TD4

Ibrahim ALAME

8/11/2023

1 Dictionnaire

1.1 Dictionnaire statistique

Soit `dico` un dictionnaire où les clés sont des chaînes de caractères et les valeurs sont des listes de chaînes de caractères. Tester les opérations suivantes :

```
# Définition d'un dictionnaire en plusieurs étapes:
animals = {'a': ['horse'], 'b': ['baboon'], 'c': ['giraffe']}
animals['d'] = ['donkey']
animals['d'].append('dog')
animals['d'].append('dingo')
# Affichage
print(animals)
```

- Écrire une fonction `how_many (dic)` qui renvoie la somme du nombre de valeurs associées à tout clés du dictionnaire. Dans l'exemple ci-dessus, la fonction doit renvoyer 6.
- Écrire une fonction `biggest(dic)` qui retourne la clé correspondant à l'entrée avec la plus grande nombre de valeurs qui lui sont associées. S'il y a plus d'une entrée de ce type, renvoyez l'une des clés correspondantes. Sur l'exemple ci-dessus, la fonction doit retourner 'd'.
- Écrire une fonction `dstats(dic)` qui retourne un 2-tuple constitué de la somme du nombre de valeurs dans le dictionnaire et le plus grand nombre de valeurs. Sur l'exemple ci-dessus, la fonction devrait retourner le tuple (6, 3).

```
# Nombre d'espèces dans le dictionnaire animals
def how_many(dic):
    sum = 0
    for (k,v) in dic.items():
        .........
    return sum

# Recherche du maximum
def biggest(dic):
    biggest = 0
    key = ''
    for (k,v) in dic.items():
        if len(v) > biggest:
            ...........
            ...........
    return key

```

```
def dstats(dic):
    sum = .........
    largest = .........
    return (sum,largest)

animals = {'a': ['horse'], 'b': ['baboon'], 'c': ['giraffe']}
animals['d'] = ['donkey']
animals['d'].append('dog')
animals['d'].append('dingo')

print(dstats(animals))
```

1.2 Ordre d'une date dans l'année

Écrivez un programme qui lit le mois (abréviation de 3 lettres) et le jour du mois et affiche le numéro de ce jour (un nombre compris entre 1 et 365). On suppose en premier temps que l'année bissextile puis en généralise pour une année quelconque.

Votre programme construira d'abord un dictionnaire dont les clés sont les noms des mois (`'jan'` ... `'dec'`) et dont les valeurs sont 28, 30 ou 31, et parcourra les mois en accumulant les jours.

```
# Saisir le mois et le jour
m =  input("month : ")
d = int(input("day : "))

months = {'jan':31, ..........................................., 'dec':31}
# Traitement
def day(m,d):
    days = 0
    for mois in months:
        if mois == m:
            ............
            ............
        else:
            ...........

    return days
# Affichage
print('Day of the year: ', day(m,d))
```

Exemple d'exécution :

```
mois : jul
jour : 12
Day of the year:  193
```

1.3 Index

À la fin des livres, il y a généralement un index qui répertorie les pages où un certain mot apparaît. Dans cet exercice, vous allez créer un index pour un texte mais, au lieu du numéro de page, vous utiliserez les numéros de ligne. Vous allez écrire un programme `index.py` qui lit le nom d'un fichier texte et une séquence de mots. Pour chaque mot dans la liste, votre programme trouvera les lignes dans le fichier texte où le mot apparaît et imprimera le numéros de ligne

correspondants, la numérotation commence à 1. Un mot peut apparaître plus d'une fois sur une ligne, mais ne doit être indexé qu'une seule fois par ligne. Pour l'exemple un fichier d'entrée `onteaching.txt` peut être téléchargé à partir de e-campus.

```
# Saisir le nom du fichier et la liste des mots
filename = "onteaching.txt"
L=input("Liste de mots : ")
# dict est un dictionnaire contenant les mots proposés comme clés dont les valeurs sont des listes vides
dict = {}
for k in L.split(None):
    dict[k]=[]
#Ouverture du fichier en lecture seule
f = open(filename, 'r')
#Lecture des lignes dans le fichier ouvert
Lines = f.readlines()
#Cloture du fichier
f.close
for i in range(len(Lines)):
    #Pour chaque mot de la ligne i
    Li = str(Lines[i])
    #On teste l'appartenance au dictionnaire
    for k in dict.keys():
        #Si le mot appartient au dictionnaire on ajoute le numéro da la ligne dans la liste de la valeur correspondate
        if k in Li:
            ........
#Affichage
for k,v in dict.items():
    .........
    for i in v[:-1]:
        .......
    .........
```

