Algorithmique

Procédures, fonctions et fonctions récursives

Anicet E. T. Ebou, ediman.ebou@inphb.ci

Préambule motivant

Ecrivez un algorithme permettant de calculer la somme suivante pour les valeurs de x = 2, 3 et 4: $\sum_{k=0}^{n} x^k$

Préambule motivant

```
n = int(input(“Entrez l’exposant maximale de la somme”))
x,y,z = 0,0,0
for i in range(0, n + 1):
     x = x + 2**i
for i in range(0, n + 1):
     y = y + 2**i
for i in range(0, n + 1):
     z = z + 2**i
print(x,y,z)
```

Préambule motivant

```
n = int(input(“Entrez l’exposant maximale de la somme”))
x,y,z = 0,0,0
for i in range(0, n + 1):
     x = x + 2**i
for i in range(0, n + 1):
     y = y + 2**i
for i in range(0, n + 1):
     z = z + 2**i
print(x,y,z)
```

1

2

3

Préambule motivant

On observe sur la slide précédente une répétition du code pour chaque somme calculée.

Comment éviter cette répétition de code?

En utilisant des fonctions!

01

Notions et syntaxe de base d’une fonction

Notions et syntaxe de base

Une fonction en informatique est une séquence d'instructions, dépendant de paramètres d'entrée (appelés arguments), et retournant un résultat.

Deux points de vue, souvent complémentaires, permettent de préciser ce qu'est une fonction :

- c'est une séquence d'instructions qui permet de réaliser un calcul précis, que l'on peut utiliser plusieurs fois;
- c'est une brique de base d'un problème plus complexe.

Notions et syntaxe de base

La structure générale d'une déclaration de fonction en pseudo-code se fait avec le mot-clé `Fonction` de la façon suivante:

```
Fonction nomFonct(p1,p2:type1; p3:type2):typeRetour
Variable
     Vlocal1:type
     vretour:typeRetour
Debut
     Instruction1
     Retourner(vretour)
FinFonction
```

Notions et syntaxe de base

La structure générale d'une déclaration de fonction en Python se fait avec le mot-clé `def` de la façon suivante:

```
def nom_fonction(a_1,a_2,..,a_k):
    # nom_fonction: nom de la fonction, a_1,..,a_k : arguments
    """ Description de l'action de la fonction """
    instruction 1
    instruction 2
    ....
    instruction p
# ici, on est hors de la définition de la fonction.
```

Notions et syntaxe de base

- La première ligne `def nom_fonction(a_1,...,a_k)` est l'en-tête de la fonction. Les éléments `a_1,...,a_k` sont des identificateurs appelés arguments formels de la fonction et `nom_fonction` est le nom de la fonction.

  Pour une fonction ne prenant pas d'arguments, on écrit simplement `def fonction():`, les parenthèses étant indispensables. Le nom de la fonction est un identificateur qui suit les mêmes règles que les identificateurs de variables.

Notions et syntaxe de base

- La seconde ligne (qui est facultative et peut être sur plusieurs lignes) est une chaîne de caractères appelée chaîne de documentation décrivant la fonction: ce que doivent respecter les paramètres passés en entrée, l'action effectuée et la nature du résultat retourné.
- La suite d'instructions est le corps de la fonction.
- Le retour à une indentation au même niveau que `def` marque la fin de la fonction, tout ce qui est à ce niveau ne fait plus partie de la fonction.

Notions et syntaxe de base

Attention, le rôle d'une définition de fonction n'est pas d'exécuter les instructions qui en composent le corps, mais uniquement de mémoriser ces instructions en vue d'une exécution ultérieure (facultative!), provoquée par une expression faisant appel à la fonction.

Exemple

```
def f(x, n):
    s = 0
    for k in range(n + 1):
        s += x**k
    return s
```

```
Fonction f(x,n: Entier): Entier
Var
    s: Entier
Debut
    s <- 0
    Pour k <- 0 à n + 1 faire
        s <- s + x**k
    FinPour
    Retourner(s)
FinFonction
```

Travaux Pratiques

Écrivez une fonction permettant de tester la parité d’un nombre entier.

02

Notions et syntaxe de base d’une procédure

Similarité des fonctions

L’objectif des procédures est le même que celui des fonctions c'est-à-dire repérer des motifs et ne pas les répéter bêtement à la main.

