CAHIER D'EXERCICES

ALGORITHMIQUE

&

PROGRAMMATION

LANGAGE PYTHON

ISEL 2

Révisions générales

Exercice 1: Rôle des fonctions et procédures

Quand faire appel à une fonction ou à une procédure ? Quand utilise-t-on une fonction plutôt qu'une procédure ? Et une procédure plutôt qu'une fonction ?

Exercice 2: Passages de paramètres

Retrouver comment fonctionne le passage de paramètre des sous-programmes en langage Python.

Exercice 3: Nombres de Armstrong

Parmi tous les entiers strictement supérieurs à 1, seuls quatre peuvent être représentés par la somme des cubes de leurs chiffres. Ces nombres sont appelés nombres de Armstrong. Ainsi, par exemple 153 est un nombre de Armstrong, en effet on a : $153 = 1^3 + 5^3 + 3^3$.

Ecrire un algorithme, puis un programme qui permet de déterminer les quatre nombres de Armstrong.

Indication : Les nombres d'Armstrong sont des nombres à 3 chiffres.

Exercice 4: Le nombre à construire

Écrire un algorithme, puis un programme, qui attend une suite de 4 chiffres de manière à représenter un nombre entier. L'utilisateur devra pouvoir effectuer des saisies successives et l'algorithme acceptera tout caractère saisi mais :

- il n'affichera « en écho » à l'écran que les chiffres,

- il ne tiendra pas compte des caractères différents d'un chiffre,

- il ne tiendra pas compte des nombres,

- il attendra d'avoir reçu effectivement 4 chiffres (et non seulement 4 caractères quelconques) pour passer à la suite.

On demande ensuite d'afficher le nombre de 4 chiffres ainsi que le double de ce nombre.

Remarque : L'accès aux code ASCII des caractères courants s'effectue à l'aide des fonctions chr() et ord() : **chr**(n) renvoie le caractère dont le code Ascii est n, et **ord**(c) renvoie le code Ascii du caractère c (par exemple, si le caractère est le chiffre 2, on écrira ord('2')).

Note : Pour les exercices suivants :

- Prévoir de s'assurer du type des variables saisies par l'utilisateur ;

- Pour chaque programme où cela sera pertinent, des sous-programmes pourront être mis en œuvre.

Exercice 5: Moyenne arithmétique et écart-type

Écrire un algorithme, puis un programme qui prend en entrée une suite de nombres positifs (saisis un par un).et qui en calcule la moyenne et l'écart type afin de les afficher. La fin de la suite sera repérée par un nombre négatif (le seul) qui ne sera pas à prendre en compte pour les calculs. On ne connaît pas à priori le nombre d'éléments de la suite, et l'on interdit l'utilisation de tableaux.

Indication : Si les éléments de la suite sont notés x_i pour i de 1 à n (n le nombre d'éléments de la suite), et si on note μ la moyenne, on rappelle que l'écart type σ est défini par :

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

On pourra montrer que l'on a l'égalité suivante :

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n} - \left(\frac{\sum_{i=1}^{n} x_i}{n}\right)^2}$$

Exercice 6: Le distributeur automatique

Un distributeur automatique dispose, pour rendre la monnaie, de 20 billets de 5 euros et 100 pièces de 1 euro. Ecrire un algorithme, puis un programme, qui :

- demande à l'utilisateur le prix de l'article choisi (en euros entiers) et la somme qu'il va introduire dans le distributeur ;

- calcule la monnaie à rendre à l'utilisateur en billets de 5 euros et pièces de 1 euro ;

- déduit de ses propres ressources le nombre de billets et pièces rendues ;

- propose un nouvel achat jusqu'à ce qu'il n'y ait plus de monnaie à rendre.

Indication : la caisse contenant la monnaie à rendre et celle recevant l'argent des clients sont distinctes (il arrivera donc un moment où il ne sera plus possible d'effectuer un achat, indépendamment du nombre d'articles restant disponibles).

Exercice 7: Facteurs premiers

Écrire un algorithme, puis un programme qui décompose des nombres entiers en facteurs premiers. Rappelons qu'un nombre est premier s'il admet exactement deux diviseurs distincts (1 et lui-même, 1 n'est donc pas premier).

