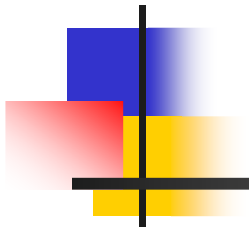
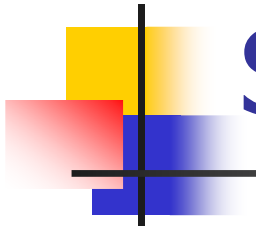


Algorithmique fonctionnelle

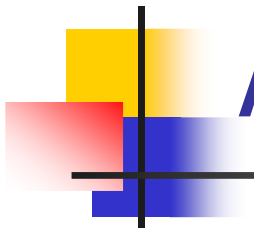
Scheme avancé



Scheme avancé?

Voici quelques notions de Scheme qui sont utiles, voir indispensables pour la réalisation de votre projet.

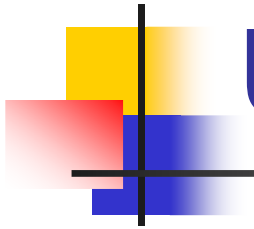
- (display …)
- programme principal
- (let …)
- (set! …)
- (read)
- (begin …)
- (include …)
- (require …)
- création d'un exécutable



Affichage textuel

La commande « display » affiche une chaîne. Il s'agit d'une *procédure* , c'est-à-dire que contrairement à une fonction, il n'y a pas de valeur de retour. Utilisez "\n" pour passer à la ligne.

Programme X
Début
  Ecrire ("a\nbc\nd") (display "a\nbc\nd")
Fin

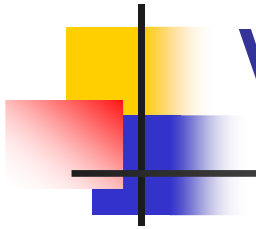


Un programme

Le programme principal est tout simplement *sans mot-clé*. Les instructions sont écrites « en vrac » dans le fichier source.

```
Fonction X (n: Entier): Entier
Début
    Retourner n*n
Fin
Programme Exemple
Début
    Ecrire (X(5))
    Ecrire (X(7))
Fin
```

```
(define (X n)
        (* n n))


(display (number->string (X 5)))
(display (number->string (X 7)))
```



Variables locales

La commande « let » permet de définir des variables locales, c'est-à-dire des variables visibles au sein d'une seule fonction.

```
Fonction A (n: Entier): Entier
Variables
    v, w: Entier
Début
    v ← 5
    w ← n * n
    Retourner v+w
Fin
```

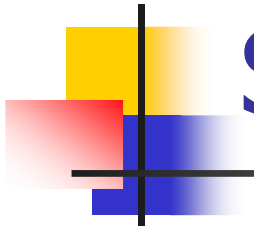
```
(define (A n)
  (let (
        (v 5)
        (w (* n n))
        )
        (+ v w)
  )
)
```

Mise à jour d'une variable

La commande « set! » permet de modifier la valeur d'une variable (globale ou locale) *déjà définie*.

Variables global: **Entier**
Fonction X (n: **Entier**): **Entier**
Variables
    local: **Entier**
Début
    local ← n
    global ← local + 1
    local ← global * global
    Retourner local
Fin

```
(define global 0)

(define (X n)
 (let ((local n))
  (set! global (+ local 1))
  (set! local (* global global))
  local
 )
)
```

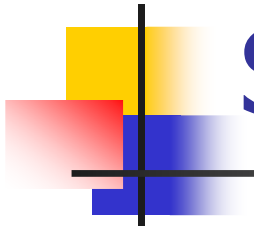


Saisie clavier

La commande « read » permet de demander une saisie clavier à l'utilisateur.

Programme Exemple
Variables
    n: **Entier**
Début
    Ecrire ("Saisir un entier:\n")
    Lire (n)
    Ecrire ("Le double est: ", (2*n))
Fin

```
(display "Saisir un entier:\n")

(let ((n (read)))
  (display (string-append
        "Le double est \n"
        (number->string (* 2 n))
)))
```

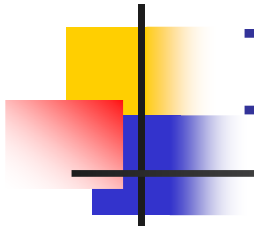


Séquence d'instructions

La commande « begin » permet d'exécuter une séquence d'instructions en Scheme et *renvoie le résultat de la dernière instruction*.

```
Fonction Exemple(x: Réel): Réel
Début
    Si x=0 Alors
        Ecrire ("C'est zéro!")
        Retourner 0
    Sinon
        Retourner 1/x
    FinSi
Fin
```

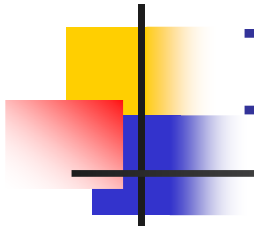
```
(define (exemple x)
  (if (= x 0)
      (begin
        (display "C'est zéro!")
        0    ; renvoie 0
      )
      (/ 1 x)
  )
)
```



Inclusion

La commande « include » permet d'insérer le contenu d'un fichier texte dans un autre. Grâce à ce concept, il est possible de décomposer un grand projet en plusieurs fichiers.

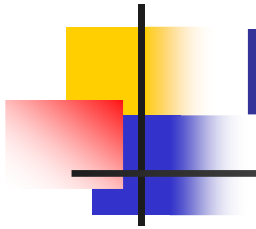
```
(include "toto.ss")
; le fichier toto.ss contient
; la définition de la fonction toto
(define (exemple n)
  (toto 1 2 3 n))
```



Importation

La commande « require » importe des bibliothèques complémentaires de commandes qui ne sont pas disponibles par défaut.

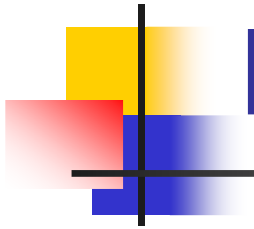
```
(require scheme/date)

(define (exemple)
  (display (string-append
     "Il est"
     (date->string (seconds->date (current-seconds)) #t)
     "\n")))
```



Exécutable

Scheme langage compilé ?

- mode **Lanceur**: Création d'un fichier qui lance un interpréteur Scheme sur le fichier considéré , nécessite le *fichier source* et une *installation DrScheme*
- mode **Autonome**: Création d'un fichier "compilé" qui ne nécessite pas la source mais une *installation DrScheme*
- mode **Distribution**: Création d'un fichier exécutable complètement *indépendant* qui peut être *déployé sur d'autres machines*



Exécutable

Comment créer un exécutable?

- Choisir le langage « Module » (« Pretty big » ne marche pas)
- Ajouter `#lang scheme` (ou `#lang scheme/gui` si vous avez un affichage graphique) en première ligne de votre code
- Cliquer sur « Scheme -> Créer un exécutable » dans le menu de DrScheme
- Choisir le mode souhaité (lanceur, autonome, distribution)
- Choisir l'interpréteur (« MzScheme » pour des programmes purement textuels, « MrEd » pour les programmes graphiques)
- Lancer le programme généré à partir d'un terminal