



Samedi 04 Avril 2020

OPTION : SCIENCES DU NUMERIQUE

MP / PC / PSI / PT / TSI

Durée : 2 Heures

Condition(s) particulière(s)

Calculatrice interdite

Indiquer votre code candidat sur le QCM et l'insérer dans votre copie d'examen

Consignes Python

Tout code doit être écrit dans le langage Python.

- Tout code Python non indenté ne sera pas corrigé.
- Tout ce dont vous avez besoin (fonctions, méthodes) est indiqué ci-dessous.
- Vous pouvez écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).
Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.

Fonctions et méthodes autorisées

Vous pouvez utiliser la fonction range :

```
>>> for i in range(10):
...:     print(i, end=' ')

0 1 2 3 4 5 6 7 8 9

>>> for i in range(5, 10):
...:     print(i, end=' ')

5 6 7 8 9
```

Sur les listes, vous pouvez utiliser la méthode append et la fonction len :

```
>>> help ( list . append )
Help on method_descriptor : append (...)
L.append ( object ) -> None -- append object to end of L

>>> help (len)
Help on built-in function len in module builtins : len (...)
len ( object )
Return the number of items of a sequence or collection .
```

Les matrices sont représentées par des listes de listes comme dans l'exemple ci-dessous :

```
>>> M1 = [[1, 10, 3, 0, 3, 10, 1],
          [1, 0, 1, 8, 1, 0, 1],
          [10, 9, 4, 1, 4, 9, 10],
          [10, 3, 7, 1, 7, 3, 10],
          [7, 8, 5, 1, 5, 8, 7]]
```

Introduction

Le sujet de l'option informatique contient quatre exercices allant de l'écriture de fonctions à l'analyse de code. Le langage utilisé, dans tous les cas, est Python. Le premier exercice parle d'arbres binaires, les exercices 2 et 3 traitent de matrices, quant au 4ème il s'agit de coder le principe de Kaprekar à l'aide de sous-fonctions et de listes. Vous trouverez en annexe les différentes fonctions et structures de données autorisées.

A) Arbres binaires : occurrences ... (2 points)

Un arbre binaire est une structure par nature récursive. Il peut être soit vide (\emptyset), soit la composée d'un noeud racine et de deux sous-arbres ($\langle o, G, D \rangle$) où o est le noeud racine, et G et D sont deux arbres binaires disjoints (respectivement sous-arbre gauche et sous-arbre droit). Un nœud peut donc être le père de 0, 1 ou 2 nœuds fils (respectivement gauche et droit) suivant que l'arbre dont il est racine soit la composée de 0, 1 ou deux sous-arbres non vide.

La taille d'un arbre binaire est définie comme étant le nombre de nœuds qui le composent et sa hauteur comme étant le maximum des hauteurs de ses nœuds : la hauteur d'un nœud étant égale à 0 s'il est la racine de l'arbre et à la hauteur de son père plus 1 dans tous les autres cas.

Notons qu'il est possible de représenter un arbre binaire sous forme d'occurrences. Dans ce cas, si un nœud a pour occurrence le mot μ , alors son fils gauche aura pour occurrence le mot $\mu 0$ et son fils droit le mot $\mu 1$.

Nous nous proposons donc de représenter un arbre binaire à l'aide d'une chaîne de caractères constituée des diverses occurrences de cet arbre, séparées les unes des autres par une virgule. Nous représenterons la racine par la chaîne vide.

Soit l'arbre B défini sous forme d'occurrences et représenté à l'aide d'une chaîne de caractères de la manière suivante :

$$B = "", 0, 1, 00, 01, 10, 11, 001, 101, 1010, 1011"$$

Comment sans représenter l'arbre B peut-on déterminer :

- 1) $T(B)$ la taille de B ?
- 2) $H(B)$ la hauteur de B ?
- 3) Le fait qu'un nœud de B soit une feuille (pas de fils) ?
- 4) $LC(B)$ la longueur de cheminement de B (la somme des hauteurs des nœuds de l'arbre) ?

B) Matrices : transposée ... (3 points)

La matrice transposée d'une matrice **A** est la matrice **A^T** obtenue en échangeant les lignes et les colonnes de **A**.

$$A = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \text{ alors } A^T = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

Écrire la fonction **transpose(M)** qui construit et retourne la matrice transposée de la matrice **M** non vide.

C) Matrices : symétrie ... (5 points)

1	1	10	10	7
10	0	9	3	8
3	1	4	7	5
0	8	1	1	1
3	1	4	7	5
10	0	9	3	8
1	1	10	10	7

Mat1

1	1	10	10	7
10	0	9	3	8
3	1	4	7	5
3	1	4	7	5
10	0	9	3	8
1	1	10	10	7

Mat2

1	1	10	10	7
10	0	9	3	8
3	1	4	7	5
0	8	1	1	1
3	1	4	7	5
10	0	9	3	8
1	1	10	10	6

Mat3

Écrire la fonction **symetric(M)** qui vérifie si une matrice **M** est symétrique selon l'axe horizontal (symétrie verticale).

Exemples d'applications sur les matrices Mat1, Mat2 et Mat3 :

```
>>> symetric(Mat1)
True

>>> symetric(Mat2)
True

>>> symetric(Mat3)
False
```