Algorithmique/Python

Master *"Technologie et Handicap"* : Intensifs d'Automne

Examen du 6 octobre 2012, 17h
Durée 2h, documents (poly et notes de cours) autorisés

Correction

1 Fonctions et tableaux

1. On souhaite écrire une fonction qui renvoie un nombre donné arrondi à un nombre de décimales déterminé.
   (a) Quel(s) est(/sont) l'(/les) argument(s) de cette fonction ?
   (b) Que renvoie cette fonction ?
   (c) Écrire la fonction. On pourra utiliser la fonction *partieEntiere* :
   `partieEntiere(var v réel var valeur) : entier`
   (d) Implémenter cette fonction en Python
   Rappel : la fonction *partieEntiere* en python est : `def int(v)`

Solutions

(a) Il y a deux arguments, le nombre (réel) qu'on souhaite arrondir et le nombre de décimales (entier), tous deux passés par valeurs (on pourrait aussi les passer par variables mais cela n'a pas d'intérêt particulier ici).

(b) Elle renvoie un réel.

(c)
```
fonction arrondit(var v : réel par valeur, n : entier par valeur) : réel
debut
  v ← v*(10^n) // 10 puissance n
  v ← partieEntiere(v)
  v ← v/(10^n)
  retourner v
fin
```

Plus efficace (calculer une puissance est long et de toutes façons comprend une boucle) :

```
fonction arrondit2(var v : réel par valeur, n : entier par valeur) : réel
debut
  i ← 0
  tantque i < n faire  v ← v * 10  fait
  v=partieEntiere(v)
  tantque i < n faire  v ← v / 10  fait
  retourner v
fin
```

(d) Implémentation en Python :

```
def arrondit(v, n):
  if n==0: return int(v)
  for i in range(n): v=v*10
  v=int(v)*1.0
  for i in range(n): v=v/10
  return v
```

2. On souhaite maintenant écrire une fonction permettant d'arrondir tous les éléments d'un tableau *(en utilisant la fonction précédente, bien sûr)*.

   (a) Quel(s) est(/sont) l'(/les) argument(s) de cette fonction ? (en particulier comment cet(/ces) argument(s) est(/sont)-il(s) passé(s)).

   (b) Que renvoie cette fonction ?

   (c) Écrire la fonction.

   (d) Implémenter cette fonction en Python

   Solutions

   (a) Il faut passer en argument le tableau (par variables), ainsi que sa longueur (par valeurs).

   (b) Elle n'a pas besoin de renvoyer quoique ce soit, car le tableau lui-même sera modifié, mais elle peut renvoyer le tableau.

   (c)
```
fonction arronditTableau(var t: réels[] par reference,
                                n : entier par valeur)
var i:entier
debut
pour i allantde 0 a taille(n)-1 faire
  tab[i] ← arrondit(tab[i],n)
fait
fin
```

   (d) Implémentation en Python :

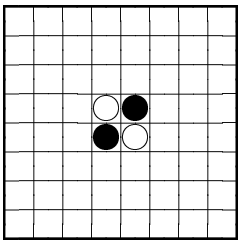
```
def arronditTableau(tab,n):
  for i in range(len(tab)):
    tab[i]=arrondit(tab[i],n)
```

2 Réversi

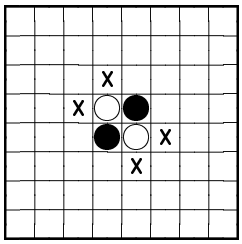
Le jeu de *Réversi* est inspiré du jeu chinois de Go (le plus ancien jeu de stratégie connu). Deux joueurs s'affrontent avec des pierres blanches et noires sur un plateau de 8 x 8 cases.

Le principe du jeu est simple. Un joueur doit poser sa pierre à l'extrémité d'une ligne de pierres adverses et dont l'autre extrémité comporte déjà un de ses propres pions. Les pions ainsi encadrés sont alors pris à l'adversaire (et remplacés par des pierres de la couleur de celui qui vient de prendre). S'il ne peut rien prendre, le joueur doit passer son tour. Dans le jeu de *Réversi* les lignes peuvent être horizontales, verticales ou diagonales. Pour simplifier **on ne considérera ici que les lignes horizontales et verticales**.

