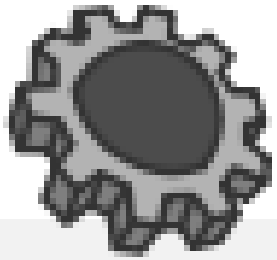
(Séquence 10.3

Processus d'évaluation

Évaluation

```
(define (f ...) ...)
(verifier f
   ... => ... )
(define (g ...) ...)
(verifier g
   ... => ... )
```



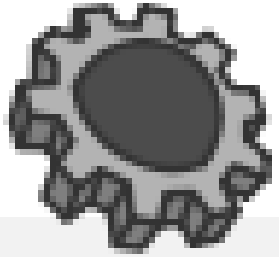
Bouton

```
;;; MrScheme: string -> Valeur
;;; (MrScheme chaîne) calcule la valeur du
;;; programme écrit dans chaîne.
```

Conditionnement

Regroupement en une seule S-expression :

```
(let ()
  (define (f ...) ...)
  (define (g ...) ...)
  (verifier f
     ... => ... )
  (verifier g
     ... => ... ) )
```



Bouton

```
;;; MrScheme: S-expression -> Valeur
;;; (MrScheme sexp) calcule la valeur du
;;; programme représenté par sexp.
```

Auto-évaluation

On nommera cette fonction `valeur`. C'est l'***interprète*** du langage de ce MOOC. Ainsi

```
(valeur '(+ 2 3))   →5
```

Auto-auto-évaluation

La fonction `valeur` est écrite dans le langage même que reconnaît `valeur` :

```
(valeur '(+ 2 3))  →5

(valeur '(let ()
              (define (valeur p) ...)
              (valeur '(+ 2 3)) ))  →5
```

Auto-auto-auto-évaluation

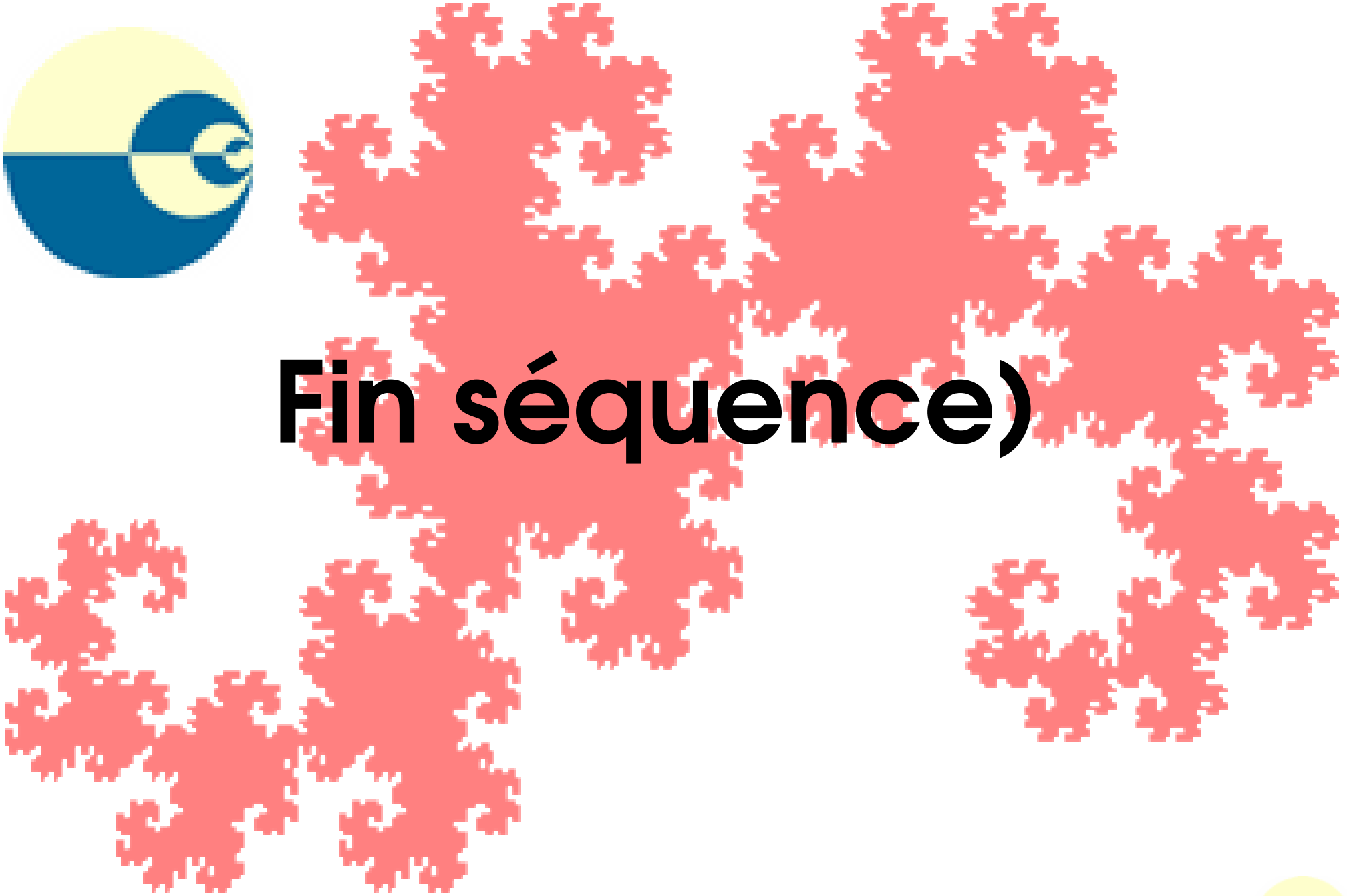
et même :

```
(valeur '(+ 2 3))  →5

(valeur '(let ()
           (define (valeur p) ...)
           (valeur '(+ 2 3)) ))  →5

(valeur
  '(let ()
     (define (valeur p) ...)
     (valeur '(let ()
                (define (valeur p) ...)
                (valeur '(+ 2 3)) )) ))  →5
```

La fonction `valeur`

- la base de tout interprète ou compilateur
- la base de tout metteur au point
- la possibilité d'engendrer des programmes
- une autre façon de comprendre Scheme (nommée `eval`)

Le code correspondant est l'objet de la leçon de cette dernière semaine.



Fin séquence)