Exemple : 2*2*3*7*11*23 = 21252.

Chaque diviseur trouvé sera affiché à l'écran.

Exercice 8: Crible d'Erathostène pour les entiers naturels

Le crible d'Erathostène (env. -276 , -194 av. J.C.) est une méthode de recherche de tous les nombres premiers inférieurs à un entier naturel N donné. Le principe est de supprimer de la liste des entiers de 1 à N tous les nombres multiples d'un autre entier. Le procéde peut se décomposer comme suit :

- Considérer la liste de tous les entiers de 1 à N ;
- Eliminer 1 de la liste ;
- Conserver 2, et éliminer tous les multiples de 2 ;
- Pour tous les entiers i non déjà éliminés (donc nécessairement impairs), de 3 à $E(\sqrt{n})$ exclue (partie entière de $\sqrt{n}$), conserver i puis éliminer tous les multiples de i ;

Les entiers restant sont premiers.

Ecrire un algorithme puis un programme Python mettant en œuvre le crible d'Erathostène.

Exercice 9: PPCM

Écrire un algorithme, puis un programme, qui calcule le PPCM (Plus Petit Commun Multiple) de deux nombres entiers. On commencera par proposer une méthode manuelle ne faisant pas appel aux facteurs premiers, puis on programmera cette méthode.

