TP noté : corrigé

PTSI Lycée Eiffel

18 décembre 2020

I. Pour s'échauffer : quelques programmes manipulant les listes.

Question 1 : Écrire une fonction Python **appartient** qui prend comme arguments une liste de nombres L et un nombre x, et qui renvoie True si le nombre x apparait au moins une fois comme élément de L, False sinon.

Une façon simple de faire est de parcourir tous les éléments de la liste via une boucle for : on teste pour chaque élément s'il est ou non égal à x, si c'est le cas, on peut arrêter le parcours de la liste et directement renvouer True, sinon on continue notre parcours.

```
def appartient(L,x) :
    for i in L :
        if i==x :
            return True
    return False
```

Question 2 : Écrire une fonction Python **compte** qui prend comme arguments une liste de nombres L et un nombre x, et qui compte le nombre d'éléments dans la liste L qui sont égaux à x (il existe une méthode **count** qui fait exactement celà mais qu'il est bien entendu interdit d'exploiter pour cette question).

Même principe qu'à la question précédente, sauf qu'il va falloir en plus créer une variable pour compter le nombre d'apparitions de la valeur x : on initialise cette variable à 0, et à chaque fois qu'on croise une valeur égale à x dans la liste, on l'augmente d'une unité. Ici, on aura un seul return à l'extérieur de la boucle puisqu'on doit du toute façon parcourir la liste en entier.

```
def compte(L,x) :
    c=0
    for i in L :
        if i==x :
            c=c+1
    return c
```

Question 3 : Écrire une fonction Python **remplace** qui prend comme arguments une liste de nombres L et deux nombres x et y, et qui remplace chaque élément de la liste initialement égal à x par un y.

Toujours le même principe, mais cette fois-ci, il est plus pratique que notre indice de boucle i prenne comme valeur la position à l'intérieur de la liste et non pas l'élément lui-même. Ainsi, quand on croise une valeur égale à x en position i, on se contente de la remplacer par un y à la même position. Notons pour les plus tatillons d'entre vous que le programme proposé ci-dessous va modifier le paramètre L, si on veut se contenter de renvoyer une liste modifiée tout en gardant la liste initiale L intacte, on ajoute une copie de liste en début de programme.

```
def remplace(L,x,y) :
    for i in range(len(L)) :
        if L[i]==x :
            L[i]=y
    return L
```

Question 4 : Écrire une fonction Python **successifs** qui prend comme arguments une liste de nombres L et un nombre x, et qui renvoie le nombre de fois où x apparait successivement au début de la liste L. La fonction doit donc renvoyer la valeur 0 si le premier élément de la liste n'est pas égal à x, et on fera attention à gérer correctement le cas où tous les éléments de la liste sont égaux à x.

Ici, c'est une boucle while qui va être nécessaire : on parcourt la liste, et tant que les éléments croisés sont égaux à x (et qu'on n'a pas atteint la fin de la liste), on continue, en ajoutant une unité à un compteur en passant (qui nous permettra en même temps de gérer la position dans la liste)

```
def successifs(L,x) :
    c=0
    n=len(L)
    while (c<n) and (L[c]==x) :
        c=c+1
    return c
```

Question 5 : Écrire une fonction Python **maxsuccessifs** qui prend comme arguments une liste de nombres L et un nombre x, et qui renvoie le nombre maximal d'apparitions successives de la valeur x dans la liste L.

C'est déjà nettement plus pénible. En gros, l'idée est de parcourir la liste et, à chaque fois qu'on croise un élément égal à x, de compter combien de x successifs on a à partir de celui-ci (on peut pour celà appliquer la fonction écrite pour la question 4 à une sous-liste du type $L[i:]$, même si ce n'est pas l'option choisie ci-dessous). En plus de celà, il faut calculer un maximum, donc créer une variable auxiliaire à laquelle on comparera à chaque fois qu'on croise une suite de valeurs égales à x la longueur de la « chaine de x » repérée. Si cette chaine est plus longue que toute les précédentes, on actualise la valeur de la variable auxiliaire.

```
def maxsuccessifs(L,x) :
    chaine=0
    n=len(L)
    for i in range(n) :
        if L[i]==x :
            j=i+1
            c=1
            while (j<n) and (L[j]==x) :
                j=j+1
                c=c+1
            if c>chaine :
                chaine=c
    return chaine
```

Question 6 : Écrire une fonction Python **memelongueur** qui prend comme arguments une liste de listes L, et qui renvoie True si tous les éléments de L sont des listes contenant le même nombre d'éléments, False sinon.

Cette question n'a rien à voir avec les précédentes. On parcourt la liste (chaque élément étant lui-même une liste), et on compare la longueur de l'élément avec celui du premier élément. S'il est différent, on arrête tout et on renvoie False. Sinon on continue.

```
def memelongueur(L) :
    a=len(L[0])
    for i in L[1 :] :
        if len(i) !=a :
            return False
    return True
```

II. Gestion de la création et de l'affichage de la grille.

Question 7 : Écrire une fonction Python **initialisegrille** qui prend comme arguments deux entiers n et p, et qui renvoie une liste de n listes contenant chacune p entiers initialisés à la valeur 0.