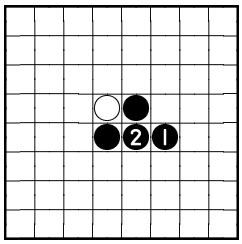
Voici la position de départ et les mouvements possibles.



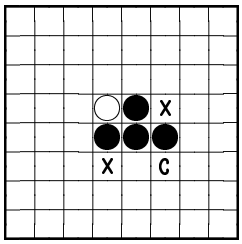
Position de départ



Mouvements possibles pour les noirs



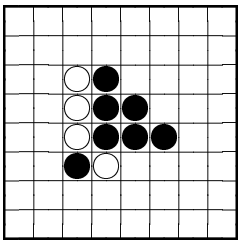
Les noirs jouent 1, la pièce 2 est prise



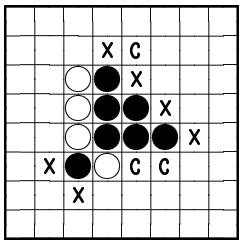
Mouvements possibles pour les blancs

NB : On a marqué d'un C les mouvements possibles dans le jeu *Réversi*, mais qu'on ne prendra pas en compte pour simplifier, comme indiqué plus haut (lignes en diagonales).

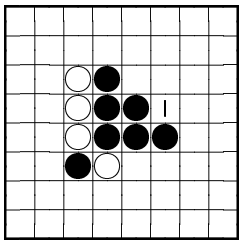
Dans cet exemple un peu plus compliqué c'est au blancs de jouer :



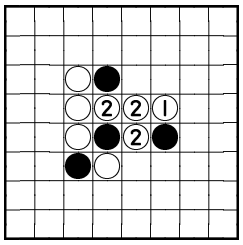
Situation initiale, les noirs viennent de jouer



Mouvements possibles pour les blancs



Les blancs jouent 1



Les pièces 2 sont prises

On représente le plateau de jeu par un tableau de caractères à deux dimensions, pour chaque case on utilisera le caractère *espace* si la case est vide, le caractère 'b' si elle contient un pion blanc, et le caractère 'n' si elle contient un pion noir.

```
var reversi: car[8,8]
```

Questions :

1. Écrire une fonction qui initialise le plateau de jeu, selon la figure « Position de départ ».
   Solution

```
fonction initPlateau(var reversi: car[,] par variables)
var i,j: entiers
debut
  pour i allantde 0 a 7 faire
    pour j allantde 0 a 7 faire
      reversi[i,j] ← ' '
    fait
  fait
  reversi[3,3] ← 'b'
  reversi[3,4] ← 'n'
  reversi[4,3] ← 'n'
  reversi[4,4] ← 'b'
fin
```

Nombreux sont ceux qui ont traité les 4 cas particuliers dans la boucle, avec des tests :

```
...
debut
  pour i allantde 0 a 7 faire
    pour j allantde 0 a 7 faire
      si i=3 et j=3 ou i=4 et j=4 alors
        reversi[i,j] ← 'b'
      sinon
        si i=3 et j=4 ou i=3 et j=4 alors
          reversi[i,j] ← 'n'
        sinon
          reversi[i,j] ← ' '
        finsi
      finsi
    fait
  fait
fin
```

Outre que cet algo est beaucoup plus long à saisir ($\approx +25\%$), elle est ***beaucoup moins efficace*** *(pour gagner 4 affectations, on fait grosso modo* $64 \times 8 = 512$ *évaluations de tests (*`i=...`*,* `j=...`*).*

2. Écrire une fonction qui renvoie vrai si une case du plateau, dont les coordonnées sont données en arguments, est libre, et faux sinon.
   Solution

```
fonction libre(var reversi: car[,] par variables,
                   l,c: entiers par valeurs)
debut
  return reversi[l,c] = ' '
fin
```

3. Écrire une fonction qui compte le nombre de cases occupées par une des couleurs donnée en argument.
   Solution

```
fonction compte(var reversi: car[,] par variables,
                    p: car par valeurs)
debut
  cc ← 0
  pour i allantde 0 a 7 faire
    pour j allantde 0 a 7 faire
      si reversi[i,j] = c alors
        cc ← cc + 1
      finsi
    fait
  fait
  retourner cc
fin
```