Un peu de graphisme

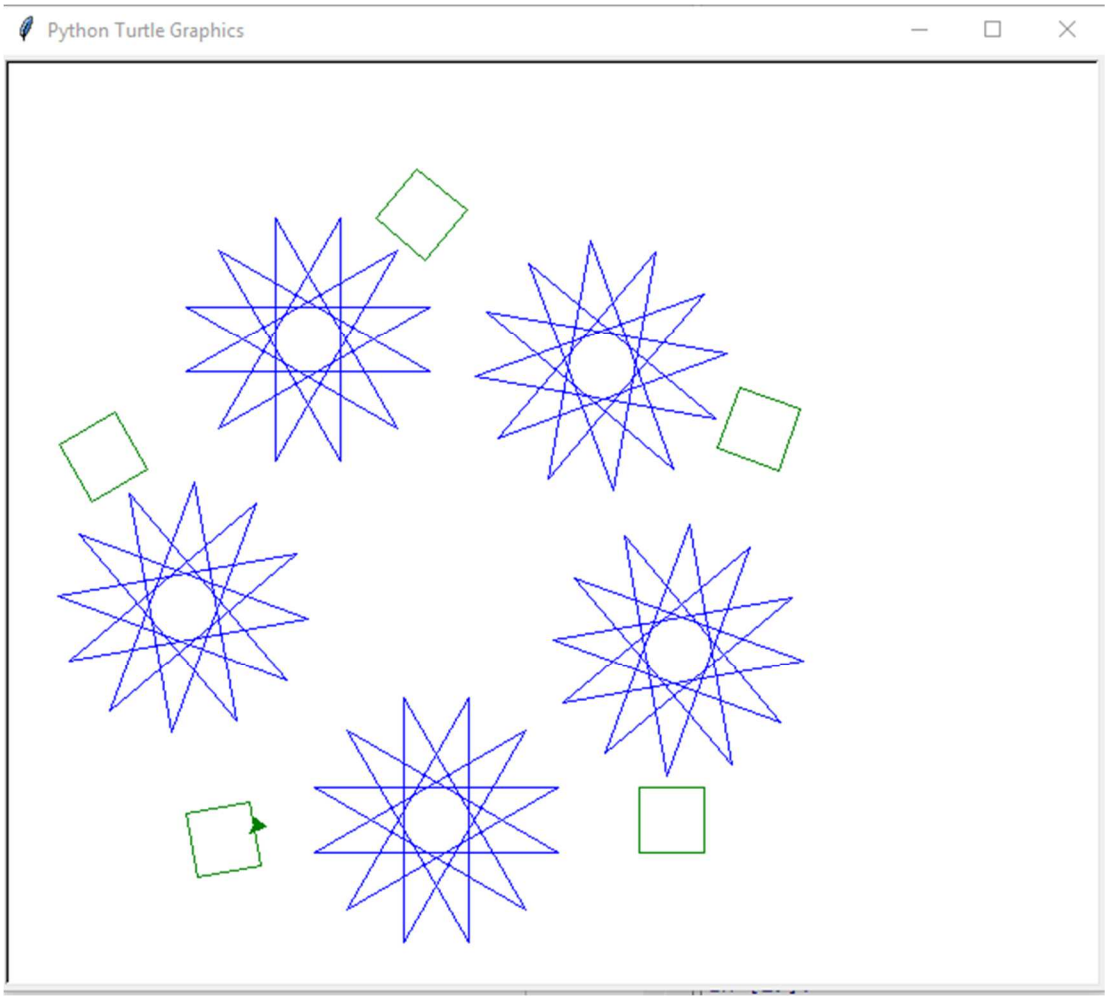
Exercice 10: Guirlande de formes

Reprendre l'exemple de dessin suivant :

```
from turtle import *
a=0
while a < 12:
    a=a+1
forward(150)
left(150)
exitonclick()


#quelques fonctions graphiques possibles supplémentaires:
#   setup(a,b,c,d) définit la taille de la fenêtre et sa position en px : a = largeur, b = hauteur,
# c = abscisse du coin haut gauche de la fenêtre par rapport au même coin de l'écran,
# d= ordonnée du coin haut gauche de la fenêtre par rapport à celui de l'écran.
# Si seulement a et b indiqués , fenêtre centrée par défaut sur l'écran, si a et b non précisés :
#  largeur de 50% écran et hauteur de 75% écran par défaut ; dans ce cas, écrire :
# setup(startx = c, starty = d) pour placement de la fenêtre.
#    goto(x,y) aller au point (x,y)
#   forward(d) avancer d'une distance donnée d
#   backward(d) reculer de d
#   up() relever le crayon ( pouvoir se déplacer sans dessiner)
#   down() abaisser le crayon pour recommencer à dessiner
#   color('blue') dessiner en bleu
#   left(angle) tourner à gauche d'un angle en degrés
#   right(angle) tourner à droite d'un angle donné
#   width(epaisseur) choisir l'épaisseur du trait
#   fill(1) remplir un contour fermé de la couleur sélectionnée
#   write(texte) écrit la chaîne de caractères texte là où est positionné le stylo
#exitonclic()
```

Compléter le programme en créant un sous-programme réalisant le dessin ci-dessus. Puis, définir un sous-programme « carre() » dessinant un carré afin de reproduire le dessin ci-dessous.



Listes et tableaux

Exercice 11 Suite de Syracuse

La suite de Syracuse est une suite mathématique d'entiers naturels $(S_n)_{n>0}$ définie comme suit :

- choisir un entier S_1 strictement positif,

- pour tout n>1, le $n+1^{ième}$ terme de la suite est obtenu à partir du précédent selon la règle suivante :

  - si S_n est pair, $S_{n+1} \leftarrow S_n / 2$ (division entière),
  - si S_n est impair, $S_{n+1} \leftarrow 3*S_n + 1$.

Partant d'un entier p, on appelle la suite de valeurs obtenues la suite de Syracuse du nombre p. On appelle également cette construction de suite de nombres l'algorithme de Collatz. En effet, ce problème a été soulevé par le mathématicien allemand Lothar Collatz en 1937. C'est lors de la présentation de ce problème sur les itérations d'entiers à l'université de Syracuse aux Etats-Unis qu'il prit le nom de "suite de Syracuse" (au cours des année 1950-1960).

Conjecture de Syracuse

La particularité de cette suite est caractérisée sous le nom de "conjecture de Syracuse", ou "conjecture de Collatz" :

Pour tout entier positif S_1=p, il existe un rang n tel que S_n =1.

Il s'agit d'une conjecture, puisque cela n'a pas pu être démontré mathématiquement, mais le résultat a été vérifié pour tout nombre entier positif inférieur à 2^{62}. Aucun contre-exemple n'a pu être trouvé jusqu'à présent.