Différences avec les fonctions

- Les procédures peuvent avoir une influence sur l'état du programme en modifiant les variables globales par exemple;
- Les procédures peuvent ne pas avoir de valeur de retour;
- Si la procédure a une valeur de retour, on peut utiliser l'appel comme on le fait avec une fonction;
- Une procédure sans valeur de retour donnera son résultat en modifiant la valeur d'une variable.

Syntaxe de base en pseudo-code

```
Procedure nomProc(p1,p2:type1; p3:type2)
Var
     Vlocal1:type
     vlocal2:type
Debut
     Instruction1
     Instruction2
     ...
FinProcedure
```

Exemple

```
Algo doubler
Var:
     m, n: Entier
Procedure double(m: entier; n: entier)
Debut
    n <- 2*m
FinProcedure
Debut
    Ecrire("Entrer un nombre: ")
    Lire(m)
    double(m, n)  # pas d'affectation car pas de valeur de retour
    Ecrire(n)
Fin
```

Syntaxe de base en Python

```
def ma_proc(param1, param2, param3):
    global var1, var2
    …
    …
```

Syntaxe de base en Python

En Python, à l'intérieur d'une procédure, on peut accéder :

- en lecture aux arguments passés en paramètre;
- en lecture aux variables qui étaient initialisées avant le `def`;
- en écriture aux variables déclarées avec le mot clef `global` (pour plus de détails, consulter la documentation de `global` et de `nonlocal`).

Exemple

```
def double():
    global n
    n = 2*n


print("Entrer un nombre: ")
n = int(input())
double(n)
print(n)
```

Exemple - Ne fonctionne pas

```
def double(n):
    n = 2*n   # ce n fait de l'ombre au n global


print("Entrer un nombre: ")
n = int(input())
double(n)     # le n global ne pourra pas être modifié ainsi
print(n)      # affiche le nombre entré et non son double
```

03

Fonctions récursives

Notion de récursivité

La notion de récursivité est une notion essentielle, et pas seulement en informatique. En effet, pour expliquer une situation, on utilise souvent cette même situation à un état précédent, voire, dans certains cas plus complexes, on intègre une version de cette situation dans elle-même.

Notion de récursivité

On dit qu'on définit une notion nouvelle de manière récursive lorsque cette notion fait partie de sa propre définition.

Notion de récursivité

Images comportant la notion de récursivité.

Récursivité en informatique

Que fait ce programme?

```
def boucle():
    print("je boucle")
    boucle()
boucle()
```

3.1

Factorielles

Factorielle

L’algorithme est basé sur le fait que

$$n! = 1 \times 2 \times 3 \times \ldots \times (n-1) \times n$$

$$= n \times (n-1) \times \ldots \times 3 \times 2 \times 1$$

$$= n \times \underbrace{(n-1) \times \ldots \times 2 \times 1}_{(n-1)!}$$

$$4! = 4 \times \underbrace{3 \times 2 \times 1}_{3!}$$

Factorielle

```
def fact(n):
    if n == 1:     # comme 0! = 1, on peut aussi tester n == 0
        return 1
    else:
        return n * fact(n-1)
```

3.2

Suites récurrentes

Suite récurrentes

Nous explorerons les suites arithmétiques et géométriques. Pour rappel, elles sont définies telle que suit:

- Suite arithmétique: $U_{n+1} = U_n + r$;
- Suite géométrique: $U_{n+1} = U_n \times q$.

Travaux Pratiques

Proposez une fonction en pseudo-code et en Python permettant le calcul de la suite arithmétique et de la suite géométrique.

Suite arithmétique - Pseudocode

```
Fonction suite_arithmetique(r,n: Entier, zero: reel): Entier
Var
     U: Entier
Debut
    U <- 0
    Pour i <- 1 a n faire
          Si (i = 1) alors
               U <- zero + r
          Sinon
               U <- U + suite_arithmetique(r, zero, n-1) + r
          FinSi
    FinPour
    Retourner U
FinFonction
```

Suite arithmétique - Python

```
def suite_arithmetique(r, zeros, n):
     U = 0
     for i in range(1, n + 1):
     if i == 1:
           U = zeros + r
     else:
           U = suite_arithmetique(r, zeros, n - 1) + r
    return U
```

