Informatique – T.D. 1

1. © *Algorithme d'Euclide* : on note comme en Python $a \,\%\, b$ le reste de la division euclidienne de l'entier a par l'entier non nul b (lire a *modulo* b). L'algorithme d'Euclide permet le calcul du PGCD (plus grand commun diviseur) de deux entiers naturels en utilisant les propriétés suivantes :

$$\forall a \in \mathbb{N} \quad \mathrm{PGCD}\,(a,0) = a \quad \text{et} \quad \forall b \in \mathbb{N}^* \quad \mathrm{PGCD}\,(a,b) = \mathrm{PGCD}\,(b, a \,\%\, b)\,.$$

Formaliser l'algorithme et le programmer en Python (donner une version itérative **et** une récursive).

2. © *Recherche dichotomique* : l'algorithme du cours reçoit un tableau t et une valeur x et renvoie `True` ou `False` selon que x est présent ou non dans t.

Adapter cet algorithme pour renvoyer l'indice d'une occurrence de x dans le tableau t s'il y figure, -1 sinon. Programmer une version itérative **et** une version récursive.

3. © *Évaluation d'une fonction polynomiale : algorithme de Horner*

On suppose ici les coefficients $a_0, \ldots, a_n$ du polynôme P stockés dans un tableau a. On doit écrire une fonction recevant comme paramètres a et une valeur réelle x et renvoyant

$$P(x) = \sum_{k=0}^{n} a_k x^k.$$

Évaluer le nombre d'additions et de multiplications de réels effectuées en cas de programmation "naïve".

a) *Version itérative*

Justifier l'algorithme de Horner : si l'on pose

$$y_0 = a_n \quad \text{et, pour } i \text{ allant de 1 à } n, \quad y_i = x \cdot y_{i-1} + a_{n-i}$$

alors $y_n = P(x)$ et l'on n'a effectué que n additions et n multiplications !

Implémenter cela en Python.

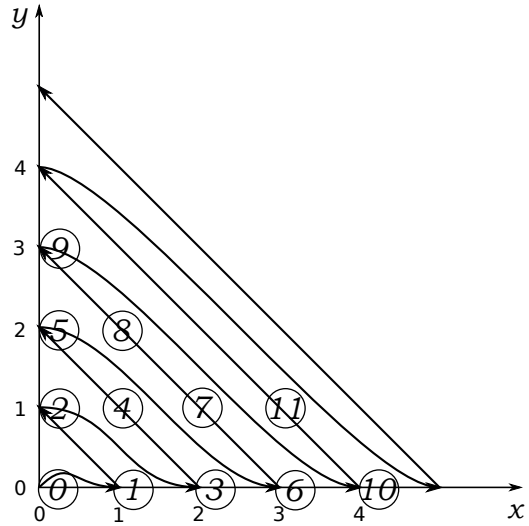
b) *Version récursive*

On peut voir les choses ainsi : en notant $P_j\,(x) = \sum_{k=j}^{n} a_k x^{k-j}$, on veut calculer $P_0\,(x)$, or

$$P_n\,(x) = a_n \quad \text{et, si } j < n, \quad P_j\,(x) = a_j + x \cdot P_{j+1}\,(x)\,.$$

Traduire cela par un programme Python récursif.

4. *Numérotation diagonale* : on peut établir une bijection de $\mathbb{N}^2$ dans $\mathbb{N}$ suivant le schéma classique



Écrire deux fonctions récursives `entier(x,y)` et `couple(n)` calculant l'entier n correspondant au couple (x, y) et (réciproquement !) le couple (x, y) correspondant à l'entier n.

5. *Calcul des coefficients binomiaux*

Pour (n,p) donné dans $\mathbb{N}^2$ tel que $1 \leq p \leq n$, on souhaite calculer $\binom{n}{p}$ uniquement à l'aide d'additions à l'aide de les relations du triangle de PASCAL :

$$\forall\,(n,p) \in \mathbb{N}^2 \quad \binom{n}{0} = 1, \quad \binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p} \text{ pour } p \in [\![1,n]\!] \quad \text{et} \quad \binom{n}{p} = 0 \text{ pour } p > n.$$

a) Écrire une première fonction ***récursive*** recevant n et p et renvoyant $\binom{n}{p}$, par application directe des relations ci-dessus. Quel est le nombre d'additions effectuées lors du calcul de $\binom{n}{p}$?

b) Pour revenir à une complexité "raisonnable", écrire une fonction ***itérative*** recevant n et p et renvoyant $\binom{n}{p}$, calculant de proche en proche les $\binom{i}{j}$, pour des couples (i,j) que l'on précisera, en les stockant dans un tableau. Quel est le nombre d'additions effectuées lors du calcul de $\binom{n}{p}$?

c) Améliorer l'algorithme précédent pour n'utiliser qu'un tableau **à une dimension** pour le stockage.

6. Écrire un algorithme récursif déterminant si un mot donné est un palindrome, puis un algorithme récursif déterminant si deux mots donnés sont l'anagramme l'un de l'autre.

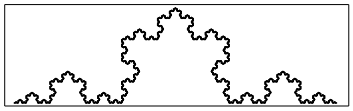
7. *Les tours de Hanoï* : le matériel de ce casse-tête classique est constitué de trois piquets, numérotés 1,2,3 et de n rondelles de diamètres différents percées en leur centre de sorte qu'elles peuvent être enfilées sur les piquets. Dans la position initiale, les rondelles sont enfilées par ordre de diamètre décroissant sur le piquet 1 ; le but du jeu est de les transférer vers le piquet 3 en respectant la règle suivante :

"déplacer une seule rondelle à la fois, de sorte qu'à aucun moment
il n'y ait une rondelle posée sur une autre de diamètre inférieur".

Écrire un programme récursif affichant une succession de déplacements permettant de résoudre ce problème.

Déterminer le nombre de déplacements effectués en fonction de n.

8. *Le flocon de Von Koch* : un peu de dessin pour changer... Le flocon de Von Koch est une célèbre figure fractale obtenue (intuitivement...) de la façon suivante : à tout segment du plan euclidien $[A,B]$ (orienté de A vers B), on associe la ligne brisée $AA_1A_2A_3B$ suivante (où A_1 et A_3 partagent le segment initial en trois parts égales et où le triangle $A_1A_2A_3$ est équilatéral). On applique alors cette transformation à un segment initial, puis on l'applique à chacun des 4 segments obtenus, puis à chacun des 16 segments obtenus et ainsi de suite... Ce qui donne la figure ci-dessous. Bien entendu le processus est arrêté lorsque les nouveaux segments à créer deviennent indiscernables à l'œil nu. Mais en théorie un agrandissement de n'importe quelle portion de la "courbe" montre une figure semblable à la figure complète, c'est l'idée même de fractale...



Écrire une procédure récursive recevant pour paramètres deux points a et b et traçant le flocon associé. Pour faire des calculs sur les coordonnées des points, il est conseillé de stocker les coordonnées $[x,y]$ dans un tableau `numpy.array`. Mais attention ! C'est la commande `pyplot.plot([xa,xb],[ya,yb])` qui trace le segment $[a,b]$ (la première liste contient les abscisses, la seconde les ordonnées).

Pour obtenir une figure ressemblant davantage à un flocon, on peut utiliser trois fois le programme précédent :