La récursivité

Programmation en Python–MPSI–

MPSI2-3

mazza8azzouz@gmail.com

23 décembre 2024

Plan

Rappel

- L'approche efficace pour résoudre un problème complexe consiste souvent à le décomposer en plusieurs sous-problèmes plus simples qui seront étudiés séparément.
- D'autre part, il arrive souvent qu'une même séquence d'instructions sera utilisée à plusieurs reprises dans un programme, et on souhaite évidemment ne pas avoir à la reproduire systématiquement.
- La programmation modulaire permet de résoudre les difficultés évoquées ci-dessus en utilisant des fonctions.
- Toutes les fonctions qu'on a définit jusqu'au maintenant représentent des algorithmes itératifs.
- Une fonction peut appeler une autre fonction. Un cas particulier elle peut appeler elle même, c'est **l'objectif de ce cours**.

Rappel

Syntaxe d'une fonction

La syntaxe python pour la définition d'une fonction est la suivante :
def nomdelafonction (paramètres eventuels) :
  bloc d'instructions

Exemples

1. Exemple 1 : Une fonction qui nécessite un paramètre
```
def cube(x) :
    y=x**3
    return(y)
```
   En entrant print(cube(2)), on obtiendrait l'affichage du nombre 8
2. Exemple 2 : Une fonction ne nécessite pas forcément de paramètre.
```
def table8() :
    n=1
    while n<=10 :
        print(n, " x ",8," = ",n*8)
        n=n+1
```
   L'appel de la fonction lancerait l'affichage de la table de 8.

Récurrence en mathématique

Nous allons commencer par l'exemple de la suite numérique, (U_n) définie pour n $\in$ IN par

$$\begin{cases} U_0 = 1 \\ U_n = 2 * U_{n-1} + 3 \end{cases}$$

- U_n est appelé une suite récurrente.
  $U_n = 2 * (2 * (...(2 * U_0 + 3)...) + 3) + 3$
- La conception d'une fonction récursive n'est pas éloignée du principe de démonstration par récurrence.
- Le principe de démonstration par récurrence est le suivant :
  1. On démontre d'une part que la suite U_n satisfait une telle propriété (croissante, décroissante,....) pour le cas de base U_0
  2. D'autre part, on suppose que cette propriété est valide pour U_{n-1} et on démontre que cela implique que la suite U_n satisfait aussi cette propriété pour tout $n > 0$.

Définition d'algorithme récursif

Fonction récursive

Une fonction est dite récursive si elle s'appelle elle-même au cours de son exécution.

Avantages de la récursivité

- La récursivité permet d'exprimer d'une manière élégante la solution de plusieurs problèmes :
  - Récurrences mathématiques classiques
  - Tour d'Hanoï
  - Tri rapide, tri fusion, ...
  - Recherche dichotomique, ...

  .
- La récursivité est particulièrement adapté lorsqu'elle est appliquée à une structure récursive.
- Les listes et les arbres peuvent être vu comme des structure récursives.

Comment écrire une fonction récursive ?

Principe

- L'idée de base pour écrire une fonction récursive consiste à définir tout d'abord le modèle mathématique de la fonction de récurrence.
- Dans ce modèle de mathémtique, il faut determiner la condition d'arrêt pour assurer **la terminaison de l'algorithme**.

.

Exemple

Nous pouvons donc définir la fonction factorielle de la manière suivante :

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n * (n-1)! & \text{sinon} \end{cases}$$

- n !=n*(n-1) ! représente la relation de récurrence.
- 0 !=1 représente la valeur de la terminaison de l'algorithme.

.

Le code en Python

Méthode itérative

```
def fact_Iter(n) :
    F=1
    for i in range (1,n+1) :
        F=F*i
    return F
```

Méthode récursive

```
def fact_Rec(n) :
    if n==0 :
        return 1
    else :
        return n*fact_Rec(n-1)
```

La récursivité simple

Définition :pour ce type de récursivité on fait un seul appel récursif pour la fonction P dans le corps d'une fonction récursive P.

Exemple 1 : calcul de puissance

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ x * x^{n-1} & \text{sinon} \end{cases}$$

.