Suite géométrique - Pseudocode

```
Fonction suite_geom(q: Entier, zero: reel, n: Entier): Entier
Var
     U: Entier
Debut
     U <- 0
     Pour i <- 1 a n faire
          Si (i = 1) alors
               U <- zero * q
          Sinon
               U <- U + suite_arithmetique(q, zero, n-1) * q
          FinSi
     FinPour
     Retourner U
FinFonction
```

Suite géométrique - Python

```
def suite_geometrique(q, zeros, n):
    U = 0
    for i in range(1, n + 1):
    if i == 1:
        U = zeros * q
    else:
        U = suite_arithmetique(q, zeros, n - 1) * q
    return U
```

3.3

Dessins de fractales

Fractales

Une figure fractale ou fractale est une courbe ou surface de forme irrégulière ou morcelée qui se crée en suivant des règles déterministes ou stochastiques impliquant une homothétie interne (Muller, 2015).

Le terme « fractale » est un néologisme créé par Benoît Mandelbrot (qui était polytechnicien par ailleurs) en 1974 à partir de la racine latine *fractus*, qui signifie brisé, irrégulier.

Fractales

Un des meilleurs exemples de fractale donné par la nature est le chou Romanesco (à gauche). Si l'on ne regarde qu'une des pointes, on a l'impression de voir un chou entier.

Fractales

On retrouve de l'auto-similarité dans les fougères : chaque feuille ressemble à la fougère entière.

Domaines d’application

- En biologie, répartition des structures des plantes, bactéries, feuilles, branches d'arbres, etc;
- En géologie, étude du relief, côtes et cours d'eau, structures de roches, avalanches, etc;
- En morphologie animale, structures des invertébrés, plumes d'oiseaux, etc;
- En médecine, structure des poumons, intestins, battements du cœur, etc.

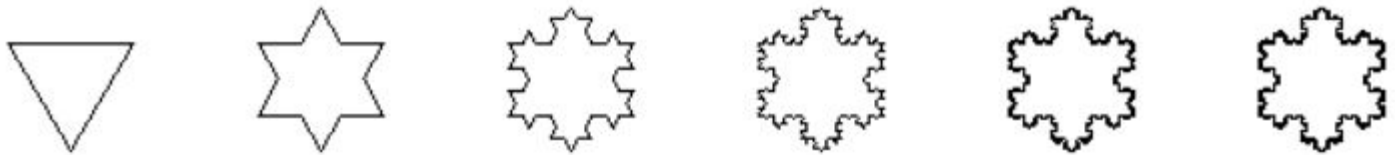
Travaux Pratiques N°2

Flocon de Von Koch

Niels Fabian Helge Von Koch, (Suédois 1870-1924) est un mathématicien qui a donné son nom à l'une des premières fractales: le flocon de Koch ou flocon de neige.