4. Écrire une fonction qui compte dit qui a le plus de pions sur le plateau de jeu (et donc qui a gagné si le jeu est terminé).
   Solution

```
fonction enTete(var reversi: car[,] par variables)
  // on retournera 'x' en cas d'ex-aequo
  si compte(reversi,'b') > compte(reversi,'n') alors
    retourner 'b'
  sinon
    si compte(reversi,'b') < compte(reversi,'n') alors
      retourner 'n'
    sinon
      retourner 'x'
    finsi
  finsi
fin
```

5. Écrire une fonction qui vérifie si un joueur peut jouer sur une case dont les coordonnées sont données en arguments.
   Solution
   Le principe est de parcourir les lignes dans les 4 directions pour détecter s'il y a des pièces à prendre. Pour chaque direction (il faut le faire 4 fois, il n'y a malheureusement pas de moyen vraiment efficace de généraliser) on va avancer tant qu'on trouve des pièces de l'autre couleur et si on trouve une pièce de sa couleur au bout, cela signifie qu'on peut jouer là.

```
fonction possible(var reversi: car[,] par variables,
                      l, c: entiers par valeurs, p: car par valeurs)
var i: entier
debut
  i ← c+1
  tantque i<7 et reversi[l,i]≠' ' et reversi[l,i]≠p faire
    i ← i+1
  fait
  si i>c+1 et i<8 et reversi[l,i]==p alors
    retourner Vrai
  finsi

  i ← c-1
  tantque i>0 et reversi[l,i]≠' ' et reversi[l,i]≠p faire
    i ← i-1
  fait
  si i<c-1 et i≥0 et reversi[l,i]==p alors
    retourner Vrai
  finsi

  i ← l+1
  tantque i<7 et reversi[i,c]≠' ' et reversi[i,c]≠p faire
    i ← i+1
  fait
  si i>l+1 et i<8 et reversi[i,c]==p alors
    retourner Vrai
  finsi

  i ← l-1
  tantque i>0 et reversi[i,c]≠' ' et reversi[i,c]≠p faire
    i ← i-1
  fait
  si i<l-1 et i≥0 et reversi[i,c]==p alors
    retourner Vrai
  finsi

  retourner Faux
fin
```

Question bonus

6. Écrire une fonction qui *« joue »* un coup, c'est à dire qui place le pion dans le plateau de jeu et change de couleur les pierres prises.
   Solution
   On reprend l'algorithme précédent. Lorsqu'une ligne de prise a été trouvée, on revient en arrière en changeant les pierres. En premier lieu on verifiera que le coup est possible.

```
fonction jouer(var reversi: car[,] par variables,
                    l, c: entiers par valeurs, p: car par valeurs)
var i: entier
```

```
debut
  si possible(reversi, l, c, p) alors
    i ← c+1
    tantque i<7 et reversi[l,i]≠' ' et reversi[l,i]≠p faire
      i ← i+1
    fait
    si i>c+1 et i<8 et reversi[l,i]==p alors
      i ← i-1
      tantque i>c faire
        reversi[l][i] ← p
        i ← i-1
      fait
    finsi

    i ← c-1
    tantque i>0 et reversi[l,i]≠' ' et reversi[l,i]≠p faire
      i ← i-1
    fait
    si i<c-1 et i≥0 et reversi[l,i]==p alors
      retourner Vrai
      i  ← i+1
      tantque i<c faire
        reversi[l][i] ← p
        i ← i+1
      fait
    finsi

    i ← l+1
    tantque i<7 et reversi[i,c]≠' ' et reversi[i,c]≠p faire
      i ← i+1
    fait
    si i>l+1 et i<8 et reversi[i,c]==p alors
      retourner Vrai
      i ← i-1
      tantque i>l faire
        reversi[i][c] ← p
        i ← i-1
      fait
    finsi

    i ← l-1
    tantque i>0 et reversi[i,c]≠' ' et reversi[i,c]≠p faire
      i ← i-1
    fait
    si i<l-1 et i≥0 et reversi[i,c]==p alors
      retourner Vrai
      i ← i+1
      tantque i<l faire
        reversi[i][c] ← p
        i ← i+1
      fait
    finsi
  finsi
fin
```

Attention : tous les arguments n'ont pas été donnés dans les énoncés.