Travaux pratiques n°3

Exercices sur des tableaux de nombres
Awalé

F. Cuvellier, J. Tanoh

M.P.S.I. 1, 2017–2018

1 Exercices sur des tableaux de nombres

Exercice 1 (Commentaire de code).
On définit la fonction `max2` dont le paramètre est un tableau `L` de nombres entiers : elle retourne la plus grande des sommes de deux éléments consécutifs de `L` si `L` possède au moins deux éléments, l'unique élément de `L` sinon.

```
def max2(L):
    n = len(L)
    if n == 1:
        return L[0]
    max = L[0] + L[1]
    for i in range(n-2):
        m = L[1+i] + L[2+i]
        if m > max:
            max = m
    return max
```

Commenter les lignes de ce code Python.

Exercice 2 (Construction de code).
On souhaite définir une fonction `MAX(L, n)` à deux paramètres, un tableau `L` de nombres entiers et un nombre entier `n`, plus grand que 2. Elle retourne la plus grande des sommes de `n` éléments consécutifs de `L` si `L` possède au moins `n` éléments, la somme de ses éléments sinon.

1. Proposer un algorithme en pseudo-code pour cette fonction.
2. Implémenter cet algorithme en Python. Le code devra être commenté.

Pour créer des tableaux à tester, on pourra s'inspirer des instructions suivantes, qui permettent d'obtenir un tableau de vingt nombres entiers tirés au sort entre 3 et 12 (inclus).

```
from random import randint
L = []
for i in range(20):
    L.append(randint(3, 12))
```

ou

```
from random import randint
L = [ randint(3, 12) for i in range(20) ]
```

Cela donne (par exemple)

```
>>> L
[5, 12, 6, 4, 4, 3, 7, 5, 12, 8, 10, 12, 7, 9, 12, 8, 3, 12, 7, 8]
```

2 Awalé

On se propose de modéliser l'awalé, un jeu de société d'origine africaine. Il se joue à deux. Son matériel est constitué de douze cases disposées sur une table de jeu suivant deux rangées et de quarante-huit graines. On convient que les cases se suivent selon le sens trigonométrique.
Les deux joueurs se font face, séparés par la table de jeu. Chacun d'eux se tient devant une rangée de six cases, qui constitue son camp. Au début, les quarante-huit graines sont réparties quatre par quatre dans les douze cases. On tire au sort qui commence. Chaque joueur, à son tour, prend les graines d'une seule de ses cases et les met dans les cases suivantes (sauf dans celle qu'on vient de vider), à raison d'une graine par case, puis, le cas échéant, vide des cases de son adversaire en respectant des règles énoncées plus bas. C'est ensuite le tour de l'autre joueur.
Le jeu consiste à prendre le plus grand nombre de graines.

Exemple de coup

Joueur B

12	*11*	*10*	*9*	*8*	*7*
0	3	2	1	0	2
0	2	7	0	2	1
1	*2*	*3*	*4*	*5*	*6*

Joueur A

Joueur B

12	*11*	*10*	*9*	*8*	*7*
0	3	3	2	1	3
0	2	0	1	3	2
1	*2*	*3*	*4*	*5*	*6*

Joueur A

Joueur B

12	*11*	*10*	*9*	*8*	*7*
0	3	0	0	1	3
0	2	0	1	3	2
1	*2*	*3*	*4*	*5*	*6*

Joueur A

Le joueur A a choisi de vider la case numéro 3 ; dans un premier temps, il a distribué les sept graines dans les cases suivantes, ce qui donne lieu à une nouvelle situation, provisoire.

Parce que la dernière case atteinte est chez l'adversaire et qu'elle contient deux ou trois graines, c'est une situation de prise : le joueur A retire les graines de cette case, et considère la case précédente ; si elle se trouve chez son adversaire et si elle contient aussi deux ou trois graines, le joueur A retire les graines, et ainsi de suite, jusqu'à ce qu'il atteigne son camp ou que la nouvelle case ne contienne pas deux ou trois graines.
Ici, le joueur A gagne cinq graines, et c'est le tour de l'autre joueur.

Règles :

- La distribution des graines se fait dans le sens trigonométrique.
- Si le nombre de graines de la case qu'on vide permet de faire plus d'un tour, on ne remet pas de graine dans la case qu'on vient de vider.
- Le coup ne doit pas empêcher le coup suivant de l'adversaire, comme cela se produirait si sa rangée était vide. Il faut que l'adversaire ait au moins une graine à jouer.
- Si lors du coup, on est amené à prendre toutes les graines de l'adversaire, on joue le coup mais sans rien prendre.
- Si un joueur n'a aucun coup possible, les graines restantes du plateau lui reviennent.
- La partie se termine dès qu'un joueur a capturé au moins vingt-cinq graines ou qu'il ne reste plus que six graines sur le plateau.

Exercice 3 (Manipulation de chaînes de caractères).
Objectif : représenter l'état de la table
L'état du plateau est représenté par une variable `Etat`, de type tableau, qui contient le nombre de graines dans les douze cases. Au début de la partie, `Etat = [4,4,4,4,4,4,4,4,4,4,4,4]`.
Pour représenter l'état du plateau à l'image de ce que voit le joueur A, on a réalisé une fonction `representation(Etat)` qui affiche à l'écran la chaîne de caractères suivante.

```
*************************************
*  4  *  4  *  4  *  4  *  4  *  4  *
*************************************
*  4  *  4  *  4  *  4  *  4  *  4  *
*************************************
```

Voici une partie du code de cette fonction :

```
def representation(Etat):
    bord = 37 * '*'
    ligne = ''
    # debut des lignes a modifier
    for i in range(6):
        ligne = ligne + '* ' + 4 * ' '
    # fin des lignes a modifier
    ligne = ligne + '*'
    print(bord)
    print(ligne)
```

```
    ligne = ''
    for i in range(6):
        st = str(Etat[i])
        if len(st) == 1: st = ' ' + st
        ligne = ligne + '* ' + st + 2 * ' '
    ligne = ligne + '*'
    print(bord)
    print(ligne)
    print(bord)
```

L'appel de cette fonction donne :

```
>>> E1 = [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
>>> representation(E1)
************************************
*     *     *     *     *     *     *
************************************
*  4  *  4  *  4  *  4  *  4  *  4  *
************************************
```

Compléter le code à partir de celui qui est proposé ici.

Exercice 4 (Manipulation de tableaux).
Objectif : jouer un coup

1. À partir d'un état `E`, on souhaite jouer la case `n` si elle n'est pas vide et déterminer le nouvel état `newE` obtenu avant d'étudier les prises et de vérifier les règles.

   (a) Proposer un algorithme qui, à partir de l'état `E` et du nombre `n`, retourne le nouvel état.

   (b) Écrire en Python la fonction `joue(n, E)` qui retourne le nouvel état.

2. Améliorer la fonction `joue(n, E)` pour qu'elle retourne le *tuple* `NouvE, valide, prise`, où `valide` est la variable booléenne validant le coup et `prise` la prise associée à ce coup si `valide` est vrai.