Exemples

Si l'on part de la valeur 14, on obtient la suite :

14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2,...

Partant de 15, on obtient :

15, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1….

Les valeurs croissent et décroissent, mais une fois la valeur 1 atteinte, la suite "boucle" sur la série de nombres 1,4,2....

Quelques caractéristiques

On définit "le **vol**" du nombre de départ p comme étant la suite des nombres obtenus, chaque nombre étant une "**étape**" du vol. "**L'altitude maximale**" d'un vol est définie comme le plus grand nombre de la suite, et la "**durée**" du vol (ou "temps de vol") est le nombre d'étapes jusqu'à l'obtention du nombre 1 (il s'agit donc du rang n pour lequel on atteint la valeur 1 à partir de l'entier p de départ choisi). Le "**temps de vol en altitude**" est le plus petit rang n tel que le n+1 e terme est inférieur ou égal à la valeur de départ p.

Dans l'exemple avec p=14, l'altitude maximale est 52, la durée du val est 18, et le temps de vol en altitude est 1.

Écrire les algorithmes des sous-programmes permettant d'obtenir :

- la $n^{ème}$ valeur d'une suite, partant d'un entier fixé (étape d'un vol);
- les n premières valeurs d'une suite (n donné en entrée, le vol);
- l'altitude maximale d'un vol,
-la durée d'un vol,

- le temps de vol en altitude.

Écrire les programmes Python correspondant ainsi que le programme global de mise en œuvre.

Exercice 12 – Nombres abondants-déficients-parfaits

Dans cet exercice, nous ferons l'hypothèse que tous les nombres sont des entiers positifs non nuls.

Écrire un programme permettant à l'utilisateur de saisir deux entiers n et p.

Écrire un sous-programme divise(p,n) qui renvoie vrai si p est un diviseur propre de n, et faux sinon. Un diviseur propre de n est un diviseur de n mais différent de n. Compléter votre programme de manière à afficher un message du type « 5 est un diviseur propre de 15 » ou encore « 2 n'est pas un diviseur propre de 11 ».

Écrire un sous-programme permettant d'afficher sur la même ligne tous les diviseurs propres d'un entier n passé en argument. Tester ce sous-programme.

Écrire un sous-programme permettant de calculer la somme de tous les diviseurs propres d'un entier n. Tester ce sous-programme.

Un nombre n est dit **abondant** si la somme de ses diviseurs propres est strictement supérieure à n. Un nombre n est dit **déficient** si la somme de ses diviseurs propres est strictement inférieure à n. Un nombre n est dit **parfait** si la somme de ses diviseurs propres est égale à n. Écrire trois fonctions estAbondant(n), estDeficient(n) et estParfait(n) à résultat booléen qui suivent les définitions données précédemment. Tester ces fonctions.

Écrire une procédure permettant d'afficher la liste des entiers de 1 à k, k étant l'argument de la procédure. L'affichage se fera de la manière suivante : on affiche en rouge les nombres abondants, en vert les nombres déficients et en blanc les nombres parfaits.

```
Donnez n : 11
Donnez p : 2
2 n'est pas un diviseur propre de 11
Les diviseurs propres de 11 sont :    1
La somme des diviseurs propres de 11 est : 1
11 est un nombre deficient

Voici la liste des nombres de 1 a 100
Rouge = Abondant ; Vert = Deficient ; Blanc = Parfait
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30
 31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
 61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 91  92  93  94  95  96  97  98  99 100
```

Exercice 13: Recherche par dichotomie

Écrire un algorithme, puis un programme Python qui réalise les étapes suivantes :

- Demander à l'utilisateur combien (n) il souhaite de valeurs dans une liste L,
- Remplir la liste L avec n valeurs tirées aléatoirement dans l'intervalle de valeurs entières [1,n^2],
- Trier en ordre croissant la liste L (en Python, appliquer à L la fonction sort() permettant d'ordonner ses valeurs (syntaxe : L.sort()),

- Afficher le contenu trié de la liste L,
- Demander à l'utilisateur une valeur v dans l'intervalle [1, n^2],
- Afficher la première position de v dans la liste L si elle est présente, et -1 si elle n'est pas présente.
- Appliquer ensuite une méthode de recherche par dichotomie pour afficher si la valeur v est présente ou non dans la liste L (si elle est présente, afficher la position trouvée),.
- Appliquer ensuite une fonction de mélange sur la liste L,
- Afficher enfin la première position de la valeur v dans la liste mélangée, et -1 si v n'est pas dans L.

