Algorithmique
Master MEEF, préparation CAPES Maths
Feuille d'exercices 2 (sur machines), Octobre 2018

1 Algorithmes itératifs

Exercice 2.1 1. Écrire une fonction python non récursive `somme_cubes` qui prend en argument un entier n et qui renvoie $\sum_{k=0}^{n} k^3$ si $n \geqslant 0$, et 0 si $n < 0$.

2. Écrire une version récursive de cette fonction.
3. Écrire une fonction `somme_entiers` qui prend en argument un entier n et qui renvoie $\sum_{k=0}^{n} k$ si $n \geqslant 0$, et 0 si $n < 0$.
4. Écrire une fonction `verifier_identite` qui prend en argument un entier N et qui vérifie que pour tout entier n entre 0 et N, on a

$$\sum_{k=0}^{n} k^3 = \Big(\sum_{k=0}^{n} k\Big)^2.$$

Exercice 2.2 On représente un point P dans le plan par un couple $(x\,,y)$, où x est l'abscisse de P et où y est son ordonnée. En Python, si `P` est le couple `(x,y)`, on accède à la valeur de `x` par `P[0]` et à celle de `y` par `P[1]`.

1. Écrire une fonction `distance(P1, P2)` qui prend en arguments deux points P_1 et P_2 et qui retourne la distance entre ces points. Pour utiliser la fonction *"racine carrée"*, insérer la ligne `from math import *` en début de fichier. La fonction racine carrée est alors accessible par `sqrt`.
2. Écrire une fonction `distance_minimale(L)` prenant en argument une liste `L` de points du plan et qui retourne `None` si cette liste a moins de deux éléments, et deux points P et Q de la liste situés à des positions distinctes dans `L` et dont la distance est minimale parmi toutes les distances entre points situés à des positions distinctes de `L`.

   Rappels. On accède à la longueur d'une liste `L` par `len(L)`. Connaissant une position `i` dans la liste, entre 0 et `len(L)-1`, on accède à l'élément à cette position par `L[i]`. Enfin, on peut parcourir toutes les positions grâce à une boucle `for i in range(len(L))`.
3. Évaluer la complexité de votre fonction en fonction de la longueur de la liste.

2 Algorithmes récursifs

Exercice 2.3 La suite $(u_n)_{n\geqslant 0}$ est définie par les égalités suivantes :

$$u_0 = 1, \quad u_1 = 42 \quad \text{et pour } n \geqslant 0,\ u_{n+2} = u_{n+1} - 2u_n.$$

1. Écrivez une fonction récursive traduisant naïvement cette définition mathématique.
2. Évaluer le nombre d'appels récursifs de votre fonction.
3. En utilisant l'exponentiation rapide et le fait que $\binom{u_{n+2}}{u_{n+1}} = M \cdot \binom{u_{n+1}}{u_n}$ pour une matrice M bien choisie, écrivez une fonction calculant u_n réalisant $O(\log_2(n))$ opérations arithmétiques, dans le cas le pire.
4. Testez vos deux fonctions pour n valant 1, 10, 30, 40, 100, 100000000.

Dans les exercices 4, 5 et 6, on utilise le module `turtle` de python. Ce module permet de dessiner des segments de droites, en déplaçant un stylo (appelé "tortue" en raison de sa rapidité de déplacement) sur une fenêtre. Pour pouvoir utiliser ce module, vous devez écrire :

```
from turtle import *
```

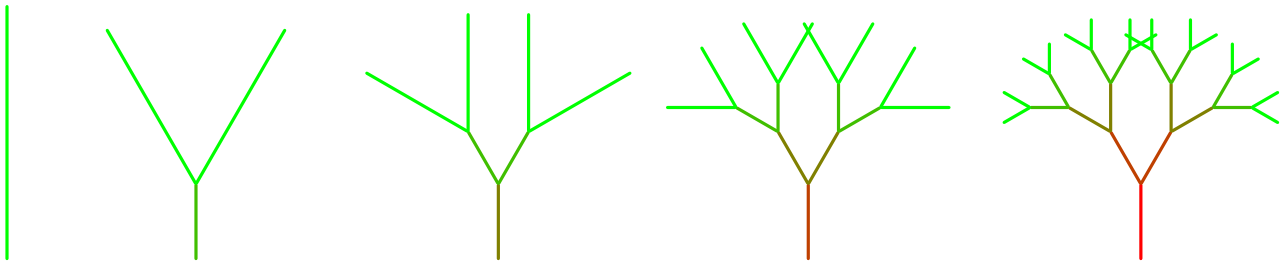
ou, ce qui est plus propre mais demande d'ajouter "`turtle.`" devant chaque nom de fonction :

```
import turtle
```

Plusieurs fonctions deviennent alors disponibles, dont les suivantes :

- `Screen()`, ou `turtle.Screen()` : affiche la fenêtre de dessin.
- `clear()` : efface les dessins effectués.
- `reset()` : idem, repositionne la tortue et remet les variables à leur valeur par défaut.
- `up()` : lève le stylo, et `down()` : abaisse le stylo.
- `forward(d)`, `backward(d)` : avance ou recule la tortue de `d` pixels à partir de sa position courante.
- `goto(x,y)` : déplace la tortue sur un segment de droite depuis sa position courante jusqu'à la position `(x,y)`. Si le stylo est baissé, ce segment de droite sera dessiné.
- `left(a)`, `right(a)` : fait tourner la tortue sur la gauche ou la droite de l'angle `a` (en degrés).
- `setheading(a)` : positionne la tortue à l'angle `a`. Si l'angle est 0, la tortue regarde vers la droite.
- `speed('fastest')` : augmente la vitesse.

