Devoir en temps libre n° 2

Ce deuxième devoir en temps libre est à préparer pour le jeudi 8 novembre 2018 et sera à rendre pendant le TP, à 13h15 (vous aurez 15 minutes pour prendre connaissance de vos éventuelles erreurs et pour les corriger). Seul le code Python est à rendre, il n'y a rien à rédiger sur feuille. Vous devez apporter votre programme Python sur une clé USB *et* vous l'être envoyé par mail depuis chez vous.

Vous devez télécharger l'archive `dm2.zip` sur le site `https://cpge.info`. Il faut ensuite compléter le fichier `dm2.py` qui se trouve dans l'archive une fois décompressée, *sans en modifier le nom*. L'encodage du fichier devra être UTF-8 (*Fichier* > *Encodage* > utf-8 sous Pyzo). Le code devra impérativement compiler : *aucune* erreur ne doit apparaître lorsque l'on exécute le script. Il est impératif de veiller au strict respect de toutes ces consignes.

Vous pouvez vérifier que vos fonctions sont correctement prises en compte et passent les premiers tests (il y en aura d'autres) en exécutant le fichier `verif_dm2.py` qui se trouve dans l'archive décompressée. Attention, il faut bien que votre code `dm2.py` se trouve dans cette même archive et, sous Pyzo, choisir « *Démarrer le script* », que je vous conseille de lier à F5 chez vous si ce n'est pas déjà fait.

Ce script de vérification permet aussi de vérifier que vous respectez les conventions et les bonnes pratiques en vigueur. Avant de commencer le TP, il faut aller lire l'article

`https://sup3.prepa-carnot.fr/pep8`

Vous pouvez ensuite installer PyLint. Pour cela, dans la console IPython de Pyzo ou dans une console Unix entrer la commande `pip install pylint` (une connexion internet est nécessaire). Le script de vérification vous donne une note sur 10 pour le style et vous indique tout ce qui ne va pas dans votre code. Si vous ne comprenez pas un des messages, vous pouvez aller regarder sur `http://pylint-messages.wikidot.com/all-codes`. Si vraiment vous ne voyez pas, vous pouvez déposer un message sur le forum de la classe, m'envoyer un mail ou nous demander le jour du rendu.

✔

On prendra bien soin de tester toutes les fonctions sur des cas triviaux, puis un peu plus complexes. La vérification ne garantit aucunement que vos fonctions sont correctes.

Exercice 1 (*Échauffement*)

1. Écrire une fonction `nombre_impairs_divisibles_par_7(n)` qui renvoie le nombre d'entiers impairs compris entre 1 et n divisibles par 7.
2. Écrire une fonction `somme_impairs_divisibles_par_7(n)` qui renvoie la somme des entiers impairs compris entre 1 et n divisibles par 7.
3. (*Bonus*) `https://projecteuler.net/problem=1`

Exercice 2 *Liste d'éléments positifs*

Écrire une fonction `tous_positifs(liste)` qui prend en entrée une liste `liste` de nombres et qui renvoie `True` si tous les termes de la liste sont positifs ou nuls et `False` dans le cas contraire.

Exercice 3 *`https://projecteuler.net/problem=48`*

On peut calculer la somme $1^1 + 2^2 + 3^3 + \cdots + 10^{10} = 10405071317$.

Écrire une fonction `probleme_48(n)` qui renvoie le nombre constitué des 10 derniers chiffres de la somme $1^1 + 2^2 + 3^3 + \cdots + n^n$.

Exercice 4 *Liste des petits et des grands*

Écrire une fonction `separe_petits_grands(liste, seuil)` qui renvoie un couple de listes `(petits, grands)` contenant, dans le même ordre relatif, les éléments plus petits, respectivement strictement plus grands, que la valeur `seuil` dans une liste de nombres donnée en argument.

Exercice 5 *Achat d'ingrédients*

Les stocks des ingrédients nécessaires à la réalisation d'un onguent commencent à se vider et les savants vous chargent d'aller en ville les réapprovisionner d'une certaine quantité de chaque ingrédient. Le comptable étant particulièrement pointilleux, il vous donnera exactement la quantité d'argent dont vous avez besoin, arrondie à l'euro supérieur, mais pas une pièce de plus. Heureusement, vous connaissez à l'avance le prix au kilogramme de chaque ingrédient et la quantité exacte dont vous avez besoin.

Écrire une fonction `prix_total(prix_au_kg, masses_voulues)` qui prend en argument deux listes qui représentent respectivement les prix au kilogramme de chaque ingrédient et les masses (en kilogrammes) nécessaires à la fabrication de l'onguent. Elle doit renvoyer la quantité totale d'argent à demander au comptable. Par exemple, si l'on souhaite 200g de salsepareille à 90€ le kilogramme et 250g de poudre de baobab à 61.33€ le kilogramme, on utilise l'appel `prix_total([90, 61.33], [0.2, 0.25])` qui renvoie la réponse 34.

