; CORRECTION *TP tortue Partie II*

```
;*******************************
;     VON KOCH
;*******************************
; pour bien positionner la tortue
(define (prepa-von-koch)
  (begin (nettoie) (lc) (fxy -300 -77)(fcap 90) (bc)))
(define (1+ x) (+ x 1))
(define (1- x) (- x 1))
; la procédure de von-koch
(define (von-koch prof cote)
    (if (zero? prof)
        (av cote)
        (begin (let ((cote (/ cote 3))
                     (prof (1- prof)))
                 (von-koch prof cote)
                 (tg 60)
                 (von-koch prof cote)
                 (td 120)
                 (von-koch prof cote)
                 (tg 60)
                 (von-koch prof cote)))))

; pour que ce soit bien positionner faire
; (von-koch prof 600)

; pour faire une animation avec les courbes de vonkoch qui se dessinent
; du niveau de profondeur 0 à n-1

(define (anim-von-koch n) ; ca commence à 0
  (let ((i 0))
  (repeat n
          (prepa-von-koch)
          (write (string-append "La courbe de Von Koch au niveau " (number-
>string i)))
          (newline)
          (msg 200 50 (string-append "La courbe de Von Koch au niveau " (number-
>string i)))
          (von-koch i 600)
           (set! i (1+ i))
           (sleep 2)
           )))

;b) a1 = a0    par construction, quand on passe du niveau i au niveau (i+1),pour
chaque trait dessiné on en supprime un tiers et on en rajoute deux tiers=>
;ai = ai-1=( )i a0 donc ai tend vers l'infini quand i tend vers l'infini !


;*******************************
;           FLOCON
;*******************************
;préparation pour dessiner un flocon

(define (prepa-flocon)
  (begin (nettoie) (lc) (fxy -150 85)(fcap 90) (bc)))

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

; pour faire une animation de flocon de différents niveaux

(define (anim-flocon n) ; ca commence à 0
```

```
(let ((i 0))
(repeat n
        (prepa-flocon)
         (write (string-append "Flocon au niveau " (number->string i)))
        (newline)
        (msg 250 300 (string-append "Flocon au niveau " (number->string i)))
        (flocon i)
         (set! i (1+ i))
         (sleep 2))))
```

Etude de la surface du flocon :

b) au niveau zéro, la courbe est un triangle équilatéral $S_0 = \frac{\sqrt{3}}{4} a^2$

c) au niveau 1, on rajoute 3 triangles équilatéraux de coté $\frac{a}{3}$

donc $S_1 = S_0 + 3 \times \frac{\sqrt{3}}{4} \times (\frac{a}{3})^2 = \frac{\sqrt{3}}{4} a^2 (1 + \frac{1}{3})$

d) quand on passe du niveau i-1 au niveau i, sur chaque trait de longueur a'_{i-1} on construit un nouveau triangle équilatéral de côté $\frac{a'_{i-1}}{3}$

$\Rightarrow S_i = S_{i-1} + 3 \times nb\_trait_{i-1} \times \frac{\sqrt{3}}{4} \times (\frac{a'_{i-1}}{3})^2$ or

niveau	nb_trait	a long global	a' long trait
0	1	a	a
1	4	(4/3)a	a/3
2	16	(16/9)a	a/9
i-1	$(4)^{i-1}$	$(4/3)^{i-1}a$	$a/3^{i-1}$
i	$(4)^{i}$	$(4/3)^{i}a$	$a/3^{i}$

$S_i = S_{i-1} + 3 \times (4)^{i-1} \times \frac{\sqrt{3}}{4} \times (\frac{a}{3^i})^2 = S_{i-1} + \frac{1}{3} \times (\frac{4}{9})^{i-1} \times \frac{\sqrt{3}}{4} a^2$

on remarque que pour i = 1 on retrouve la relation entre S_1 et S_0 d'où

$S_i = S_0 + \frac{1}{3} \times \frac{\sqrt{3}}{4} a^2 \times [1 + \frac{4}{9} + (\frac{4}{9})^2 + ... + (\frac{4}{9})^{i-1}]$