Exemple d'exécution :

```
> Liste de mots : wisdom knowledge understanding science
wisdom 5, 7
knowledge 2, 22, 23
understanding 10, 11, 24
science 17
```

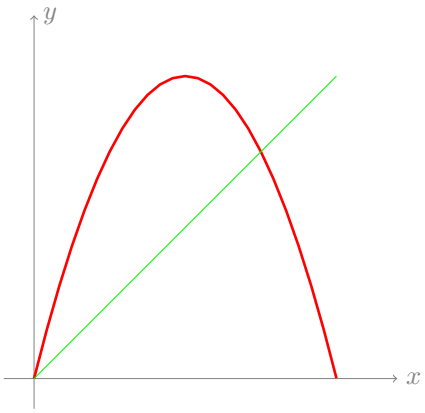
2 Manipulation élémentaire des tableaux

2.1 Suite numérique

1. Construire une liste numpy contenant les points $x = (x_i)_{i=0..n-1}$ d'une subdivision régulière de l'intervalle $[a, b]$ en $n-1$ sous intervalles (`numpy.linspace(a,b,n)`). Afficher la liste x en choisissant $a = 0$ et $b = 1$ et $n = 10$. Refaire la même liste à l'aide de la fonction (`numpy.arange(a,b,h)`) où $h = \frac{b-a}{n-1}$. Comparer les deux subdivisions.
2. Soit f une fonction réelle définie sur l'intervalle $[a, b]$ par $f(x) = 4\beta x(1-x)$. Calculer la liste $y = f(x)$ définie par $y_i = f(x_i)$ pour $i = 0, 1, ..., n-1$ et tracer la courbe de f sur l'intervalle $[a, b]$ pour $\beta = 1$. Pour la représentation graphique on utilisera le package `pyplot` de la librairie `matplotlib` :

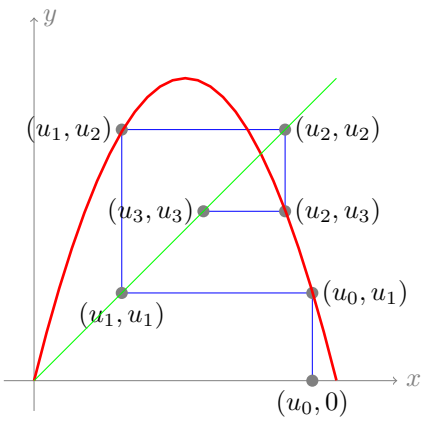
```python
import numpy as np
import matplotlib.pyplot as plt

# beta est un paramètre critique de la suite stratégique f(x)=4*beta*x*(1-x)
beta=1
# Code de la fonction considérée
def f(x):
    return 4*beta*x*(1-x)
# La fonction graphe1 permet de tracer la courbe de la fonction f sur l'
    intervalle [a,b] pour n points de discrétisation
def graphe1(f,a,b,n):
    x = np.linspace(a, b, n)
    y=f(x) # ici y est la liste des images y=(yi) où yi=f(xi)
    plt.plot(x, y, markersize=2,color='r')
    plt.show()

graphe1(f,0,1,40)
```

3. Écrire une fonction `graphe2(f,a,b,n)` permettant de tracer sur le même graphique la courbe de la fonction $f : [a, b] \to \mathbb{R}$ et la première bissectrice $y = x$ pour une discrétisation à n points. Pour le cas où $f(x) = 4x(1 - x)$ le graphique doit ressembler à ceci :