- Questions :
  1. Ecrire le code de la fonction récursive Puissance qui retourne la valeur de x^n.
  2. Donner la trace d'exécution pour calculer 2^3

Exemple 2 : calcul de puissance (version rapide)

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ a * a & \text{si n est pair, avec } a = x^{\frac{n}{2}} \\ x * a * a & \text{si n est impair, avec } a = x^{\frac{n}{2}} \end{cases}$$

.

- Questions :
  1. Ecrire le code de cette fonction récursive nommée Puiss_rapide.
  2. Donner la trace d'exécution pour calculer $Puiss\_rapide(3, 5)$

La récursivité multiple

Définition : Une récursivité est multiple si il y a plusieurs appels résursifs a une fonction P dans le corps d'une fonction récursive P.

Exemple 3 : suite de Fibonacci

$$F_n = \begin{cases} 1 & \text{si } n = 0 \\ 0 & \text{si } n = 1 \\ F_{n-1} + F_{n-2} & \text{si } n > 1 \end{cases}$$

.

- Questions :
  1. Ecrire le code de la fonction récursive *Fibo_Re* qui retourne la valeur de F_n.
  2. Donner la trace d'exécution pour calculer F_4

Exemple 4 : Calcul de combinaison

$$C_n^p = \begin{cases} 1 & \text{si } p = 0 \\ 1 & \text{si } n = 0 \\ C_{n-1}^p + C_{n-1}^{p-1} & \text{sinon} \end{cases}$$

.

- Questions :
  1. Ecrire le code de cette fonction récursive.
  2. Donner la trace d'exécution pour calculer C_6^2

Trace d'exécution d'une fonction récursive

Principe de fonctionnement

- L'exécution d'une fonction récursive est basée sur une pile de la mémoire vive.
- La manipulation de cette zone mémoire appelé pile est similaire à la structure données pile traitée cours de la première année.
- On empile les appels successifs dans une pile. (LIFO : Last In First Out)

.

Attention il faut être sur que la fonction se termine. Considérons par exemple cette fonction proche de la factorielle :

Une fonction

$$f(n) = \begin{cases} 1 & \text{si } n = 0 \\ n * f(n+1) & \text{si } n > 0 \end{cases}$$

Que vaut f(4) ?

Trace d'exécution pour calculer 3 !

- Phase 1 : Empiler les apples

PILE (étape 1)
fact(3) = 3*fact(2)

- Dans étape 1 on empile fact(3)
- Appel de fact(3) : retourne 3*fact(2)

.

PILE (étape 2)
fact(2) = 2*fact(1)
fact(3) = 3*fact(2)

- Dans étape 2 on empile fact(2)
- Appel de fact(3) : retourne 3*fact(2)
- Appel de fact(2) retourne 2*fact(1)=2

.

Trace d'exécution pour calculer 3 !

- Phase 1 : Empiler les apples

PILE (étape 3)
fact(1) = 1*fact(0)
fact(2) = 2*fact(1)
fact(3) = 3*fact(2)

- Dans étape 3 on empile fact(1)
- Appel de fact(3) : retourne 3*fact(2)
- Appel de fact(2) : retourne 2*fact(1)
- Appel de fact(1) : retourne 1*fact(0)

.

PILE (étape 4)
fact(0) = 1
fact(1) = 1*fact(0)
fact(2) = 2*fact(1)
fact(3) = 3*fact(2)

- Dans étape 4 on empile fact(0)
- Appel de fact(3) : retourne 3*fact(2)
- Appel de fact(2) retourne 2*fact(1)=2
- Appel de fact(1) : retourne 1*fact(0)
- Appel de fact(0) retourne 1

.

Trace d'exécution pour calculer 3 !

- Phase 2 : Dépiler les apples

PILE (étape 1)
fact(1) = 1*fact(0)=1*1=1
fact(2) = 2*fact(1)
fact(3) = 3*fact(2)

- Dans étape 1 on dépile fact(0)
- Appel de fact(3) : retourne 3*fact(2)
- Appel de fact(2) : retourne 2*fact(1)
- Appel de fact(1) : retourne 1*fact(0)= 1
- Appel de fact(0) : retourne 1

.

PILE (étape 2)
fact(2) = 2*fact(1)=2*1=2
fact(3) = 3*fact(2)

- Dans étape 2 on dépile fact(1)
- Appel de fact(3) : retourne 3*fact(2)
- Appel de fact(2) : retourne 2*fact(1)=2*1=2
- Appel de fact(1) : retourne 1*fact(0)= 1
- Appel de fact(0) : retourne 1

.