Exercice 14: Jeu de Conway

Le jeu de Conway est également connu sous le nom de « Jeu de la vie ». Il doit son nom à John Horton Conway, mathématicien britannique, qui le modélisa en 1970. Il s'agit d'un jeu fondé sur une structure d'automate cellulaire, qui ne fait appel à aucun joueur mais où chaque étape résulte automatiquement de la configuration précédente. Le principe en est le suivant :

Considérons une grille à deux dimensions composée de cellules sur n lignes et m colonnes. Chacune de ces cellules (cases) peut ne peut prendre que l'une de deux valeurs : « vivante » ou « morte ».

Le jeu commence par un remplissage aléatoire de la grille avec les valeurs ci-dessus.

Puis, à chaque étape (ou « tour ») du jeu, la grille est modifiée comme suit :

- Si une cellule morte est entourée de 3 cellules vivantes, elle devient vivante (elle naît), sinon elle reste dans son état ;
- Si une cellule vivante possède 2 ou 3 cellules voisines vivantes, elle reste dans son état, sinon elle meurt ;

Remarque : les voisines d'une cellule sont toutes les cellules qui l'entourent : horizontalement, verticalement, ou en diagonale.

Définir les structures de données permettant de modéliser ce problème, puis écrire un algorithme permettant de résoudre le problème.

Implémenter ensuite l'algorithme sous forme de programme Python.

On pourra envisager une solution en mode texte, puis une solution en mode graphique.

Exercice 15:Jeu de Tic-Tac-Toe

Deux joueurs se voient attribuer chacun un symbole. A tour de rôle, chaque joueur choisit une case libre sur une grille de 3 lignes et 3 colonnes, et place son symbole dessus. Le premier joueur qui réussit à remplir une ligne horizontale, verticale ou diagonale avec son symbole est déclaré vainqueur. Il se peut qu'il n'y ait pas de vainqueur lorsque la configuration est telle qu'après quelques tours de jeu les deux symboles sont présents sur chaque ligne et diagonale.

- Préparer la mise en œuvre de l'algorithme en faisant appel à une analyse descendante ;
- Détailler l'algorithme en faisant appel à des sous-programmes ;
- Implémenter le programme correspondant.

Récursivité

Exercice 16: Puissance récursive et itérative d'un réel

Ecrire un algorithme récursif permettant de calculer x^n pour x réel et n un entier. Compléter avec le programme Python. Représenter sous forme d'arbre d'appels la suite d'appels récursifs nécessaires pour aboutir au résultat. Comparer le nombre d'opérations à réaliser par cet algorithme avec celui d'une méthode itérative de calcul de x^n.

Exercice 17: Suite de Fibonacci

Faire appel à un sous-programme récursif pour calculer les nombres de Fibonacci. Compléter en écrivant un algorithme, puis le programme, affichant les n premiers nombres de la suite, n étant demandé à l'utilisateur. Ecrire sous forme d'arbre d'appels la suite d'appels récursifs nécessaires pour aboutir au résultat.

Exercice 18: Crible d'Erathostène récursif

Reprendre le principe du crible d'Erathostène en donnant une version récursive.