Plein de possibilités pour faire ça, soit en utilisant des boucles for (qu'on peut insérer directement dans les définitions de listes), soit en utilisant des commandes du type $[[0] * p] * n$

```
def initialisegrille(n,p) :
    l=[[0 for i in range(p)] for j in range(n)]
    return l
```

Question 8 : Écrire une fonction Python **affichegrille(g)** qui prend comme argument une liste de listes g et affiche la grille de jeu correspondante sous le format donné ci-dessus (le but principal est que les différentes lignes de la grille soient affichées les unes en-dessous des autres, et on essaiera de séparer les différents éléments de chaque ligne par des barres verticales. Les lignes horizontales encadrant la grille sont accessoires.

On peut par exemple faire une double boucle for, pour afficher chaque ligne à l'aide d'un print séparé. La ligne complète est obtenue dans le programme ci-dessous sous forme de chaîne de caractères, en insérant une barre verticale entre chaque valeur prise dans la ligne. Il n'y a rien derrière le return, c'est ici voulu puisque le programme a déjà affiché la grille.

```
def affichegrille(g) :
    for i in range(len(g)) :
        s='|'
        for j in range(len(g[i])) :
            s=s+str(g[i][j])+'|'
        print(s)
    return
```

III. Programmation des fonctions indispensables au fonctionnement du jeu.

Question 9 : Écrire une fonction Python **couppossible(g,i)** qui prend comme arguments une grille de jeu g (liste de listes de nombres entiers) et un entier i représentant le numéro de colonne où le joueur souhaite placer un pion, et qui renvoie True si le coup est possible, et False sinon (donc si le numéro de colonne est incohérent avec la taille de la grille, ou si la colonne i est déjà remplie). On supposera que le joueur numérote les colonnes à partir de 1 et non à partir de 0 comme le fait Python.

Il suffit de regarder si l'élément ligne 0 colonne $i-1$ (en décalant les indices pour être cohérent avec les habitudes de Python) est égal à 0 (le coup est alors possible) ou non (le coup est alors impossible, la colonne est pleine). Version minimaliste :

```
def couppossible(g,i) :
    return g[0][i-1]==0
```

Question 10 : Écrire des fonctions Python **coupj1** et **coupj2** qui prennent comme arguments une grille de jeu g et un entier i et qui modifient (quand le coup est possible) la grille en plaçant un jeton du joueur 1 ou du joueur 2 dans la colonne i. Le return de la fonction doit renvoyer la grille de jeu, modifiée ou non.

On vérifie d'abord si le coup est possible, puis on « descend » dans la colonne numéro $i-1$ jusqu'à croiser un emplacement déjà plein (donc une valeur autre que 0). À ce moment-là on place la valeur 1 ou 2 (selon le joueur) dans la case juste au-dessus. Attention quand même à bien arrêter le programme quand on atteint le bas de la grille (si la colonne est vide, il faut que le 1 ou le 2 apparaisse dans la dernière ligne). Les deux programmes sont identiques à un caractère près.

```
def coupj1(g,i) :
    if not couppossible(g,i) :
        return g
    c=0
    while c<len(g)-1 and g[c+1][i-1]==0 :
        c+=1
    g[c][i-1]=1
    return g

def coupj2(g,i) :
    if not couppossible(g,i) :
        return g
    c=0
    while c<len(g)-1 and g[c+1][i-1]==0 :
        c+=1
    g[c][i-1]=2
    return g
```

Question 11 : Écrire une fonction Python **verifiegain** qui prend comme argument une grille de jeu g et qui vérifie si l'un des deux joueurs a quatre pions alignés (cette fonction étant pénible à programmer, on pourra se contenter d'une version où on ne vérifie que les alignements horizontaux ou verticaux).

C'est effectivement fort pénible : pour chacun des deux joueurs, il faut parcourir les lignes et les colonnes (les lignes sont simplement les éléments de la grille, mais pour les colonnes il faut les recréer à la main), puis calculer le maximum des alignements (en utilisant pas exemple la fonction maxsuccessifs programmée plus haut). Tant qu'à faire, on peut inclure un petit message félicitant le joueur qui a gagné le cas échéant.

```
def verifiegain(g) :
    for joueur in range(1,3) :
        m=0
        for i in range(len(g)) :
            a=maxsuccessifs(g[i],joueur)
            if a>m :
                m=a
```

```
        for j in range(len(g[0])) :
            liste=[g[i][j] for i in range(len(g))]
            a=maxsuccessifs(liste,joueur)
            if a>m :
                m=a
        if m>=4 :
            print('Le joueur ',joueur,' a gagne la partie !')
            return True
    return False
```

Question 12 : Écrire une fonction Python **puissance4** qui prend comme arguments deux entiers n et p, et qui permet à deux joueurs d'effectuer une partie de Puissance 4™ sur une grille à n lignes et p colonnes. On essaiera bien sûr de réutiliser les questions précédentes, et de proposer une interface de jeu compréhensible (un message du type « Joueur 2, à vous de choisir une colonne » s'affichera pour indiquer à chaque joueur que c'est à son tour de jouer). Toute version, même inaboutie, sera récompensée.

