennent sont ordonnés de façon croissante de gauche à droite. Par exemple, `1134668` ou `349` sont de tels nombres. Fier de sa découverte et soucieux de les classifier, Abe décide de les nommer "les nombres cool". Ce sont ces nombres qui nous intéressent.

▷ **Question 1:** Ecrire un algorithme simple qui, étant donne un nombre donné, renvoie le premier nombre cool qui le précède. Une méthode simple est réalisable en traitant chaque nombre comm une liste de chiffres.

▷ **Question 2:** Implémentez votre algorithme en Python sur les bases suivantes : - vous disposer d'un fichier `in.txt` qui comprend plusieurs lignes. La première ligne vous indique combien de lignes utiles sont à considérer. Chaque ligne suivante contient un nombre dont le précédent cool est recherché.

Vous trouverez ci-dessous un exemple de fichier d'entrée :

```
3
1354
112367
9999288126549867676
```

La première ligne nous dit qu'il y a 3 nombres à traiter. Les lignes 2 à 4 contiennent chacune le nombre à traiter.

En sortie, votre programme génère un fichier `out.txt` qui contient (1) le nombre de lignes puis, pour chaque ligne, le résultat du calcul du nombre cool. Cela ressemble à :

```
3
1349
112367
8999999999999999999
```

▷ **Question 3:** Ecrire un programme qui trouve tous les nombres cools qui précèdent le nombre donné. Les afficher.

★ **Exercice 3:** Le TwoSum

Le problème *Two-Sum* est un problème classique des entretiens techniques de recrutement (il est d'ailleurs repris dans de nombreux TDs et TPs d'algorithmique dans les plus grandes universités mondiales comme Stanford). Sa formulation est extrêmement simple : Vous disposez d'une liste de n entiers et un nombre k. Déterminer, si elle existe, une paire d'éléments dans le tableau dont la somme est exactement k.

▷ **Question 1:** Commencez par analyser le problème. Quels sont les paramètres de votre programme ? que doit-t-il renvoyer ?

▷ **Question 2:** La formulation initiale du problème est-t-elle suffisamment précise ? Avez-vous des questions complémentaires dont les réponses vous permettront de construire une fonction robuste qui va fournir une solution qui répond aux attentes du client ?

Si oui, posez-les à votre GCC (compilateur C de GNU mais également, et uniquement à TNCY, Gentil Coach de Code !).

▷ **Question 3:** En ne vous servant que de boucles, écrivez une fonction dont le nom et le profil vous sont donnés ci-dessous qui résout le problème. Les paramètres de cette fonction devront répondre aux attentes de vos clients. Pour cela, proposez une signature de fonction, motivez là et faite la valider avant de démarrer votre codage. La fonction sera définie dans un fichier nommé `twosum.py`.

```
1 def twoSum(...):
```

▷ **Question 4:** Quelle est la complexité algorithmique du calcul ?

▷ **Question 5:** Si la réponse précédente est quadratique ou supérieure, vous pouvez mieux faire... Faites mieux !

▷ **Question 6:** Réécrivez votre programme afin qu'il prenne en entrée un fichier in.txt défini comme suit :

```
3
4,3,7,1,5
12,5,8,3,23,11,0,4
43,6,18,7,4
```

Le nombre dans la première ligne comprend le nombre de tests (ici 3). Sur les lignes suivantes (celles des tests), le premier nombre représente la valeur recherchée ; les autres représentent les valeurs présentes dans la liste. Par exemple dans le deuxième test, on recherche une somme de 12 dans une liste comprenant les valeurs 5,8,3,23,11,0,4.

▷ **Question 7:** En utilisant la librairie `timeit`, mesurer les temps d'exécution des différents algorithmes que vous avez implémentés dans les questions précédentes. Comparez les temps.

★ **Exercice 4:** Le mélange du savant O(u)F

ll existe de multiples manières de mélanger les cartes au sein d'un jeu mais à Poudlard, un seul type de mélange est autorisé : le "single riffle shuffle". Son principe est extrêmement simple : prenez un jeu de cartes, coupez le en deux tas (pas nécessairement de taille égale), mettez les deux tas côte à côte et fusionnez les en une opération pour obtenir un tas mélangé (voir illustration ci-dessous).

Si la règle est globalement respectée, Albus soupçonne certains de ses collègues de tricher en mélangeant les cartes. C'est pour tirer cela au clair, qu'il recrute des experts en numérique. Fraîchement recrutés à Poudlard, vous devez donc concevoir un algorithme qui vérifie que le processus de mélange des cartes en vigueur dans l'établissement est strictement respecté.

▷ **Question 1:** Comment pouvez vous faire cela ? (vous pouvez réfléchir par groupes de 2 à 4 élèves sur la modélisation)

▷ **Question 2:** Ecrivez chacune et chacun une fonction qui indique si après un mélange, la procédure du single riffle shuffle a été respectée

▷ **Question 3:** Quelle est la complexité de l'algorithme ?

▷ **Question 4:** Ecrivez une fonction qui réalise un mélange de cartes respectueux du Single Riffle Shuffle

▷ **Question 5:** Un bon magicien sait réaliser ce mélange de façon exemplaire (en faisant "claquer" les cartes). Proposer quelques métriques pour analyser la qualité d'un mélange. Testez les sur différentes expériences de mélanges.