(Séquence 6.1

Citation

Listes

En Scheme, les programmes sont représentés par des ***S-expressions*** (ou expressions symboliques).
Les parenthèses `(` ... `)` servent aussi à noter en Scheme :

- des applications fonctionnelles,
- des formes spéciales (`define`, `if`, `and`, `or`, `let`, ...).

Comment faire apparaître une liste dans un programme ?

Expression et valeur d'une expression

Faire la différence entre :

- **expression** : `(list 1 2 3)`
- **valeur** : `(1 2 3)` la liste formée des éléments 1, 2 et 3

Notations : `(list 1 2 3)` $\equiv$ `(cons 1 (list 2 3))`
$\rightarrow$ `(1 2 3)`

Comment exprimer (*citer*) la **valeur** de l'**expression**
`(list 1 2 3)` ?

Syntaxe de la citation

- Syntaxe de la citation

  (**quote** *exp*) **ou** '*exp*

- La citation est une **forme spéciale** : on n'évalue pas le paramètre mais on le retourne tel quel. La citation d'une expression signifie « J'ai pour valeur ce qui suit »

La citation

- la citation d'une constante est la constante elle-même
  `(quote 2)` ≡ `'2` ≡ `2` → `2`
- la citation d'un symbole : `(quote a)` ≡ `'a` → `a`
- la citation d'une liste est la liste des citations de ses éléments

```
(quote (e1 e2 e3)) ≡ '(e1 e2 e3)
                   ≡ (list 'e1 'e2 'e3)
```

```
(quote ()) ≡ '() ≡ (list)
```

Exemples :

```
(cons 'a '()) → (a)
(quote (a1 a2)) ≡ '(a1 a2) → (a1 a2)
(quote (1 2 3)) ≡ '(1 2 3) → (1 2 3)
```

Distinguer variable et symbole

Dans vos programmes Scheme, vous avez utilisé des ***symboles***

- comme mots clef : `if`, `and`, `or`, `let`, `define`
- comme identificateurs : `*`, `map`, `n`, `...`

On peut manipuler les symboles pour eux-mêmes.

Remarques I

```
(let ((x 42))
  ...
  (display (list 'x '= x))    ; Imprime (x = 42)
  ... )
```

- NOTA1 : La « fonction » `display` rend, bien sûr, un résultat mais la norme de Scheme (IEEE 1178-1990) prescrit que ce résultat n'est pas spécifié. En conséquence, il ne faut jamais utiliser ce résultat puisque l'on ignore ce qu'il peut bien être.

Remarques II

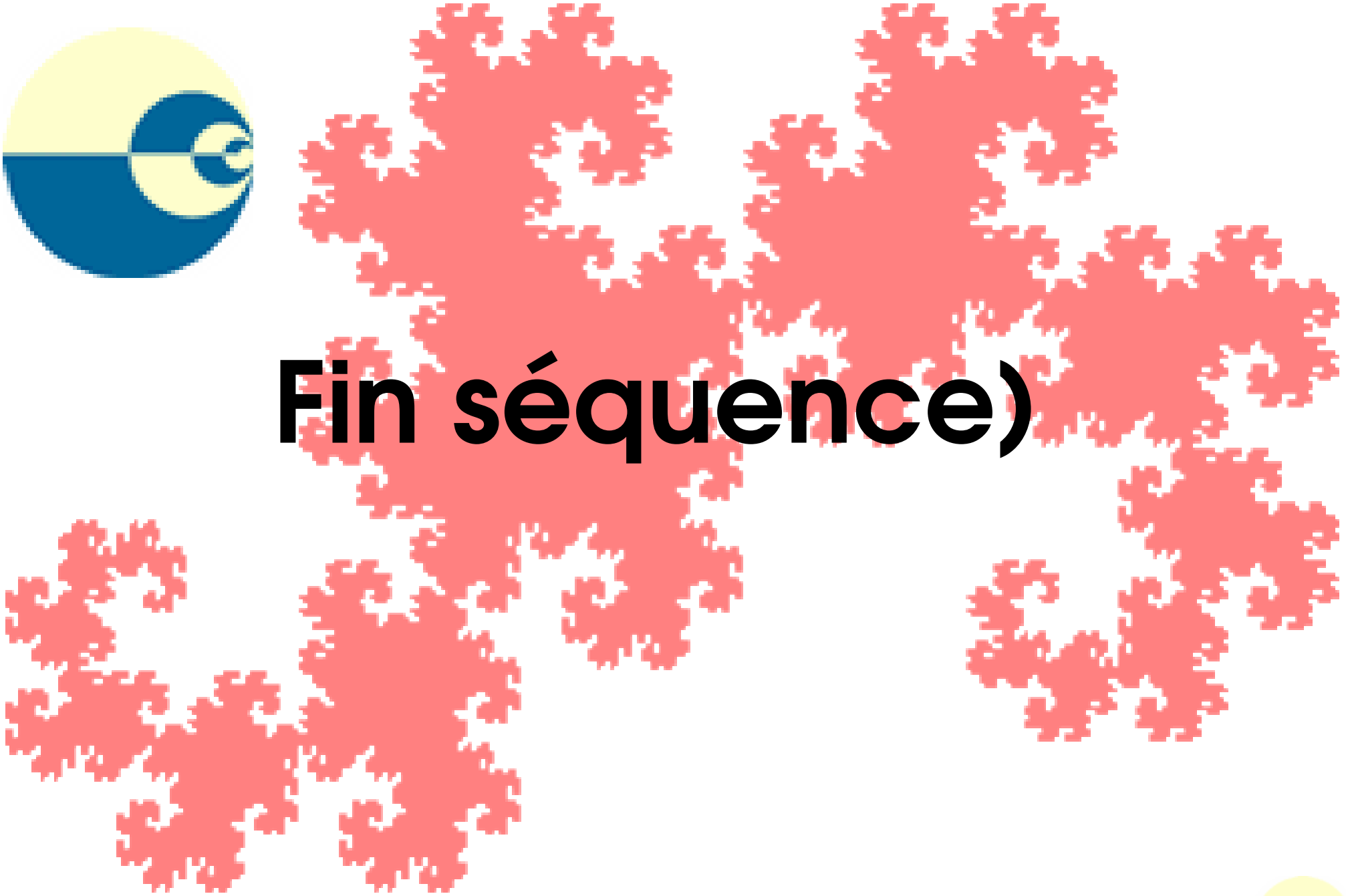
- NOTA2 : La « fonction » `display` imprime à l'écran son argument : c'est, en jargon, un « effet de bord » (traduction littérale de *side-effect* qui est en fait un effet secondaire). Si l'on veut imprimer une valeur (pour mettre au point son programme) et retourner cette valeur, il faut utiliser une séquence dont le mot-clé est `begin` et ainsi écrire :

```
(begin
  (display (list 'x '= x)) ; imprimer la valeur
  x )                      ; retourner la valeur
```

Remarques III

- NOTA3 : Fort heureusement, on a rarement besoin de ce mot-clé car il est implicite dans le corps des fonctions et le corps des blocs locaux. Ainsi peut-on écrire directement :

```
(define (foo x)
  (display (list 'x '= x))
  x )
```



Fin séquence)