(Séquence 4.4

Typage

Typage d'une fonction

Le ***typage*** vérifie la **cohérence** entre

- Les types qui sont précisés dans la spécification
- Ce qui sera calculé par le code de la définition

Cf. équations aux dimensions en physique

Typage d'une fonction mystere001

```
(define (carre x)
  (* x x) )

carre: ???    -> ???
```

```
(define (mystere001 X)
  (if (pair? X)
      (+ (carre (car X)) (mystere001 (cdr X)))
      0 ) )

mystere001: ???              ->  ???
```

Typage d'une fonction mystere001

```
(define (carre x)
  (* x x) )

carre: ???    -> ???
carre: Nombre -> Nombre (du fait de *)
```

```
(define (mystere001 X)
  (if (pair? X)
      (+ (carre (car X)) (mystere001 (cdr X)))
      0 ) )

mystere001: ???              ->  ???
```

Typage d'une fonction mystere001

```
(define (carre x)
  (* x x) )

carre: ???    -> ???
carre: Nombre -> Nombre (du fait de *)
```

```
(define (mystere001 X)
  (if (pair? X)
      (+ (carre (car X)) (mystere001 (cdr X)))
      0 ) )

mystere001: ???              ->  ???
mystere001: ???              -> Nombre (car 0 et +)
```

Typage d'une fonction mystere001

```
(define (carre x)
  (* x x) )

carre: ???    -> ???
carre: Nombre -> Nombre (du fait de *)
```

```
(define (mystere001 X)
  (if (pair? X)
      (+ (carre (car X)) (mystere001 (cdr X)))
      0 ) )

mystere001: ???              ->  ???
mystere001: ???              -> Nombre (car 0 et +)
mystere001: LISTE[???]       -> Nombre (car car et cdr)
```

Typage d'une fonction mystere001

```
(define (carre x)
  (* x x) )

carre: ???    -> ???
carre: Nombre -> Nombre (du fait de *)
```

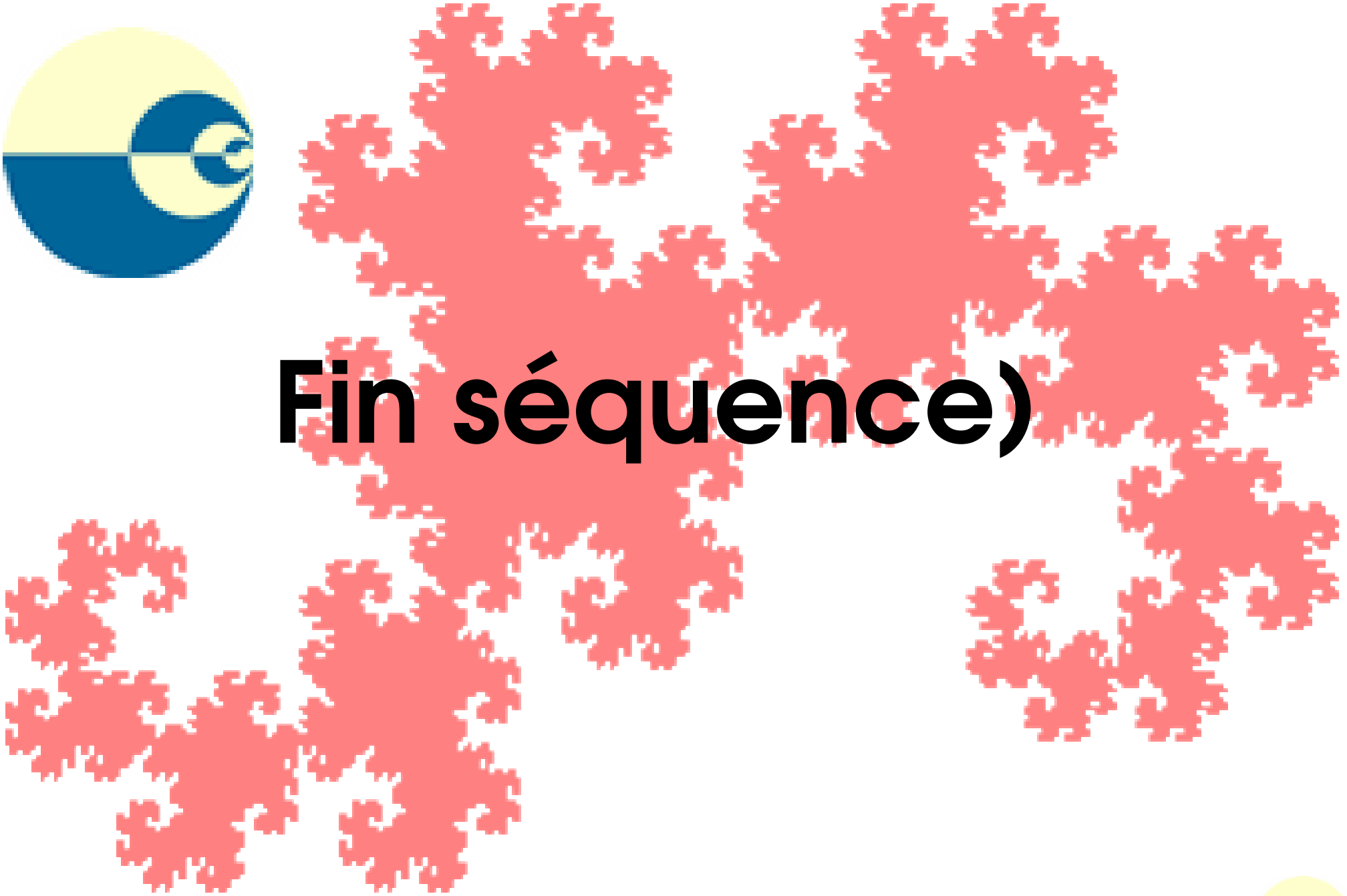
```
(define (mystere001 X)
  (if (pair? X)
      (+ (carre (car X)) (mystere001 (cdr X)))
      0 ) )

mystere001: ???              ->  ???
mystere001: ???              -> Nombre (car 0 et +)
mystere001: LISTE[???]       -> Nombre (car car et cdr)
mystere001: LISTE[Nombre]    -> Nombre (car carre)
```

Somme des carrés d'une liste

```
;;; carre: Nombre -> Nombre
;;; (carre n) rend le carré du nombre n
(define (carre n)
  (* n n) )

;;; somme-carres: LISTE[Nombre] -> Nombre
;;; (somme-carres L) rend la somme des carrés des
;;; éléments de L et rend 0 pour la liste vide
(define (somme-carres L)
  (if (pair? L)
      (+ (carre (car L)) (somme-carres (cdr L)))
      0 ) )
```



Fin séquence)