Exercice 6 *Carré magique*

Un carré magique de taille $n \in \mathbb{N}^*$ est une grille carrée, représentée par une liste de listes, dans laquelle des entiers entre 1 et n^2 sont placés de telle sorte que la somme des nombres de chaque colonne, chaque ligne et de chacune des deux diagonales soit la même. De plus, le carré doit contenir exactement une fois chaque entier de 1 à n^2.

Écrire une fonction `est_carre_magique` qui vérifie si une grille d'entiers fournie en argument est un carré magique. On suppose que la grille représente bien un carré de côté non nul, mais on ne suppose rien *a priori* sur les entiers.

Par exemple,

Python
```
carre = [[2, 9, 4],
         [7, 5, 3],
         [6, 1, 8]]

est_carre_magique(carre)
 ↪ True
```

Python
```
carre = [[1, 19, 4],
         [7, 19, 3],
         [4, -1, 8]]

est_carre_magique(carre)
 ↪ False
```

Exercice 7 *Tricheurs au cinéma*

Vous êtes employé dans un cinéma et votre patron décide de lancer une offre spéciale. Toute personne possédant une carte de fidélité a le droit, pendant un mois, de voir un film gratuit par jour. Bien entendu certaines personnes vont essayer de tricher en venant plusieurs fois au cinéma dans la même journée et votre travail consiste à détecter ces tricheurs.

Écrire une fonction `detection_tricheur` qui prend en entrée la liste des numéros de carte utilisés dans la journée (supposée non vide) et qui renvoie le numéro d'abonné de la personne qui a le plus triché (s'il y en a au moins un) ou `None` (sinon). Si plusieurs personnes ont triché un même nombre de fois, on demande le numéro du tricheur qui est venu le premier dans la journée. Par exemple `detection_tricheur([18, 20, 33, 20, 4, 33])` doit renvoyer `20`. On garantit que les numéros sont tous des entiers positifs « pas trop grands » (de sorte à pouvoir facilement être utilisés comme indices d'une liste par exemple). On pourra librement utiliser la fonction `max`.

Exercice 8 *Prison de Sikinia*

Dans la prison centrale de Sikinia, il y a 30 cellules numérotées entre 1 et 30, toutes occupées. Les portes des cellules peuvent être dans deux états : ouvertes ou fermées. On peut passer d'un état à l'autre en faisant faire un demi-tour au bouton de la porte. Au moment où commence l'histoire, toutes les portes sont fermées.

Pour fêter le vingtième anniversaire de la république de Sikinia, le président décide d'une amnistie. Il donne au directeur de la prison les ordres suivants :

> *« Tournez successivement d'un demi-tour les boutons :*
> 1. *de toutes les portes,*
> 2. *puis d'une porte sur deux, à partir de la deuxième,*
> 3. *puis d'une porte sur trois, à partir de la troisième,*
> 4. *puis d'une porte sur quatre, à partir de la quatrième.*
> 5. *...*
>
> *Continuez ainsi jusqu'à la dernière cellule. Libérez alors les prisonniers dont la porte de cellule est ouverte. »*

Pour des raisons de sécurité, le directeur de la prison aimerait connaître à l'avance quels seront les prisonniers libérés. Écrire un programme plus général`liberation_prisonniers(n)` qui prend en argument le nombre de cellules de la prison (numérotées de 1 à n, avec $n > 0$) et qui doit renvoyer la liste des numéros des cellules ouvertes, dans l'ordre croissant.

Exercice 9 *Suite de* Conway

Les premiers termes de la suite de Conway sont : 1, 11, 21, 1211, 111221. Chaque terme étant obtenu en lisant à haute voix le terme précédent, c'est pourquoi Conway avait baptisé cette suite *Look and say*. Par exemple, le terme 1211 se lit « un 1, un 2, deux 1 » donc le terme suivant est 111221.

1. Écrire une fonction `look_and_say(terme)` qui prend en entrée une chaîne de caractères quelconque mais non vide de chiffres entre 0 et 9 et qui retourne le terme suivant en appliquant ce processus. Par exemple, `look_and_say("046666911111")` doit renvoyer `"1014461951"`.
2. Écrire une fonction `suite_de_conway(n)` qui renvoie la liste des n premiers termes (sous forme de chaînes de caractères) de la suite de Conway de premier terme 1.

Si on note u_n le nombre de chiffres du n^e terme de cette suite alors on peut montrer que le rapport $\frac{u_{n+1}}{u_n}$ tend vers une constante, appelée constante de Conway.

3. Calculer une valeur approchée de la constante de Conway en évaluant $\frac{u_{41}}{u_{40}}$ que l'on mettra dans une variable globale `CONSTANTE_DE_CONWAY`.

Une propriété remarquable de cette constante est qu'elle ne dépend pas de la valeur initiale (à l'exception de 22 qui produit une suite constante).

4. (*Bonus*) Le vérifier expérimentalement en testant différentes valeurs initiales.
5. (*Bonus*) Montrer que dans la suite de Conway de premier terme 1 ne peuvent apparaître que les chiffres 1, 2 et 3.