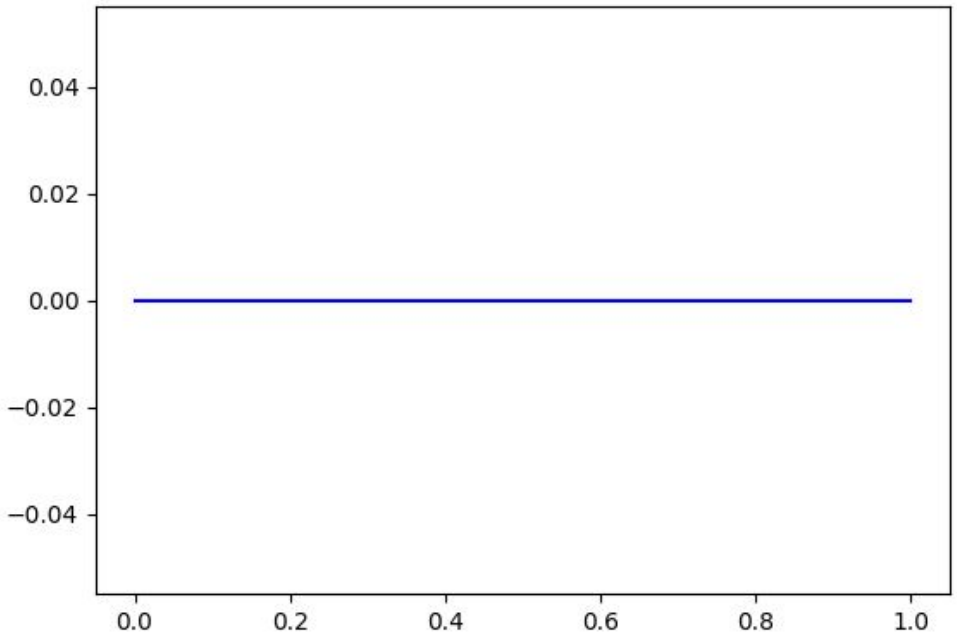
Flocon de Von Koch

Le flocon de Von Koch est défini à partir d'un triangle équilatéral de périmètre 1 (étape i = 0) auquel on construit extérieurement au triangle de manière régulière trois triangles équilatéraux de côté le tiers du triangle précédent (étape i = 1) on répète la construction aux étapes suivantes.



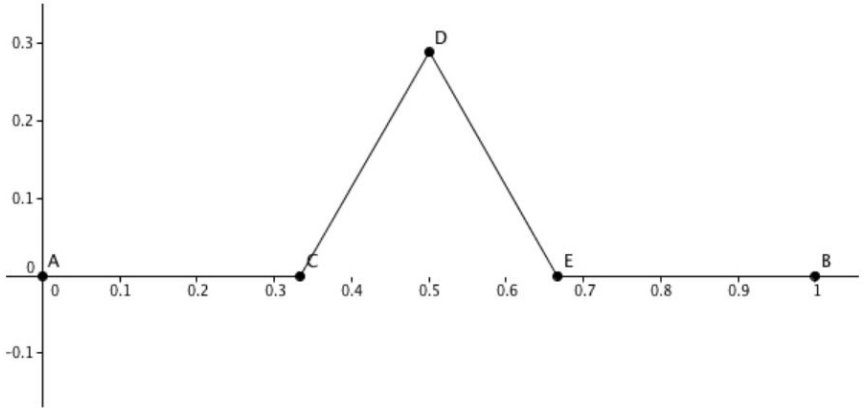
Flocon de Von Koch

De façon pratique, la construction d'une courbe de Von Koch, commence par un segment [A, B]$\in \mathbb{C}$ avec A d'affixe 0 et B d'affixe 1.



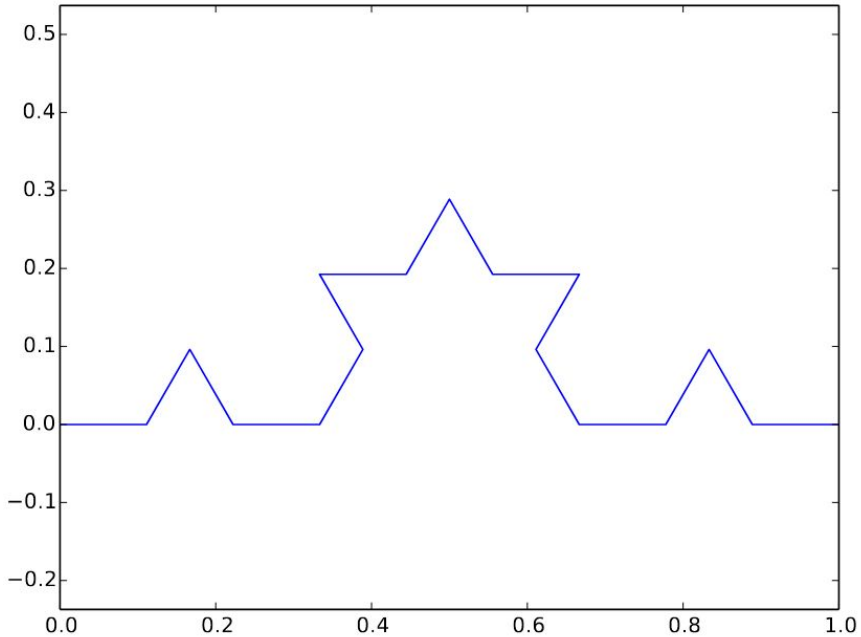
Flocon de Von Koch

Ensuite il faut transformer ce segment en la ligne brisée suivante où C est d'affixe 1/3, E d'affixe 2/3 et CDE est équilatéral.