Exercice 2.4 On veut dessiner des arbres obtenus récursivement, de la façon suivante :



L'arbre de rang 0 (à gauche) est réduit à un segment vertical. Pour $n \geqslant 1$, l'arbre de rang n est composé d'un segment vertical et de deux arbres de rang $n-1$ commençant chacun en haut du tronc, et inclinés d'un angle donné (par exemple $30°$) symétriquement par rapport au tronc.

1. Écrivez une fonction

   ```
   def arbre(n, longueur, angle, ratio):
   ```

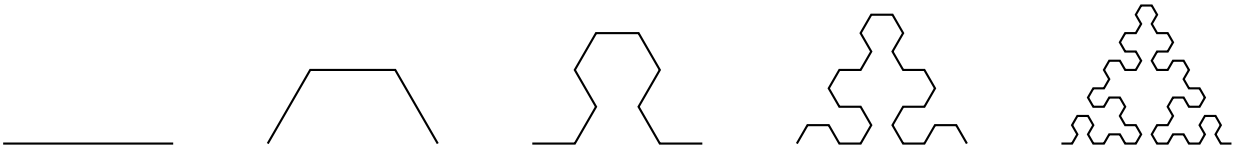
   qui dessine l'arbre de rang n. La longueur (de la base du tronc aux extrémités des branches) est donnée par le second argument. Le troisième argument est l'angle des sous-arbres par rapport au tronc. Le dernier argument est le ratio entre la longueur du tronc et la longueur totale.
2. Combien d'appels récursifs sont-ils effectués par cette fonction ?

Exercice 2.5 Les flocons de Koch sont obtenus en répétant récursivement un motif. Les premiers flocons sont les suivants :

Le flocon de rang 0 (figure de gauche) est simplement un segment. Pour $n \geqslant 1$, le flocon de rang n est obtenu en juxtaposant 4 flocons de rang $n-1$: le premier horizontal, le second incliné de $60°$, le troisième incliné de $-60°$ et le quatrième également horizontal.

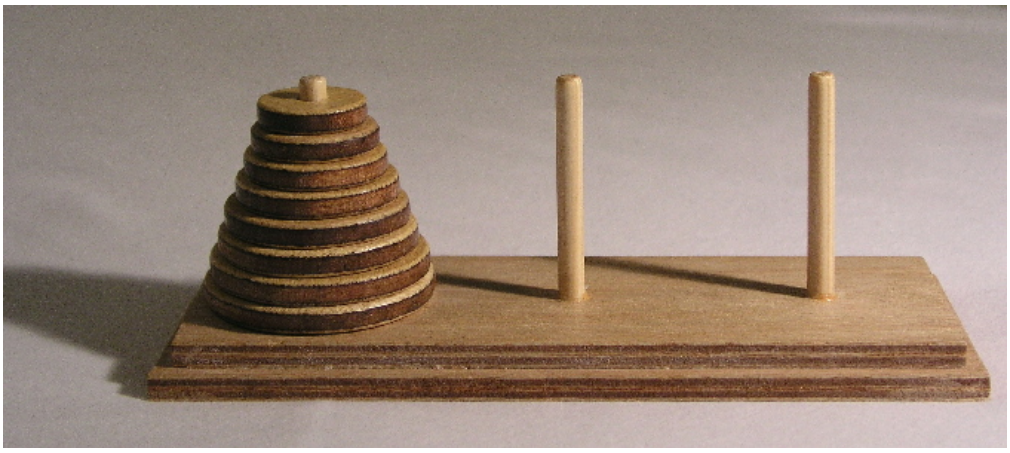
1. Écrivez une fonction `koch(n, longueur)` qui dessine le flocon de rang n, dont la longueur de l'extrémité gauche à l'extrémité droite est donnée par le second paramètre.
2. Combien d'appels récursifs sont-ils effectués par cette fonction ?

Exercice 2.6 Même exercice pour le triangle de Sierpinski dont les premiers rangs sont représentés sur la figure suivante. Testez la fonction pour l'itération 8.



Exercice 2.7 Le jeu des tours de Hanoi se joue avec 3 piquets, appelés d (comme départ), a (comme arrivée), et i (comme intermédiaire), et n disques de tailles différentes que l'on met sur ces piquets.

On part de la configuration où les n disques sont disposés sur le piquet d de départ, comme sur la photo suivante (avec $n = 8$).



L'objectif est de déplacer les disques du piquet d vers le piquet a, en déplaçant un seul disque à chaque étape. La contrainte à respecter est qu'on n'a pas le droit de placer un disque sur un disque plus petit que lui.

1. En supposant que vous savez déplacer les $(n-1)$ disques les plus petits d'un piquet vers un autre en utilisant une suite de mouvements, comment faire pour déplacer les n disques du piquet d vers le piquet a ?
2. Décrire un algorithme récursif permettant de résoudre le jeu, en affichant (par l'instruction `print`) la suite des déplacements.
3. Combien cet algorithme fait-il de déplacements de disques ?