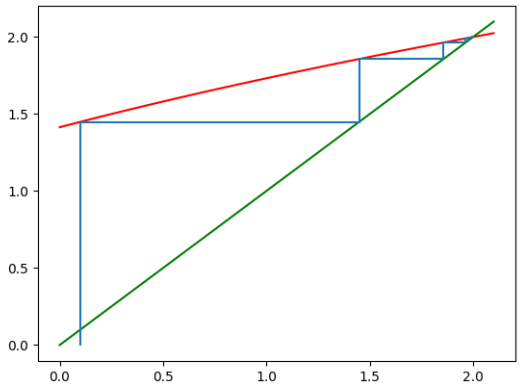


4. Soit $(u_k)_{k\in\mathbb{N}}$ une suite numérique définie par $u_{k+1} = f(u_k)$, u_0 étant donné. Créer une liste `numpy` contenant les termes de la suite $U = (u_k)_{k=0,N-1}$. Afficher U pour $N = 10, 50, 100, ...$ et pour $u_0 = 0.92$.
5. Soient les deux listes $X = [u_0, u_0, u_1, u_1, u_2, u_2, u_3, u_3, ..., u_{N-2}, u_{N-2}, u_{N-1}]$ et $Y = [0, u_1, u_1, u_2, u_2, u_3, u_3, ..., u_{N-1}, u_{N-1}]$. Construire X à partir de U à l'aide d'une boucle `for` du type (`for u in U:`) et déterminer Y à partir de X sans utiliser une boucle `for` ou `while`, puis tracer la ligne brisée formée des points suivants

$$(u_0, 0), (u_0, u_1), (u_1, u_1), (u_1, u_2), (u_2, u_2), (u_2, u_3), ...(u_{N-2}, u_{N-1}), (u_{N-1}, u_{N-1})$$



Pour cela vous allez écrire une fonction `graphe3(f,a,b,n)` permettant de tracer sur le même graphique la courbe de la fonction $f : [a, b] \to \mathbb{R}$, la première bissectrice $y = x$ pour une discrétisation à n points et la ligne en escalier en en escargot définie par les deux liste des abscisses et des ordonnées X et Y respectivement. Pour le cas où $f(x) = 4x(1 - x)$ le graphique doit ressembler à ceci :



6. Refaire l'étude graphique de la suite $(u_k)_{k\in\mathbb{N}}$ dans les deux cas suivants :
   — $\beta = 0.7$ et $u_0 = 0.01$.
   — $\beta = 0.82$ et $u_0 = 0.485$.

2.2 Morpion

Le morpion est un jeu de réflexion se pratiquant à deux joueurs (représentés par x et o) au tour par tour et dont le but est de créer le premier un alignement de trois de leurs symboles sur une grille.

Le jeu peut se terminer par une égalité si l'ensemble le tableau se lève sans qu'aucun joueur ne termine une telle ligne. Vous devez écrire une simulation du jeu où un utilisateur (x) joue contre l'ordinateur (o).

— nous pouvons supposer que l'utilisateur démarre toujours le jeu, en plaçant son marqueur x en premier sur le plateau.

— si c'est au tour de l'utilisateur de jouer, il / elle entrera 2 entiers indiquant la ligne et la colonne de la cellule choisie. La cellule supérieure gauche est spécifiée par 0 0 et la cellule inférieure droite par 2 2. Votre programme doit assurer que la cellule sélectionnée est vide et demande à l'utilisateur de ressaisir une valeur au cas où la sélection choisie n'est pas possible.
— si c'est au tour de l'ordinateur, une cellule parmi celles disponibles doit être choisie. Votre programme peut utilisez n'importe quelle stratégie pour choisir la cellule. Par exemple, une sélection aléatoire parmi les cellules vides peut être choisi. Vous pouvez également utiliser une stratégie plus élaborée.

Votre programme doit stocker :

— La liste des cellules vides restantes que l'on désigne par `empty_cells` sous forme de liste de couples (tuples) de coordonnées `(row,col)`.
— L'état du jeu sous forme de liste de 3 listes, chacune représentant une ligne de la grille. par exemple la grille suivante :

	o		x	
		o	o	
	x		x	

est représentée par la variable :
`g = [['o', '_', 'x'], ['_', 'o', 'o'], ['x', '_', 'x']]`
Les éléments seront `'x'`, `'o'` ou `'_'` , désignant respectivement une cellule de l'utilisateur, une cellule de l'ordinateur ou une cellule vide.

L'affichage sera effectué par une fonction à double boucle for :

```
# function d'affichage d'une liste de 3 listes
def print_grid(grid):
    for row in grid:
        for e in row:
            ..............
        print()
```

Voici quelques fonctions que vous voudrez peut-être écrire pour décomposer le programme en morceaux gérables.

