[Q1] Formule de Faulhaber

En mathématiques, *la formule de Faulhaber*, nommée en l'honneur du mathématicien allemand Johann Faulhaber, exprime la somme

$$\sum_{k=1}^{n} k^p = 1^p + 2^p + 3^p + \ldots + n^p$$

où les paramètres `n` et `p` sont des entiers positifs et `n` est strictement positif. Par exemple, pour `n = 6` et `p = 2` on obtient la somme

$$1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 = 91$$

On vous demande d'écrire le corps d'une fonction `faulhaber_max(x,p)` qui retourne, pour un nombre entier `x > 0` et une valeur de `p ≥ 0` donnée, le plus grand entier `n > 0` pour laquelle le résultat de la formule de Faulhaber est strictement plus petit que x `(< x)`. Par exemple, `print(faulhaber_max(120,2))` imprime la valeur 6

car

$$1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 = 91 < 120$$

[Q2] Occurrences de caractères

Étant donnée une liste `l` de chaînes de caractères, écrivez le corps d'une fonction `caracteres_occurrences(l)` qui retourne un dictionnaire contenant, pour chaque caractère dans les différentes chaînes de caractères, (en ordre croissant) les indices des chaînes qui contiennent le caractère au moins une fois. Par exemple,

```
l = ["ceci n'est pas une pipe", "le fils de l'homme", "golconda"]
print(caracteres_occurrences(l))
```

imprime

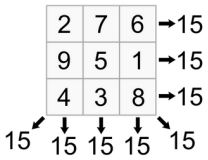
```
{'c': [0,2], 'e': [0,1], 'i': [0,1], ' ': [0,1], 'n': [0,2], "'": [0,1], 's': [0,1], 't': [0], 'p': [0], 'a': [0,2], 'u': [0], 'f': [1], 'l': [1,2], 'd': [1,2], 'h': [1], 'o': [1,2], 'm': [1], 'g': [2]}
```

car:

- le caractère `'c'` apparaît au moins une fois dans la première chaîne `"ceci n'est pas une pipe"` (à l'indice `0` de la liste `l`) ainsi que dans la troisième chaîne `"golconda"` (à l'indice `2` de la liste `l`);
- le caractère `'e'` apparaît au moins une fois dans les première et deuxième chaînes (respectivement aux indices `0` et `1` de `l`);
- et ainsi de suite.

[Q3] Carré magique

En mathématiques, un carré magique d'ordre `n > 0` est composé de `n*n` entiers strictement positifs, écrits sous la forme d'un tableau carré. Ces nombres sont disposés de sorte que leurs sommes sur chaque rangée, sur chaque colonne et sur chaque diagonale principale soient égales. On nomme alors *constante magique* la valeur de ces sommes. Voici un exemple d'un carré magique d'ordre 3, avec comme constante magique `15` :



En Python, on peut représenter ce carré magique d'ordre 3 comme une liste de listes, ou chaque liste imbriquée représente une rangée :

```
carre3 = [ [ 2, 7, 6 ], [ 9, 5, 1 ], [ 4, 3, 8 ] ]
```

Implémentez une fonction `carre_magique(carre)` qui prend comme argument `carre` une liste (non-vide) de listes

Question 1: carre_magique

```
def carre_magique(carre):
    """
    @pre: carre est une liste de taille n > 0 dont chaque element est une liste
          de taille n, contenant des entiers strictement positifs
    @post: retourne la constante magique si cette liste de listes represente un
          carre magique, retourne False sinon.
          carre n'est pas modifie.
    """
    # a completer
```

[Q4] Numéros de comptes

En Belgique, les numéros de compte en banque se composent de 12 chiffres. Les 3 premiers chiffres permettent d'identifier la banque, les 7 chiffres suivants constituent le véritable numéro du compte au sein de la banque, les 2 derniers chiffres servent à vérifier la validité du numéro ; les chiffres sont séparés par le symbole `'-'`.

Un exemple de numéro de compte en banque *valide* est la chaîne `"068-2492526-41"`. Un numéro est *valide* si le résultat de l'opération des 10 premiers chiffres modulo `97` donne les deux derniers chiffres. Par exemple, pour le numéro de compte en banque `"068-2492526-41"`, le nombre correspondant aux dix premiers chiffres est `682492526`. Ce nombre modulo `97` donne `41`, ce qui correspond exactement au deux derniers chiffres du compte. Ce numéro de compte est donc *valide*.