Exercice 19 Triangle de Pascal récursif

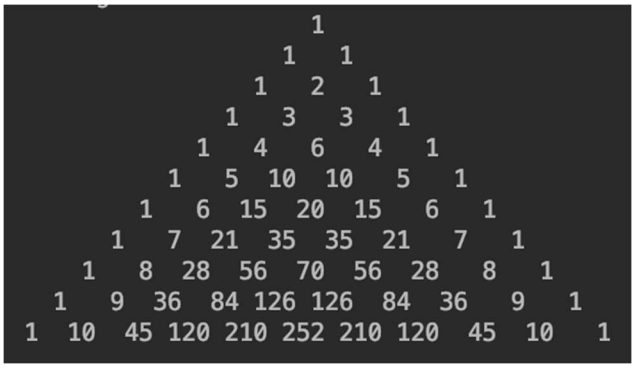
En mathématiques, le triangle de Pascal est une présentation des coefficients binomiaux dans un triangle. Pour trouver le $i^{ème}$ terme d'une ligne, on effectue la somme des deux termes sur la ligne précédente sur la même colonne et la colonne précédente. Ci-contre un exemple pour 5 lignes :

```
   1
   1    1
   1    2    1
   1    3    3    1
   1    4    6    4    1
   1    5   10   10    5    1
```

Écrire l'algorithme puis le programme permettant d'obtenir le triangle de Pascal pour un nombre de lignes donné. Le résultat sera stocké dans une liste retournée par la fonction « triangle_pascal » ayant pour paramètre le nombre de lignes.

Dans le programme principal, vous ferez appel à cette fonction et afficherez le résultat tel que présenté dans l'exemple précédent. Le nombre de lignes à afficher sera demandé à l'utilisateur. Chaque nombre sera affiché sur 4 caractères.

Centrer les lignes quel que soit le nombre de lignes à afficher.



Exercice 20: Jeu du pendu

Écrire un programme en Python qui permettra de jouer au jeu du pendu.

Pour cela, vous devrez :

1) Créer une liste d'une vingtaine de mots.
   Par la suite, le programme tirera au sort l'un de ces mots pour le faire deviner au joueur.
2) Fixer le nombre maximal d'erreurs possibles du joueur pour trouver le mot à 10.
3) Demander au joueur son prénom. Celui-ci apparaîtra à chaque tentative du joueur dans les messages affichant le nombre de coups lui restant à jouer.
4) Ecrire un algorithme avec des sous-programmes permettant de mettre le jeu en œuvre. Le joueur aura gagné s'il découvre le mot en ayant fait moins de 10 erreurs, sinon il aura perdu et le mot à trouver sera affiché.
5) Ecrire le programme Python. Compléter en permettant au joueur d'enchaîner autant de parties qu'il le souhaitera. Un score correspondant au nombre de coups restants à chaque partie sera affecté au joueur (par exemple 7 s'il a trouvé un mot en 3 coups (10-7)). Ce score pourra être cumulé au fil des parties jouées.
6)

Exercice 21: Les tours de Hanoï

Les tours de Hanoï sont un jeu inventé par Édouard Lucas en 1883. Ce jeu dérive d'une légende hindoue. Le principe est le suivant : on considère 3 poteaux et n disques de diamètres différents empilés sur un même poteau du plus grand au plus petit.

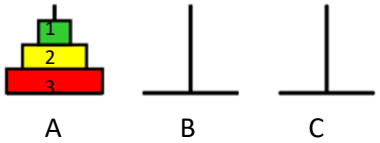
Ecrire un programme permettant de déplacer l'ensemble de la pile d'un poteau à un autre en respectant les principes suivants :

- On ne déplace qu'un seul disque à la fois,
- On ne peut poser un disque que sur un emplacement vide ou un disque de diamètre plus grand.

On pourra le tester pour n=3.

Ce problème peut être représenté de la manière suivante :

Position initiale :



Position finale :

Indication : Pour suivre les déplacements et vérifier le résultat, le programme affichera un message pour chaque « déplacement », indiquant quel disque sera déplacé de quel poteau à quel autre. On pourra obtenir une suite du type :

```
Voici les  7  déplacements à effectuer :
déplacer le disque  1 de pilier A vers  pilier C
déplacer le disque  2 de pilier A vers  pilier B
déplacer le disque  1 de pilier C vers  pilier B
déplacer le disque  3 de pilier A vers  pilier C
déplacer le disque  1 de pilier B vers  pilier A
déplacer le disque  2 de pilier B vers  pilier C
déplacer le disque  1 de pilier A vers  pilier C
```

Exercice 22: Le flocon de Von Koch

Le flocon de Von Koch est une courbe fractale définie comme suit :

