TP SCHEME TORTUE –Partie II

TP n°2

Après avoir chargé le fichier "SCHEMETORTUE" , on poursuit sur les courbes fractales;

Ces courbes, comme nous l'avons vu avec l'arbre de Pythagore, sont la répétition du même motif élémentaire mais à des échelles différentes. Ces courbes ont un grand intérêt mathématique car elles ont des applications dans des domaines très variés (Sciences Physiques, Sciences Naturelles, biologie, géologie, etc...). Il ne s'agit pas de faire ici un étude mathématique mais d'avoir une approche uniquement ludique.. et esthétique de ces notions....

De la courbe de Von Koch...

1 Soit la courbe de Von Koch définie de la façon suivante :

niveau 0 niveau 1 niveau 2

quand on passe d'un niveau au niveau suivant, chaque segment est divisé en trois parties, les deux parties extrêmes ne changent pas, et on construit un triangle équilatéral sans la base sur la partie centrale

a) Ecrire la procédure (`von_koch prof a`) ou l'on passera en argument la longueur a du trait de niveau 0 et le niveau de profondeur ou l'on veut s'arrêter.

Comme dans toute procédure récursive, il faut réfléchir :
- à la condition d'arrêt. et ce que l'on fait alors
- à la définition du problème au niveau i en fonction du même problème au niveau (i-1)

b) Une fois écrite cette procédure, écrire une procédure une procédure `(prepa-von-koch)` qui nettoie l'écran et positionne la tortue de telle façon à exécuter la courbe de Von Koch sur toute la surface de la fenêtre de dessin.

c) Soit a_0 la longueur du trait au niveau 0; que vaut a_1 la longueur du trait global au niveau 1 en fonction de a_0 ?

d) de façon générale, que vaut a_i la longueur du trait au niveau i en fonction de a_{i-1} la longueur du trait au niveau (i-1)? puis en fonction de a_0 ? Que se passe-t-il pour a_i quand i tend vers l'infini ?

d) En utilisant la primitive DrScheme `(sleep N)` qui met le système en attente pendant N secondes (par exemple `(sleep 3)` met le système en attente pendant 3 s), écrire une procédure `(anim-von-koch n)` qui fasse apparaître successivement la courbe de von koch au niveau 0, 1, 2, jusqu'à n. en laissant 3 s entre les apparitions de chaque niveau.

2 Soit la procédure :

```
(define (flocon n)
  (repeat 3 (von-koch n 300) (td 120)))
```

a) Faire exécuter cette procédure. Que fait elle ?

b) au niveau zéro, flocon dessine une courbe connue; quelle est la surface S_0 de cette courbe en fonction de a_0, a_0 étant la longueur d'un côté ?

c) quelle est la surface S_1 de la courbe flocon au niveau 1 en fonction de S_0 puis de a ?

d) de façon générale, que vaut la surface S_i de la courbe flocon au niveau i en fonction de S_{i-1} puis de a ?

Pour faciliter ce calcul, on pourra réfléchir au nombre de traits que l'on a à chaque niveau et à la longueur de chacun de ces traits.

3 Le triangle de PASCAL

Le triangle de PASCAL est obtenu à partir des coefficients du développement du binôme de Newton $(a + b)^n$

n	**$(a + b)^n$**	**coefficients du binôme**				
1	$(a + b)^1 = a + b$	1	1			
2	$(a + b)^2 = a^2 + 2ab + b^2$	1	2	1		
3	$(a + b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$	1	3	3	1	
4	$(a + b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$	1	4	6	4	1

On remarquera que pour un degré n donné, chaque coefficient est obtenu par la somme des deux coefficients précédents du degré inférieur; on se propose d'écrire une fonction qui calcule tous les coefficients pour un degré n donné.

a) écrire la fonction `somme` qui prenant en argument une liste de nombres retourne une liste dont les éléments sont la somme des nombres deux à deux.
exemple : `(somme '(4 5 6))  --> (9 11 6)` -le dernier élément est inchangé !
On s'efforcera d'écrire une version récursive, une version récursive terminale et une version impérative

b) en utilisant la version de la fonction `somme` précédente la plus efficace, écrire la fonction (`pascal n)` qui calcule la liste de tous les coefficients au degré n
exemple : `(pascal 4) --> (1  4  6  4  1)`

c) écrire la procédure `(triangle_pascal n)` qui permet d'afficher le triangle de Pascal jusqu'au degré n

d) la fonction map est une fonction prédéfinie de SCHEME qui s'applique à tous les arguments d'une liste
exemple : `(map  1+  '(1 2 3 4 5 6)) --> (2 3 4 5 6 7)`
Cette fonction a deux arguments : la fonction que l'on veut appliquer et la liste sur laquelle on veut appliquer cette fonction
En supposant que la fonction map n'existe pas, la définir en SCHEME (donner une version récursive puis un version récursive terminale et enfin une version impérative)

e) on se propose d'étudier les coefficients du binôme de PASCAL qui ont une propriété particulière (par exemple ceux qui sont pairs, impairs, divisibles par 5, 7 , 9, etc..)
Ecrire la fonction `(traitement  diviseur n)` qui pour les différents coéfficents du binôme de degré n, retourne 1 quand le coefficient est divisible par diviseur , retourne 0 sinon

exemple: `(traitement  2 4) --> (0  1  1  1 0)`

f) écrire la fonction `(etude_pascal diviseur n)` qui fait la même chose que traitement pour tout le triangle de PASCAL jusqu'au degré n

Quand vous aurez écrit cette fonction, faites une fenêtre de transcript qui occupe tout l'écran et exécuter les fonctions suivantes :

```
(etude_pascal 2 35)
(etude_pascal 3 35)
(etude_pascal 5 35)
(etude_pascal 7 35)
(etude_pascal 9 35)
```

Que remarquez vous ? Quel rapport entre les résultats obtenus et les courbes fractales étudiées précedemment ?

Pour que l'observation soit plus nette, il y a intérêt :
1) à transformer les chiffres en caractère par la fonction ***number->string***
2) à concaténer les caractères par ***string-append***
3) et à faire écrire le résultat de la concaténation avec ***write*** en allant à la ligne par ***newline***