— `getUserPick (empty_cells, grid)` obtiendra le choix de l'utilisateur, s'assurera qu'il est valide et mettra à jour la grille.

```
def get_user_pick(empty_cells, grid):
    global gUser
    while True:
        row = int(input("row= "))
        col = int(input("col= "))
        if (row,col) in empty_cells:
            grid[row][col]='x'
            empty_cells.remove((row,col))
            break
    print_grid(grid)
```

— `computerPick (empty_cells, grid)` choisira une cellule parmi celles disponibles, mettra à jour la grille et supprimer la cellule choisie de la liste des cellules vides.

```
def computer_pick(empty_cells, grid):
    while True:
        row = ..........................
        col  = ..........................
        if (row, col) in empty_cells:
            ..........................
            ..........................
            ..........................
    ..........................
```

— `checkWin (grille, joueur)` vérifie si le joueur (i.e., `x` ou `o`) a gagné.

```
def check_win(grid, player):
    # gridPlayer est la liste des coordonnées (row,col) des cases jouées par le joueur player (x ou o)
    gridPlayer = []
    for i in range(3):
        for j in range(3):
            ..........................
                ..........................
    #On teste s'il existe trois points alignés parmi les cases jouées par le player
    n=len(gridPlayer)
    for a in range(n):
        for b in range(a+1,n):
            for c in range(b+1,n):
                A = ......................
                B = ......................
                C = ......................
                # si det(vec(AB),vec(AC))=0 alors les trois points A,B et C sont alignés
                if ................................................ :
                    return True
    return False
```

Le programme principale ressemble à ceci :

```
def tictactoe():
    empty_cells = []
    for i in range(3):
        for j in range(3):
            empty_cells.append((i, j))
    g = [['_', '_', '_'], ['_', '_', '_'], ['_', '_', '_']]
    while len(empty_cells)>0:
        print("_"*50)
        print("User : ")
        get_user_pick(empty_cells, g)
        if check_win(g, "x"):
            print("User wins !")
            return
        print("_"*50)
        print("Computer : ")
        computer_pick(empty_cells, g)
        if check_win(g, "o"):
            print("Computer wins !")
            return

# start the game
```

```
22 tictactoe()
```

Exemple d'exécution :

```
--------------------------------------------------
User :
row= 1
col= 1
- - -
_ x _
- - -
--------------------------------------------------
Computer :
_ o _
_ x _
- - -
--------------------------------------------------
User :
row= 0
col= 0
x o _
_ x _
- - -
--------------------------------------------------
Computer :
x o _
_ x _
o _ _
--------------------------------------------------
User :
row= 2
col= 2
x o _
_ x _
o _ x
User wins !
```

3 Récursivité

3.1 PGCD

Écrire une fonction récursive `pgcd (a, b)` qui renvoie le plus grand commun diviseur de a et b, en utilisant le théorème d'Euclide $a \wedge b = b \wedge r$ où r est le reste de la division euclidienne de a par b. Lorsque $b = 0$, alors $a \wedge b = a$.

```
1 def pgcd(a,b):
2     if b==0:
3         return ......
4     else:
```

```
5           return .......
6
7   print(pgcd(102,68))
8
```

3.2 Inclusion

Écrire une fonction récursive `issubset (st1, st2)` qui prend deux arguments de chaînes de caractères et retourne `True` si tous les caractères de la première chaîne `st1` apparaissent quelque part dans `st2`, et `False` sinon.

Indication :

— Si `st1` est vide alors `issubset('', st2)=True`.

— Si le premier caractère de st1 est dans st2 on a l'équivalence :
`issubset(st1, st2)` $\Longleftrightarrow$ `issubset(st1[1:], st2)`

— Sinon `issubset(st1, st2)=False`

```
1   st1 = input("st1 = ")
2   st2 = input("st1 = ")
3
4   def issubset(st1,st2):
5       if len(st1)==0:
6           ..........................
7       else:
8           if st1[0] in st2:
9               ..........................
10          else:
11              ..........................
12
13
14  print(issubset(st1,st2))
```

3.3 Palindrome en version récursive

Écrire une fonction `is_palindrome2(st)`, version récursive d'une fonction déjà vu en TP1 qui renvoie un boolean indiquant si son argument est ou non un palindrome.

Indication :

— Si `st` est vide ou contient un seul caractère alors `is_palindrome2(st)=True`.

— Si les deux caractères aux extrémités de `st` sont égaux alors on a l'équivalence :
`is_palindrome2(st)` $\Longleftrightarrow$ `is_palindrome2(st[1:-1])`

— Sinon `is_palindrome2(st)=False`