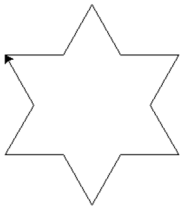
Partant d'un triangle équilatéral, pour chacun de ses côtés, modifier récursivement le segment correspondant à un côté en :

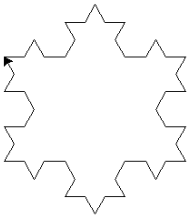
- divisant le segment de droite en trois segments de même longueur,
- construisant un triangle équilatéral ayant pour base le segment du milieu.
- supprimant le segment de base du milieu.

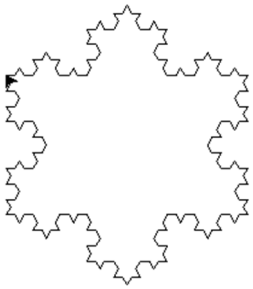
En répétant cela indéfiniment, on obtient une figure fractale figurant un flocon.

Ecrire un programme faisant appel à la récursivité pour représenter cette courbe. Si la fonction s'appelle flocon(n), on pourra obtenir les figures suivantes :

pour l'appel de flocon(1) : pour l'appel de flocon(2) : pour l'appel de flocon(3)







Exercice 23: Résolution d'équations par la méthode des dichotomies

La résolution d'équations est d'une importance capitale en Mathématiques. Cependant, on ne peut obtenirdans de nombreux cas que des valeurs approchées des racines.

Il existe des formules algébriques exactes donnant les racines des équations polynomiales de degré inférieur à 5 et le mathématicien Evariste Gallois a démontré que cette résolution exacte n'était plus possible pour des polynômes de degré supérieur ou égal à 5.

La méthode approchée de résolution d'équations la plus simple est la méthode des dichotomies.

Principe :

Soit f une fonction donnée ; on se propose de déterminer ses zéros sur un intervalle [xmin, xmax] fixé.

Supposons que l'on connaisse un intervalle [a, b] sur lequel la fonction change de signe, c'est-à-dire tel que f(a).f(b) < 0. Si la fonction est continue, nous savons que cet intervalle contient alors une racine de f.

Coupons cet intervalle en deux en posant $c = \frac{a+b}{2}$. Deux cas sont possibles :

- si f(c) est du signe de f(a), la racine appartient à l'intervalle [c, b].
- si f(c) est du signe opposé à f(a), la racine appartient à l'intervalle [a, c].

Nous avons ainsi encadré la racine par un intervalle deux fois plus petit. Il suffit de répéter le processus jusqu'à ce que la précision demandée soit atteinte.

Implémentation d'une première version :

L'utilisateur doit entrer 3 valeurs : le xmin, le xmax et la précision. C'est le programmeur qui aura implémenté directement dans le programme la fonction pour laquelle on désire connaître les zéros (l'utilisateur ne saisit pas la fonction).

Un algorithme pourrait être :

```
début
        saisir(xmin, xmax, prec);
        x := xmin;
        y := xmax ;
        fx := f(x) ;
        fy := f(y) ;
        si fx * fy < 0
        alors
                {on sait que la f change de signe}
                début
                        répéter
                                c := (x + y) / 2;
                                fc := f(c);
                                si fx * fc>0
                                alors
                                        début
                                                x := c ;
                                                fx := fc
                                        fin
                                sinon
                                        y := c ;
                        jusqu'à (abs (x-y) <prec)
                fin
        sinon avertir l'utilisateur que l'on ne sait pas si la fonction admet un zéro sur l'intervalle
fin
```

Améliorations de la méthode dichotomique :

Si l'on utilise la méthode dichotomique précédente, nous avons les inconvénients suivants :

- il faut connaître au départ un intervalle sur lequel la fonction change de signe.

- une seule racine est trouvée sur l'intervalle.

On peut éviter cela en procédant de la manière suivante : On découpe l'intervalle [xmin, xmax] en n sous-intervalles. Puis, sur chacun de ces sous-intervalles, le programme examine si la fonction change de signe ; si c'est le cas, on applique la méthode dichotomique précédente. En choisissant n et l'intervalle de départ suffisamment grand, toutes les racines sont obtenues.