Trace d'exécution pour calculer 3 !

- Phase 2 : Dépiler les apples

PILE (étape 3)
fact(3) = 3*fact(2)=3*2=6

- Dans étape 3 on dépile fact(2)
- Appel de fact(3) : retourne 3*fact(2)=3*2=6
- Appel de fact(2) : retourne 2*fact(1)=2*1=2
- Appel de fact(1) : retourne 1*fact(0)= 1
- Appel de fact(0) : retourne 1

.

PILE (étape 4)

- Dans étape 4 on dépile fact(4)
- On dépile le dernier appel (fact(3))
- Le résultat retourné est 6
- La pile devient vide

.

Limitation de la récursivité en Python

- Le langage Python limite, arbitrairement, le nombre d'appels imbriqués à 1000 (la taille de la pile).
- Une fonction qui fait plus de 1000 appels récursifs provoque une erreur
- **Exemple**

```
>>>def fact(n) :
    if(n==0) :
        return 1
    else :
        return n*fact(n-1)
>>>fact(1005)
RuntimeError : maximum recursion depth exceeded
```

- Il existe de nombreuses situations où l'on sait que le nombre d'appels sera bien inférieur à 1000.

Remarque

On peut modifier la taille de la pile par les instructions suivantes :

```
import sys
sys.setrecursionlimit(10000)
```

.

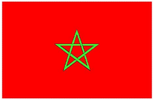
Le module turtle

- Turtle est un module graphique du langage de programmation Python il permet de construire des figures en donnant des instructions a une tortue
- Les principales fonctions du module turtle :

Exception	**Description**
reset()	**Efface la fenêtre graphique, réinitialisation**
up(), down()	**Relève, abaisse le crayon**
forward(d), backward(d)	**Avancer, reculer d'une distance d**
left(a), right(a)	**Tourner à gauche, droite, d'une angle a en degrés**
goto(x,y)	**Se déplace au point de coordonnées (x,y)**
position()	**Retourne la position courante**
color(couleur)	**Détermine la couleur = 'black', 'blue', 'red', ...**
width(l)	**Détermine l'épaisseur du trait l**
fill(1)	**Remplir un contour fermé à l'aide de la couleur sélectionnée (on termine la construction par fill(0))**
write(texte)	**texte doit être une chaîne de caractères délimitée avec des " ou des '**
circle(r)	**Trace un cercle de rayon r**
Mainloop() ou done()	**lance la construction (animation)**

Le module turtle

- **Exemple 1** :Drapeau marocain



- **Code python**

```
from turtle import *
speed(1)##### vitesse entre 0 et 10
width(5)##### epaisseur entre 0 et 10
shape('turtle')##### traceur turtle
bgcolor("red")##### couleur de fond
color('green')#####couleur du tracé
c=2 ##### longueur de la cote
u=100##### unite graphique
right(36) ##### angle 36
for k in range(5) : ##### etoile a 5 branches
    forward(c*u)
    left(144)
exitonclick()
```

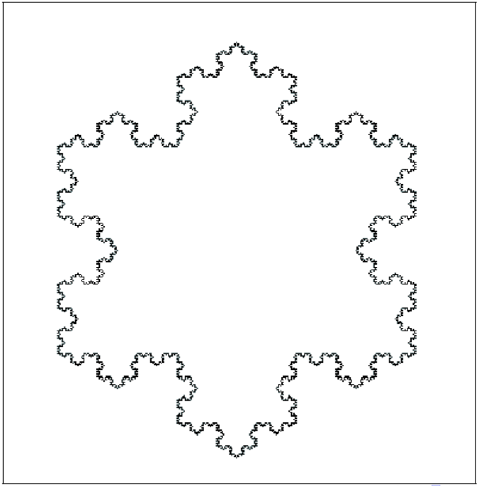
Le module turtle

- **Exemple 2** :polyôgne

- **Code python**

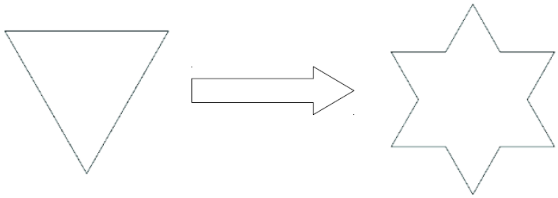
```
import turtle as tt
def polygone(n=3, l=100, clr='black') :
    forward(c*u)
    tt.color(clr)
    tt.down()
    for i in range(n) :
        tt.forward(l)
        tt.left(360/n)
#####Programme principal#####
tt.reset()
polygone()
tt.up()
tt.goto(-200,0)
polygone(12,30,'blue')
```