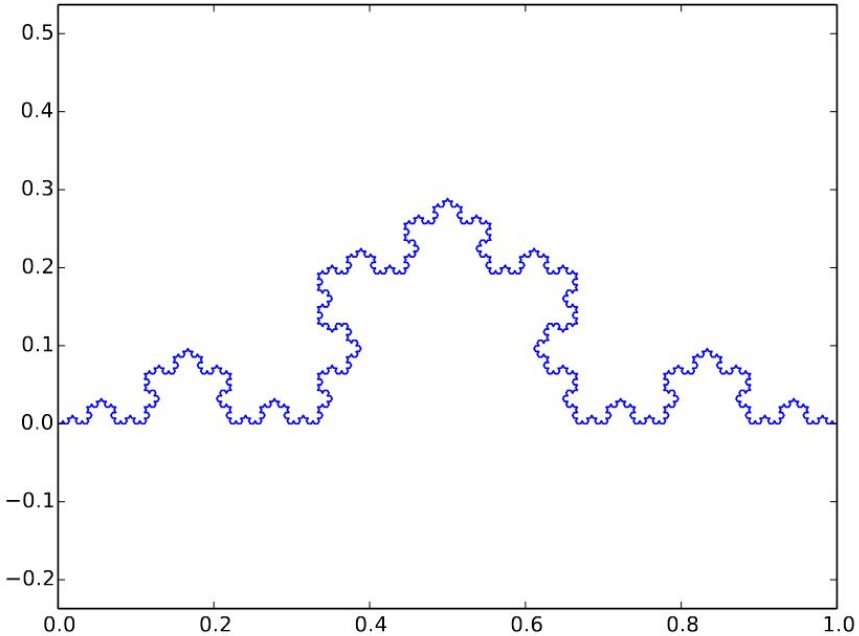
Flocon de Von Koch

Ensuite, on itère cette opération sur chacun des quatre segments de la figure précédente, ce qui donne la figure ci-contre.



Flocon de Von Koch

On peut alors recommencer autant de fois qu'on veut, par exemple après cinq itérations :



Librairie turtle

La librairie turtle est un moyen populaire d'introduire la programmation aux enfants. Pour plus d'informations consulter le lien https://docs.python.org/3/library/turtle.html .

Flocon de Von Koch

Imaginez une tortue robotique commençant à (0, 0) dans le plan x-y. Après avoir importé la librairie, donnez-lui la commande `turtle.forward(15)`, et elle se déplace (à l'écran !) de 15 pixels dans la direction à laquelle elle fait face, en traçant une ligne pendant son déplacement. Donnez-lui la commande `turtle.right(25)`, et elle pivote sur place de 25 degrés dans le sens des aiguilles d'une montre.

Flocon de Von Koch

Commencez par exécuter chaque ligne du code ci-contre en ligne de commande Python pour observer ce qui se passe.

Ensuite recopiez le code ci-contre dans un fichier que vous nommerez koch.py.

```
import turtle

t = turtle.Turtle()
wn = turtle.Screen()
wn.bgcolor('black')

t.color('orange')
t.pensize(5)
t.penup()
t.setpos(-500, 0)
t.pendown()
t.speed(0)  # réglage de visualisation le plus rapide

def koch(t, order, size):
    if order == 0:
        t.forward(size)
    else:
        t.forward(size // 3)      # faire 1/3 du chemin
        t.left(60)
        t.forward(size // 3)
        t.right(120)
        t.forward(size // 3)
        t.left(60)
        t.forward(size // 3)

size = 1000
order = 1
koch(t, order, size)

wn.exitonclick()
```

Flocon de Von Koch

En plus des instructions de tirer vers l'avant, et de tourner à gauche ou à droite, le programme est rendu plus flexible en créant des variables de `size` et `order` et en les passant comme arguments à `koch()`. Puisque l'ordre 0 est l'énoncé le plus simple possible du problème, cela semble être un bon candidat pour notre cas de base. Sinon, la tortue divise en trois la taille donnée et insère le triangle équilatéral implicite dans la section centrale.

Flocon de Von Koch

Prenez un moment pour comprendre que chaque instruction suivante à notre tortue est exécutée à partir de la dernière position et direction de la tortue. De plus, par symétrie, si nous tournons de 60° vers la gauche et de 120° vers la droite, c'est la même chose que de tourner de 60° vers la gauche et de -120° vers la gauche.

