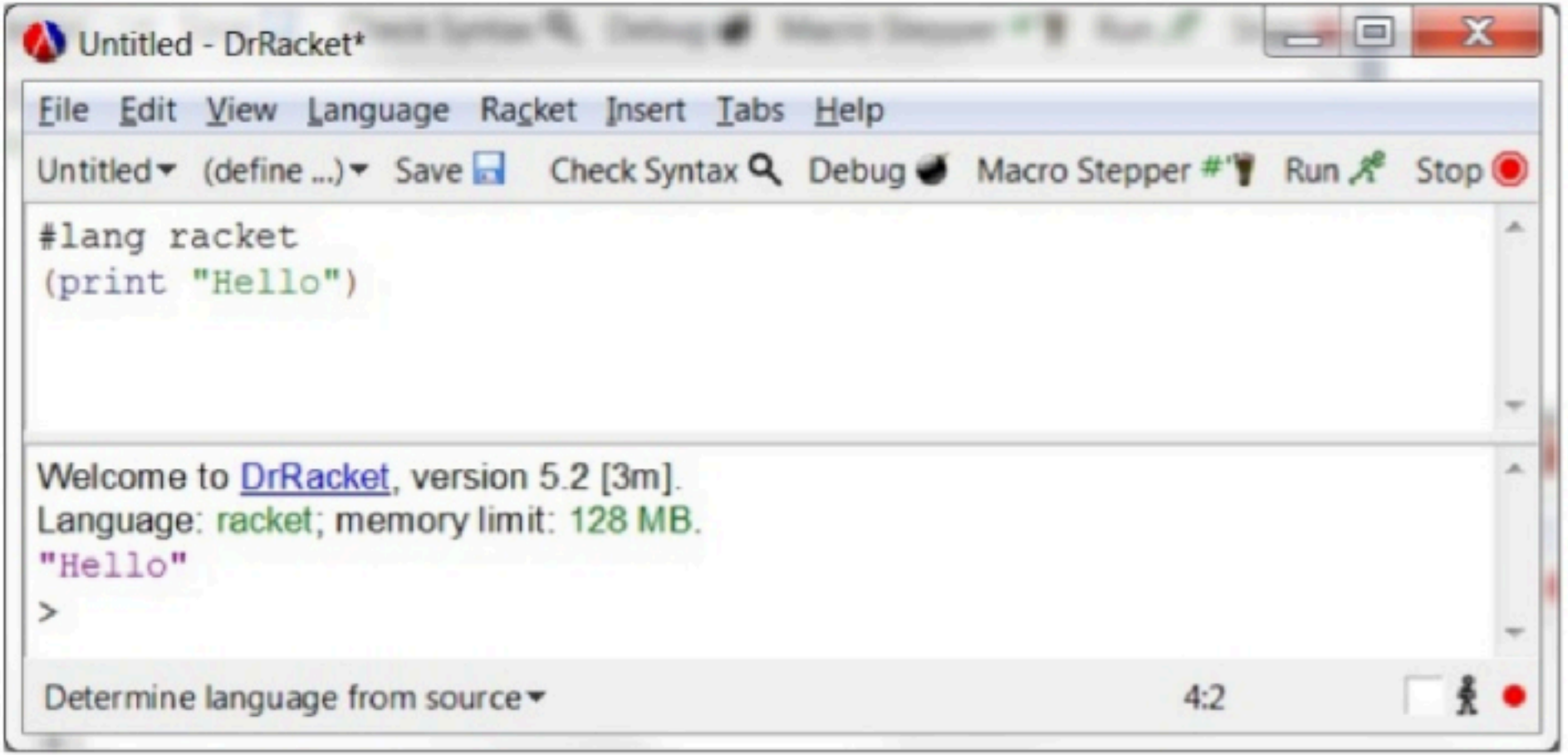

(((Racket)))

Matthias Felleisen

- 1990s PLT Scheme >> Racket functional programming

Environment

- DrRacket IDE is available to all OS platforms



Le langage Scheme

Un langage de programmation
fonctionnelle

Programmation fonctionnelle et Lisp

- Langage concu par John McCarthy entre 1956 - 1959 au MIT pour des applications liées a l'intelligence artificielle (donc l'un des plus vieux langages toujours utilisés)
- LISP = LISt Processor
- Issu de la théorie du λ-calcul (permet aux fonctions d'être les valeurs d'une expression)
- Plusieurs dialectes: Lisp 1.5 (1960), **Scheme** (1975), Common Lisp (1985)…
- Langage riche: fonctionnel, symbolique.
- Syntaxe et sémantique simples et uniformes

9 concepts clé

1. Conditions (if-then-else)
2. Fonctions en tant que type de données
3. Récursivité
4. Variables en tant que pointeurs
5. Ramasse-miette
6. le programme est une expression (non une suite d'énoncés)
7. Les symboles ou atomes
8. L'utilisation des listes et des arbres
9. Langage complet disponible en tout temps (read-eval-print)

Programmation fonctionnelle pure

- Un programme correspond à l'appel d'une fonction
- Une fonction est une composition de fonctions
- Les fonctions ne dépendent que des paramètres transmis
- Pas de variables, pas d'affectations
- Pas de boucles, pas d'énoncé de contrôle (outre la fonction if-then-else)

Programmation fonctionnelle

- Quelques concessions:
  - Permettre la définition locale de certaines valeurs
  - Permettre les affectations (donc les variables à portée lexicale)
  - Permettre l'exécution en séquence (afin de pouvoir morceler le programme).

