TP1 POO - Python

simon.girel@univ-cotedazur.fr, gilles.scarella@univ-cotedazur.fr

Consignes:

- Pour tester vos codes sur les stations du PV, vous pouvez, par exemple:
  - Lire préalablement l'aide-mémoire Linux se trouvant sous Moodle.
  - exécuter vos fichiers dans ipython3 en faisant
    %run mon_fichier.py
  - exécuter, dans un terminal Linux
    python3 mon_fichier.py
  - écrire et exécuter le code depuis *jupyter* en lançant
    /usr/local/anaconda3/bin/jupyter notebook & (n'oubliez pas de sauvegarder le notebook!)
  - Utiliser le logiciel Spyder
- L'aide pour installer/configurer python sur votre machine se trouve dans Moodle dans le fichier installation_python.pdf

1 Conversion de l'heure

Dans un fichier python *conv_heure.py*, créer une fonction python *conv_heure* avec un seul argument t, désignant un nombre de secondes, qui retourne une chaîne de caractères contenant l'heure au format 'jours', 'heures', 'minutes' et 'secondes'. Dans ce même fichier python, tester la fonction pour plusieurs valeurs de t comme ci-après.

On soignera l'affichage pour obtenir, par exemple

```
>>> conv_heure(3601)
'3601 secondes correspondent à 0 jours, 1 heures,  0
   minutes,  1 secondes'
>>> conv_heure(17112)
'17112 secondes correspondent à 0 jours, 4 heures, 45
   minutes, 12 secondes'
>>> conv_heure(156007)
'156007 secondes correspondent à 1 jours, 19 heures,
   20 minutes, 7 secondes'
```

2 Chaînes de caractères

On considère le texte suivant issu d'un poème célèbre. On définit la variable python T_0 telle que (la copier par ligne au besoin)

```
T0 = "Les sanglots longs des violons de l'automne
   blessent mon coeur d'une langueur monotone. Tout
   suffocant et blême, quand sonne l'heure, je me
   souviens des jours anciens et je pleure."
```

Dans le fichier *exo2.py*,

- Afficher le nombre de caractères de la variable T_0 en le précédant par "Le nombre de caractères du texte est".
- À l'aide de la méthode *split*, définir une variable L de type liste, qui correspond au découpage de T_0 selon les blancs ("'", "," et "." ne sont pas des blancs). Faites afficher la taille de L
- Définir la variable T_1 égale à T_0 à l'exception que "jours" est remplacé par "temps" et "automne" remplacé par "hiver".

3 Dictionnaire

Créer, dans le fichier python *dict_ depart.py*, un dictionnaire nommé *depart*, dont chaque clé est le numéro du département et la valeur associée le nom du département (attention, rôles des clés et valeurs inversés par rapport à l'exemple du cours). Le dictionnaire doit contenir au moins les clés-valeurs suivantes:

```
83 - Var
38 - Isere
26 - Drome
84 - Vaucluse
29 - Finistere
18 - Cher
```

- Créer et afficher une liste contenant les valeurs de *depart* dont les clés sont strictement inférieures à 30, dans l'ordre croissant des clés.
- Créer et afficher une liste contenant les valeurs de *depart* contenant au moins 6 lettres, dans l'ordre croissant des clés.
- Créer et afficher une liste contenant les valeurs de *depart* contenant au plus 5 lettres, dans l'ordre décroissant des clés (on pourra par exemple utiliser l'argument `reverse=True` de la méthode `sort`, ou une boucle `for`).

4 Discrétisation

Dans cet exercice, on veut discrétiser plusieurs intervalles. Définir dans un fichier python **discretisation.py**,

- une variable a, qui discrétise l'intervalle $[1, 30]$ avec un pas égal à 1 (on veut donc 30 valeurs). Pour vérification, faire afficher le contenu de la variable (utiliser éventuellement une boucle for).
- une variable b, qui discrétise l'intervalle $[0, 100]$ avec un pas égal à 4. Faire afficher le contenu de la variable (utiliser éventuellement une boucle for).
- une variable c, qui discrétise l'intervalle $[0, 30]$ avec un pas égal à 1, mais dans l'ordre inverse: c contient dans l'ordre 30, 29, ..., 2, 1, 0. Faire afficher le contenu de la liste (utiliser éventuellement une boucle for).
- un tableau numpy d, qui discrétise l'intervalle $[0, 5]$ avec un pas égal à 0.2 (5 doit être inclus). Faire afficher le contenu de d.
- un tableau numpy e, qui discrétise l'intervalle $[1, 6]$ en 50 points. Faire afficher le contenu de e.