Le programme qui suit n'est pas parfait : il compte le nombre de coups joués pour s'arrêter quand la grille est pleine sans qu'un des joueurs ait réussi à aligner quatre pions, mais il ne tient pas compte des éventuels coups impossibles tentés par les joueurs. Le reste devrait être assez clair.

```
def puissance4(n,p) :
    g=initialisegrille(n,p)
    affichegrille(g)
    coups=0
    joueur=1
    while coups<n*p :
        print('Joueur ',joueur,', choisissez une colonne')
        i=input()
        if joueur==1 :
            g=coupj1(g,i)
            joueur=2
        else :
            g=coupj2(g,i)
            joueur=1
        coups+=1
        affichegrille(g)
        verifiegain(g)
        if verifiegain(g) :
            return
    return('Match nul !')
```

IV. Complément : programmation d'une IA basique.

Question 13 : Écrire une fonction Python **puissordibete** qui prend comme arguments deux entiers n et p et qui permette de joueur une partie contre un ordinateur idiot qui choisit pour chacun de ses coups une colonne de la grille de façon totalement aléatoire. On rappelle que la fonction **randint(a,b)** issue du module **random** permet d'obtenir un entier aléatoire compris entre les valeurs a et b, incluses toutes les deux.

On importe le module random puis on adapte le programme précédent (on suppose ici que l'ordinateur jouera toujours le rôle du joueur 2) :

```
from random import randint
def puissordibete(n,p) :
    g=initialisegrille(n,p)
    affichegrille(g)
    coups=0
    while coups<n*p :
        print('Joueur humain, choisissez une colonne')
        i=input()
        g=coupj1(g,i)
        r=randint(1,p)
        g=coupj2(g,r)
        coups+=2
        affichegrille(g)
        verifiegain(g)
        if verifiegain(g) :
            return
    return('Match nul !')
```

Question 14 : Écrire une fonction Python **puissordi** qui prend comme arguments deux entiers n et p et qui permette de jouer une partie contre un ordinateur « intelligent » qui utilise la tactique suivante : à chacun de ses coups, il choisit de jouer dans la colonne lui permettant de créer le plus long alignement de ses pions possibles (ainsi, s'il a au moment de jouer au maximum deux pions alignés dans la grille, il choisira une colonne lui permettant d'aligner un troisième de ses pions si possible ; si plusieurs colonnes conviennent, il choisit la première disponible, ou il en choisit une au hasard, au choix).

Dans la proposition qui suit, la fonction meilleuralign calcule le plus long alignement de l'ordinateur (joueur 2) dans la grille g (quasiment le même programme que verifigain), puis meilleurcoup calcule le meilleur coup possible pour l'ordi (j'ai utilisé un module spécifique pour faire des copies de la grille et tester les différents coups disponibles sans modifier la grille de jeu). Enfin, le programme principal ressemble au premier programme de jeu contre l'ordinateur, mais en remplaçant le tirage aléatoire par le meilleur coup possible. Notons en passant que ce programme est vraiment complètement pourri, car en plus de ceux signalés dans l'énoncé, il a le défaut suivant : imaginez que l'ordinateur commence à aligner trois pions verticalement et qu'au quatrième coup vous lui posiez un pion au-dessus de cet alignement pour l'empêcher de le terminer. Pendant toute la suite de la partie, l'ordinateur va continuer à considérer que son meilleur alignement est un alignement de trois pions (même s'il ne sert à rien puisqu'on ne pourra jamais le compléter) et ne considèrera donc un coup comme bon que s'il lui permet d'aligner quatre pions, ce qui est passablement idiot. Il faudrait en faire réussir à calculer le « meilleur alignement complétable », ce qui pose là encore des difficultés de programmation que je préfère ne pas aborder.

```
def meilleuralign(g) :
    m=0
    for i in range(len(g)) :
        a=maxsuccessifs(g[i],2)
        if a>m :
            m=a
    for j in range(len(g[0])) :
        liste=[g[i][j] for i in range(len(g))]
        a=maxsuccessifs(liste,2)
```

```
            if a>m :
                m=a
    return m


from copy import deepcopy


def meilleurcoup(g) :
    z=1
    l=meilleuralign(g)
    for i in range(1,len(g[0])+1) :
        grilleprovisoire=deepcopy(g)
        grilleprovisoire=coupj2(grilleprovisoire,i)
        l2=meilleuralign(grilleprovisoire)
        if l2>l :
            z=i
    return z


def puissordi(n,p) :
    g=initialisegrille(n,p)
    affichegrille(g)
    coups=0
    while coups<n*p :
        print('Joueur humain, choisissez une colonne')
        i=input()
        g=coupj1(g,i)
        o=meilleurcoup(g)
        g=coupj2(g,o)
        coups+=2
        affichegrille(g)
        verifiegain(g)
        if verifiegain(g) :
            return
    return('Match nul !')
```