Flocon de Von Koch

Nous pouvons donc reformuler le dessin comme une série de virages à gauche, une régularité qui suggère que nous pouvons écrire le bloc `else` entier comme une boucle `for`.

```
def koch(t, order, size):
    if order == 0:
        t.forward(size)
    else:
        for angle in [60, -120, 60, 0]:
            t.forward(size // 3)
            t.left(angle)
```

Flocon de Von Koch

Question: Pourquoi avons-nous besoin du dernier 0 dans la liste ? Que se passe-t-il si nous l'omettons ?

```
def koch(t, order, size):
    if order == 0:
        t.forward(size)
    else:
        for angle in [60, -120, 60, 0]:
            t.forward(size // 3)
            t.left(angle)
```

Flocon de Von Koch

Pour chaque segment A, B, C, D nous voulons insérer une trisection, ce qui implique une itération supplémentaire de `koch()`. Il semble donc que l'on veuille remplacer `t.forward(size // 3)` par l'appel récursif.

```
def koch(t, order, size):
    if order == 0:
        t.forward(size)
    else:
        for angle in [60, -120, 60, 0]:
            koch(t, order - 1, size)
            t.left(angle)
```

Flocon de Von Koch

Comme nous opérons sur un tiers de la longueur de la taille, nous voulons que cette proportion soit préservée, il faut donc modifier l'argument `size` avec `size // 3`.

```
def koch(t, order, size):
    if order == 0:
        t.forward(size)
    else:
        for angle in [60, -120, 60, 0]:
            koch(t, order - 1, size // 3)
            t.left(angle)
```

Flocon de Von Koch

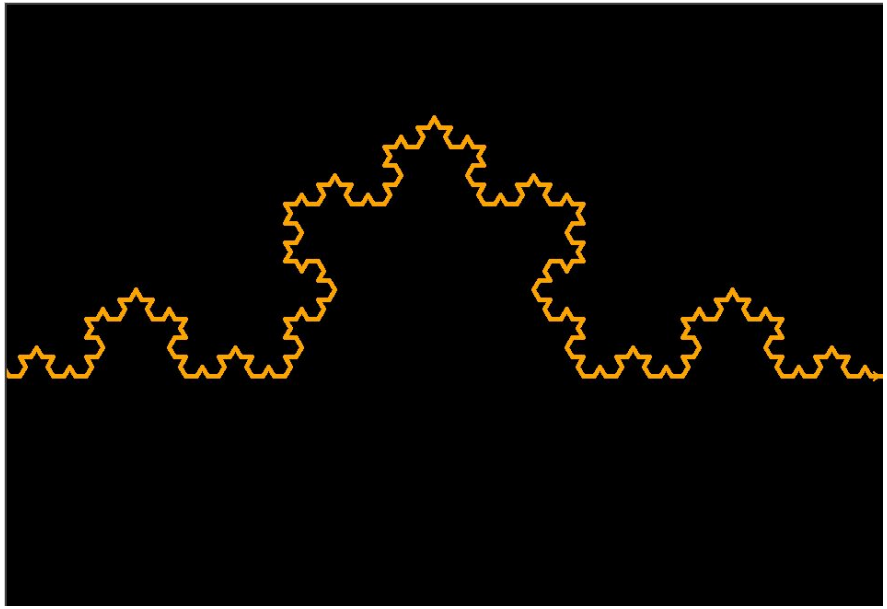
Votre fichier final koch.py devrait contenir le code ci-contre.

Nous exécutons le programme avec un ordre de 3 pour obtenir un flocon complet.

```
import turtle

t = turtle.Turtle()
wn = turtle.Screen()
wn.bgcolor('black')

t.color('orange')
t.pensize(5)
t.penup()
t.setpos(-500, 0)
t.pendown()
t.speed(0)

def koch(t, order, size):
    if order == 0:
        t.forward(size)
    else:
        for angle in [60, -120, 60, 0]:
            koch(t, order - 1, size // 3)
            t.left(angle)

size = 1000
order = 3
koch(t, order, size)

wn.exitonclick()
```

Flocon de Von Koch

On obtient la figure ci-contre.



Flocon de Von Koch

Vous obtiendrez le graphique ci-contre.