Flocon de Von Koch



Flocon de Von Koch

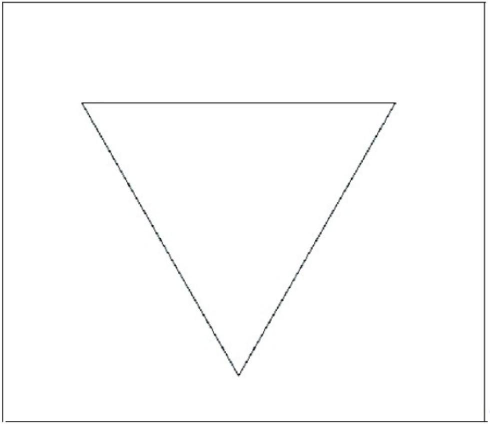
- Cette courbe est construite en partant d'un triangle équilatéral



- Sur chaque segment :
  1. Diviser en 3 segments de mêmes longueurs.
  2. Considérer le triangle équilatéral extérieur construit sur le segment du milieu.
  3. Remplacer ce segment par les 2 autres côtés du triangle équilatéral.
- Recommencer avec chaque segment obtenu.

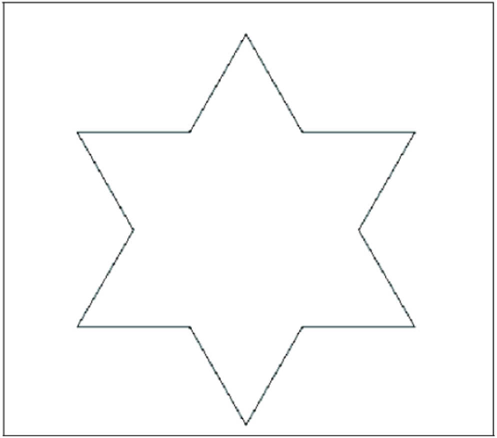
Flocon de Von Koch

$n = 1$



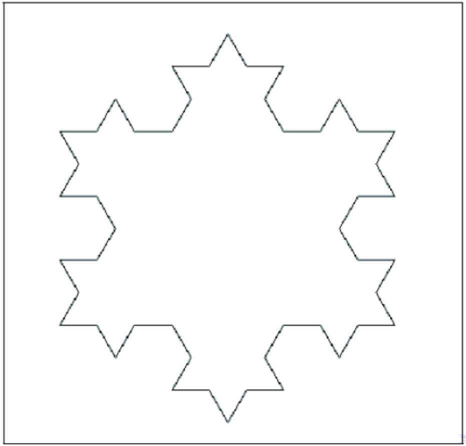
Flocon de Von Koch

$n = 2$



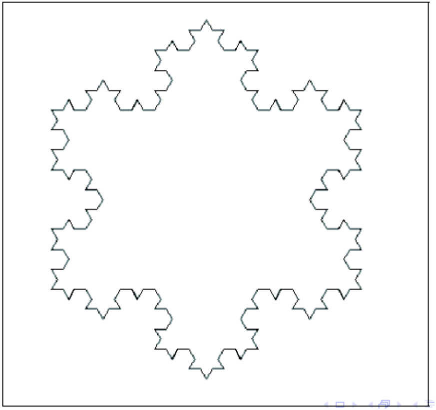
Flocon de Von Koch

$n = 3$



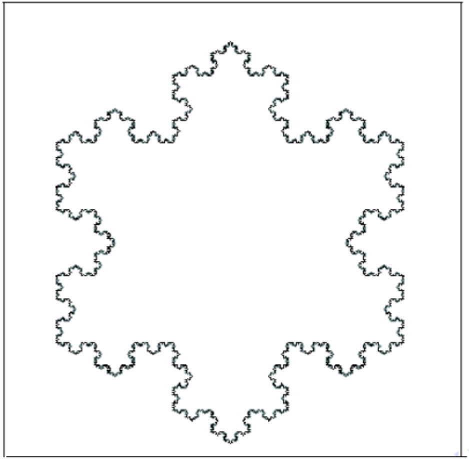
Flocon de Von Koch

$n = 4$



Flocon de Von Koch

$n = 6$



Flocon de Von Koch

- Pour le tracer, on peut procéder par récursivité sur chacun des 3 segments du triangle initial :
  vonKoch(longueur,n)
  1. appeler vonKoch(longueur /3., n-1)
  2. Tourner à gauche de 60X : tt.left(60)
  3. appeler vonKoch(longueur / 3., n-1)
  4. Tourner à droite de 120X : tt.right(120)
  5. appeler vonKoch(longueur / 3, n-1)
  6. Tourner à gauche de 60X : tt.left(60)
  7. appeler vonKoch(longueur / 3., n-1)

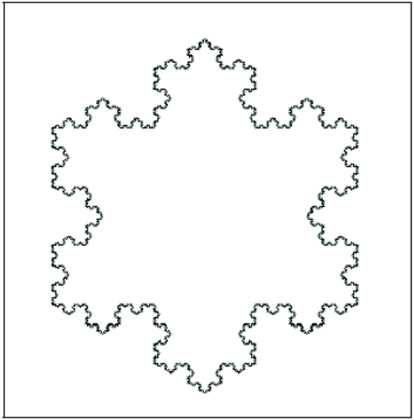
Flocon de Von Koch

```
def vonkoch(longueur,n) :
    if n == 1 :
        tt.forward(longueur)
    else :
        l = longueur / 3.
        vonkoch(l, n - 1) ; tt.left(60)