5 Racines d'un polynôme de degré 2

Dans un fichier *solve2.py*, créer une fonction *solve2* ayant une liste (ou un tableau numpy) comme argument contenant les coefficients du polynôme de degré 2 (par ordre décroissant de l'exposant).

La fonction renvoie un tableau numpy contenant les éventuelles racines du polynôme et les affiche, comme montré dans la ci-dessous.

On testera la fonction sur quelques exemples et on soignera l'affichage de manière à obtenir

```
>>> s1 = solve2([1, -4, 3])
Les racines du polynome (1*x^2) + (-4*x) + (3) sont
   1.000000 et 3.000000
>>> print(s1)
[1. 3.]
>>> s2 = solve2([1, 0, 1])
Pas de racine reelle pour le polynome (1*x^2) + (0*x)
   + (1)
>>> s3 = solve2([1, 2, 1])
La racine double du polynome (1*x^2) + (2*x) + (1) est
    -1.000000
```

6 Dichotomie

On rappelle un algorithme possible de dichotomie pour calculer le zéro d'une fonction f sur $[a, b]$

Algorithm 1 Dichotomie

Usage: x = dicho(f, a, b, kmax)
Description: Calcul de x tel que f(x)=0 sur [a,b]
Input: f; fonction définie sur $[a, b]$
Input: a, b; réels
Input: kmax: Nombre maximum d'itérations
Output: x; approximation de la racine de f sur $[a, b]$

```
deb ← a, fin ← b, x ← (a+b)/2, eps ← 1e-8
for i=0:(kmax-1) do
    if (|f(x)|≤ eps) then
        break
    end if
    if (f(fin) * f(x) < 0) then
        deb ← x
    else
        fin ← x
    end if
    x ← (deb+fin)/2.
end for
return x
```

Dans un fichier dicho.py, créer une fonction dicho implémentant l'algorithme et prenant pour arguments f, a, b, kmax. kmax est facultatif et vaut par défaut 50 (on prendra eps= 1e-8 dans la fonction).

Dans ce même fichier python, tester la fonction dicho sur deux exemples:

$$\begin{array}{lll} f_1(x) & = \sqrt{1+x^3} - 7 & \text{avec } [a,b] = [-1, 50], \\ f_2(x) & = x^2 - 6\,x\,\sin(2x) + 2 & \text{avec } [a,b] = [-20, -4] \end{array}$$

7 Géométrie [exercice facultatif]

L'objectif est de déterminer la matrice de la projection sur un plan $\mathcal{P} \subset \mathbb{R}^3$, contenant le point origine, parallèlement à une droite $\mathcal{D}$ dans $\mathbb{R}^3$.

Notons que pour qu'une telle projection soit bien définie sur $\mathbb{R}^3$, il faut que $\mathcal{D} \not\subset \mathcal{P}$.

1. Écrire une fonction *Proj3D* prenant comme argument :
   - u et v (deux tableaux numpy 1D, formant une base de $\mathcal{P}$),
   - w (un tableau numpy 1D formant une base de $\mathcal{D}$).

Cette fonction doit renvoyer la matrice attendue si celle-ci est bien définie. Dans le cas contraire elle affiche *"problème mal posé"* et ne renvoie rien.

Indications :

- on pourra utiliser `np.linalg.det(M)` et `np.linalg.inv(M)`, permettant de calculer le déterminant et l'inverse d'une matrice M.
- on pourra commencer par écrire la matrice de la projection dans la base (u, v, w).
- Pour former un tableau $2D$ à partir de tableaux $1D$ rangés côte-à-côte, on peut initialiser une matrice $2D$ et remplir ses colonnes une à une, ou encore utiliser la fonction `np.stack()` avec l'option `axis=1`.

2. Tester la fonction *Proj3D* avec le plan $P = vect\{(1, 0, 1), (0, 1, -1)\}$ et la droite $D = vect\{(1, -1, 1)\}$. On doit trouver

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 2 \end{pmatrix}$$