[Q4a] Implémentez une fonction `compte_valide(s)`, qui prend comme argument une chaîne de caractères `s` représentant un numéro de compte potentiel dans le format `"xxx-xxxxxxx-xx"` (une chaîne de caractères commençant par 3 chiffres, suivi par `"-"`, suivi par 7 chiffres, suivi par `"-"`, suivi par encore 2 chiffres). La fonction retourne `True` si cette chaîne représente un numéro de compte *valide*. Sinon, elle retourne `False`.

[Q4b] Implémentez ensuite une fonction `verifier_fichier(f)`, qui prend comme argument un nom de fichier `f`. La fonction vérifie si, sur chaque ligne, le fichier `f` contient un numéro de compte en banque valide. La fonction retourne `True` si chaque ligne du fichier contient un numéro de compte valide. Sinon, ou si la lecture du fichier produit une erreur, la fonction retourne `False`.

Attention : aucun autre caractère superflu n'est permis sur la même ligne à l'exception des espaces avant ou après le numéro de compte. Par exemple, une ligne contenant
`"  068-2492526-41 "`
sera acceptée mais
`"  068-2492526-41  blabla"`
ou
`" haha 068-2492526-41"`
ne le seront pas.

[Q5] Centre météorologique

Le Centre Météorologique de Louvain veut garder une trace journalière des températures mesurées à Louvain-la-Neuve. Ces mesures sont stockées dans une liste chaînée triée, avec les mesures les plus récentes en tête de la liste et les mesures moins récentes en queue de la liste.

Trois classes (`Temperatures`, `Mesure` et `Date`) sont déjà implémentées partiellement. Vous pouvez consulter les sources de ces différentes classes ici. Lisez-les en détail avant de commencer à implémenter votre solution.

Initialement, la liste de températures mesurées est vide :

```
temperatures = Temperatures()
```

[Q5a] Complétez la méthode `ajoute_mesure(annee,mois,jour,heure,temperature)` dans la classe `Temperatures` pour ajouter une nouvelle température mesurée à la liste chaînée. Vous pouvez supposer que cette nouvelle mesure sera toujours plus récente que celles déjà présentes. **Attention** : cette méthode doit aussi mettre à jour la température moyenne de toutes les températures déjà mesurées à Louvain-la-Neuve.

Vous êtes encouragés à ajouter des fonctions auxiliaires aux différentes classes selon vos besoins.

Avec cette méthode on peut maintenant ajouter des nouvelles températures mesurées à la chaîne :

```
temperatures.ajoute_mesure(2019,1,1, 0,12.0)
temperatures.ajoute_mesure(2019,1,1,12,14.0)
temperatures.ajoute_mesure(2019,1,2, 0,15.0)
temperatures.ajoute_mesure(2019,1,2,12,16.0)
```

(Il manque les sources des différentes classes – Températures, Mesure et Date)

[Q6] Somme d'entiers

Etant donné une liste d'entiers `l`, par exemple `l = [3,4,7,9,11,12]`, écrivez une fonction **récursive** `somme_sous_ensemble(l,n)` qui vérifie si la somme d'un sous-ensemble quelconque des entiers contenus dans `l` est égal à `n`. La fonction retourne `True` si c'est le cas et `False` sinon. Si la liste `l` est vide on peut considérer que la somme de ses éléments est égale à zéro.

Exemples :

- `print(somme_sous_ensemble([3,4,7,9,11,12],10))` imprime `True` car 3 + 7 = 10
- `print(somme_sous_ensemble([3,4,7,9,11,12],27))` imprime `True` car 7 + 9 + 11 = 27
- `print(somme_sous_ensemble([3,4,7,9,11,12],12))` imprime `True` car 12 = 12
- `print(somme_sous_ensemble([],0))` imprime `True` car la somme des éléments d'une liste vide est 0
- mais `print(somme_sous_ensemble([3,4,7,9,11,12],17))` imprime `False` car la somme d'aucun sous-ensemble de la liste `[3,4,7,9,11,12]` ne peut produire la valeur `17`.

Définissez maintenant cette fonction `somme_sous_ensemble(l,n)`. **Attention** : votre implémentation doit être *récursive*.