        vonkoch(l, n - 1) ; tt.right(120)
        vonkoch(l, n - 1) ; tt.left(60)
        vonkoch(l, n - 1)
```

```
def floconVonKoch(longueur, n) :
    tt.pen(speed = 0) # Accélération du mouvement
    tt.hideturtle() # Pour ne pas tracer la tortue
    tt.up()
    tt.goto(-longueur/2., longueur/3.) # Départ en haut à gauche
    tt.down()
    for i in range(3) :
        vonkoch(longueur,n) ; tt.right(120)
```

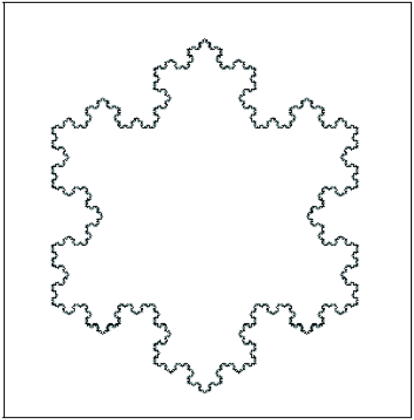
Flocon de Von Koch

- Nous illustrons Flocon de Von Koch avec l'appel suivant :
  >>> floconVonKoch(300,6)



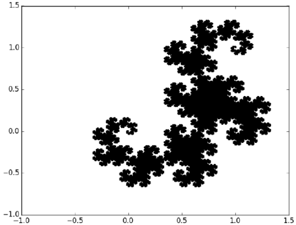
Flocon de Von Koch

- Nous illustrons Flocon de Von Koch avec l'appel suivant :
  `>>> floconVonKoch(300,6)`



La courbe du dragon

À partir d'un segment [AB] , on construit le point C tel que le triangle ACB dans le sens direct soit isocèle et rectangle en C, et on recommence le procédé pour les segments [AC] et [BC] , etc. . .. La courbe du dragon à la ne étape est la réunion de courbe du dragon d'ordre n -1 construite sur [AC] et de celle d'ordre n - 1 construite sur [BC] .
La figure ci-dessous représente la courbe obtenue pour $n = 20$, A(0, 0) et B(1, 1).



Indications : On démontrera d'abord que si A a pour coordonnées (x, y) et B pour coordonnées (z, t) les coordonnées de C sont (u, v) avec :

$$u = \frac{x - y + z + t}{2} \quad \text{et} \quad v = \frac{x + y - z + t}{2}.$$

Triangle de Sierpinski