Programmation fonctionnelle et Scheme

- Dialecte de LISP concu au MIT en 1975, principalement pour l'éducation
- Initialement petit, est maintenant un langage complet.
- Standardisé par ANSI/IEEE, le langage continue à évoluer
- Généralement interprété, il peut aussi être compilé afin d'être efficacement exécuté.

Notions de base

- La liste est la structure de données fondamentale
- Atome: un nombre, une chaine de caractères ou un symbole.
  - Tous les types de données sont égaux
- Expression: un atome ou une liste
- Liste: une série d'expression entre parenthèses
  - Incluant la liste vide () *nil,* à la fois liste et atome
- Une fonction est un objet qui peut être créée, assignée à des variables, passée comme paramètre ou retournée comme valeur.

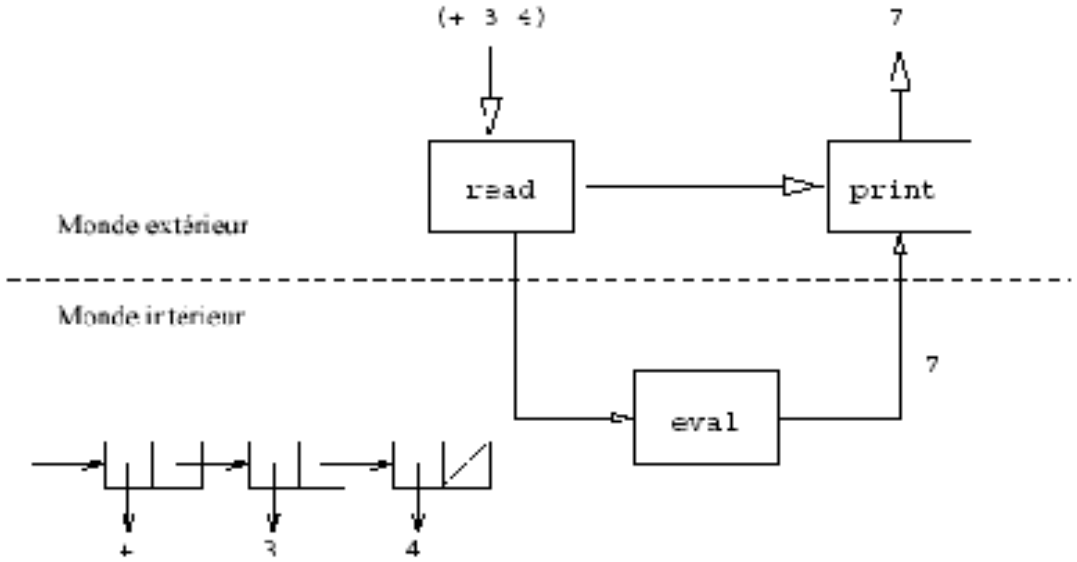
Règles d'évaluation

- Les constantes s'évaluent pour ce qu'elles sont.
- Les identificateurs s'évalue à la valeur qui leur est couramment attribuée.
- Les listes s'évalue en évaluant d'abord la première expression qui la compose;
  - la valeur de cette expression doit être une fonction
  - Les arguments de cette fonction sont les valeurs obtenues par l'évaluation des expressions contenues dans le reste de la liste

Une Session Scheme

- Dans sa forme la plus simple, Scheme utilise le modèle de programmation interactive READ-EVAL-PRINT

```
> (+ 3 4)
   7
> (quit)
```



Évaluation des expressions

- La notation préfixée est utilisée dans l'écriture d'une expression
  - 3+4*5 *devient* (+ 3 (* 4 5))
- Pour évaluer une expression, toutes les sous-expressions doivent être évaluées d' abord. L'évaluation suit donc l'ordre normal de réduction

(+ 3 (* 4 5))
(+ 3 20)
23

Formes syntaxiques spéciales

- Certaines fonctions n' obéissent pas à la règle d'évaluation normale, ces fonctions sont dites de formes syntaxiques spéciales.
- L'évaluation de leurs arguments est plutôt différée jusqu'à ce qu' il soit requis d' en connaitre la valeur.
- Les principales formes spéciales sont:
  1. L'alternative
  2. Le branchement conditionnel
  3. La création de portée locale
  4. La citation

1. L’alternative

```
(if (=  x  0) infini  (/ 1  x))
```

- L' expression qui suit le if est d' abord évaluée, si sa valeur est vraie (#t) alors le second argument est évalué et sa valeur est retournée sans évaluer le troisième argument
- sinon c' est le troisième argument qui est évalué et retourné.

2. Le branchement conditionnel

(cond ((<x xmin) xmin) ((>x xmax) xmax) (#t x))

- La fonction cond est suivie d' une série de listes composée de deux expressions. Si la première des deux expressions d' une de ces listes s'évalue à #t alors la valeur de la seconde expression est retournée
- sinon il faut passer a la liste suivante.
- Si aucune des listes s'évalue à T alors la valeur *nil* est retournée.

Exemple

```
(define (cout age)
   (cond ((or (<= age 3) (>= age 65)) 0)
         ((<= 4 age 6) 0.5)
         ((<= 7 age 12) 1.0)
         ((<= 13 age 15) 1.5)
         ((<= 16 age 18) 1.8)
         (else 2.0)))
```

3. La création de portée locale

```
(let ((pi 3) (d 4)) (* pi d))
12
```

- Le premier argument de cette fonction est une liste de liens créés entre un identificateur et une valeur
- Ces liens ne sont valides que pour l'évaluation de l'expression qui suit (il peut même y en avoir plusieurs afin de permettre l' exécution d' une séquence).