L'algorithme devient alors :

```
début
        saisir(xmin, xmax, prec, n);
        pas := (xmax - xmin) / n ;
        pour i de 0 à n - 1 faire
        début
                x := xmin + i * pas ;
                y := x + pas ;
                fx := f(x) ;
```

```
            fy := f(y) ;
            si fx * fy < O
            alors
                  appliquer la méthode précédente avec x et y
      fin
fin
```

Appliquer les méthodes ci-dessus aux deux exemples suivants.

Exemple 1 :

f(x) = x + 4 * ln (x) - 5

xmin	:	1
xmax	:	10
n	:	10
prec	:	1E-15

Solution 1 : 2.07676138

Exemple 2 :

f(x) = 4 * sin(x) - x + 1

xmin	:	-3
xmax	:	3
n	:	6
prec	:	1E-15

Solution 1 : -2.21008

Solution 2 : -0.342185

Solution 3 : 2.702061

Dictionnaires

Exercice 24:Jeu de Conway – version dictionnaire.

Proposer une version du jeu de Conway faisant appel au type dictionnaire.

Les Tris de données

Exercice 25:Quelques algorithmes de tris.

Il s'agit d'implémenter l'ensemble des tris vus en cours, ainsi que quelques variantes dans un même fichier. Ils seront appliqués au même tableau d'entiers rempli aléatoirement de sorte à comparer les temps d'exécution avec chacune des méthodes.

1) Implémenter le tri à bulle montant (l'élément le plus léger vient d'abord se placer en première position,…) ;
2) Implémenter la version améliorée du tri à bulle, avec arrêt dès que tous les éléments sont bien placés ;
3) Implémenter une version de tri à bulle descendant (l'élément le plus lourd vient d'abord se placer en dernière position) ;
4) Implémenter le tri par insertion en suivant l'algorithme classique ;
5) Implémenter une version du tri insertion faisant appel aux fonctions *insert* et *pop* des listes Python.
6) Implémenter le tri par sélection-échange.
7) Implémenter le tri rapide.

Plusieurs fonctions Python permettent de calculer le temps d'exécution machine d'un bloc d'instructions : on pourra utiliser perf_counter() de la bibliothèque time (ou également process_time()).

Exercice 26 :Tris de structures composées

On pourra reprendre l'ensemble des tris en les appliquant à des tuples de valeurs. Chaque tri sera effectué sur l'un des éléments des tuples.

Systèmes linéaires

Exercice 27 : Résolution de systèmes linéaires

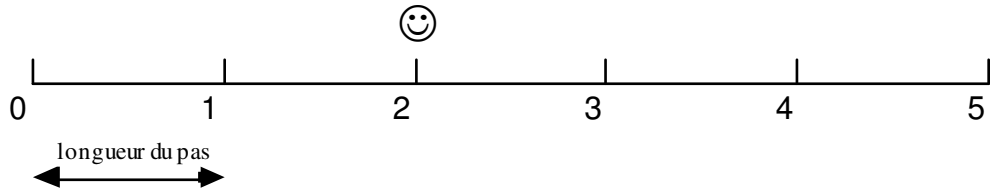
Ecrire un programme permettant la résolution d'un système linéaire par les méthodes de Jacobi, puis de Gauss-Seidel, avec affichage de la solution x et du nombre d'itérations nécessaires à chacune des deux méthodes pour un même epsilon.

Applications numériques

a) Résoudre le système suivant :

$$\begin{cases} -2x_1 + x_2 = -1, \\ x_1 - 2x_2 + x_3 = 0 \\ x_2 - 2x_3 + x_4 = 0 \\ \quad x_3 - 2x_4 = 0 \end{cases}$$

b) Un promeneur marche au hasard le long d'un chemin. En un point donné, il a les mêmes chances d'aller à gauche ou à droite. Il s'arrête lorsqu'il arrive en 0 ou en 5.



Calculer pour tout point k (k=1 à 4), la probabilité que partant de k , il s'arrête au point 0.

Table des matières