$= \frac{\sqrt{3}}{4} a^2 \times (1 + \frac{1}{3} \times [1 + \frac{4}{9} + (\frac{4}{9})^2 + ... + (\frac{4}{9})^{i-1}])$

or $[1 + \frac{4}{9} + (\frac{4}{9})^2 + ... + (\frac{4}{9})^{i-1}] = \frac{1-(\frac{4}{9})^i}{(1-\frac{4}{9})} \rightarrow \frac{1}{1-\frac{4}{9}} = \frac{9}{5}$ quand i -> ∞

donc $S_i \rightarrow \frac{\sqrt{3}}{4} a^2 \frac{8}{5}$ quand i -> ∞ !!

ce qui veut dire concrètement que la longueur du trait devient infinie, mais la surface comprise à l'intérieur de ce trait elle reste finie !!!

```
; PASCAL and so on

;Outils habituels
(define (1+ x)
  (+ x 1))
(define (1- x)
  (- x 1))
; transforme une liste de nombre à la liste des sommes des nombres avec le
suivant
(define (somme l)
  (if (null? (cdr l))
      l
      (cons (+ (car l) (cadr l)) (somme (cdr l)))))

(define (somme2 l)
  (define (somme_it l res)
    (if (null? (cdr l))
      (append  res l)
      (somme_it (cdr l) (append res (list (+ (car l) (cadr l)))))))
  (somme_it l ()))

; newton et pascal recursif terminaux
(define (newton_rec n)
  ; developpe le binome de Newton pour la puissance n
  (define (bosse i n res)
    (cond ((= i n) res)
          (else (bosse (1+ i) n (cons 1 (somme2 res))))))
  (bosse 1 n '( 1 1)))

(define (triangle_pascal_rec n)
  (define (bosse i n res)
    (cond ((= i n) (write res))
          (else (write res) (newline) (bosse (1+ i) n (cons 1 (somme2 res))))))
  (bosse 1 n '( 1 1)))

; pascal iteratif avec boucle while à revoir
(define (newton_it n)
  (let ((res '()))
    (set! res '(1 1))
    (while (> n 1)
       (set! res (cons 1 (somme2 res)))
       (set! n (1- n)))
    res))

(define (triangle_pascal n)
  (set! res '(1 1))
    (while (> n 1)
       (set! res (cons 1 (somme2 res)))
       (set! n (1- n))
       (write res)
       (newline))
    res)

; les fonctionnnelles MAP
;*************************************
(define (map_perso fonc liste)
  (if (null? liste)
      liste
      (cons (fonc (car liste)) (map fonc (cdr liste)))))

; version recursive terminale
(define (map_perso2 fonc liste)
  (define (travail liste res)
  (if (null? liste)
```

```
      res
      (travail (cdr liste) (append res (list (fonc (car liste)))))))
  (travail liste ()))
; version iterative avec while  a revoir
(define (map_imp fonc liste)
  (set! res ())
  (while (not (null? liste))
         (set! res (append res (list (fonc (car liste)))))
         (set! liste (cdr liste)))
  res)


;  TRAITEMENT
;******************

(define (traitement diviseur n);
  ;met 1 si le coeff dans le binome de newton est divisible par diviseur, 0
sinon
  (map (lambda (x) (if (zero? (remainder x diviseur))  1 0))) (pascal_rec n))

; une version recursive terminale
; affichage de 0 et 1 caractere (char) concaténés pour mieux voir les fractales
(define (etude_pascal_rec diviseur n)
  (define (bosse i n res)
    (cond ((= i (1+ n)) #t)
          (else (write (map (lambda (x) (if (zero? (remainder x diviseur)) 1 0))
res))
                (newline)
                (bosse (1+ i) n (cons 1 (somme2 res))))))
  (bosse 1 n '( 1 1)))
; une version recursive 2 terminale
; affichage de 0 et 1 caractere (char) concaténés pour mieux voir les fractales
(define (etude_pascal_rec2 diviseur n)
  (define (bosse i n res)
    (cond ((= i (1+ n)) #t)
          (else (write (eval (cons string-append
                                   (map (lambda (x) (number->string (if (zero?
(remainder x diviseur)) 1 0))) res))))
                (newline)
                (bosse (1+ i) n (cons 1 (somme2 res))))))
  (bosse 1 n '( 1 1)))
```