



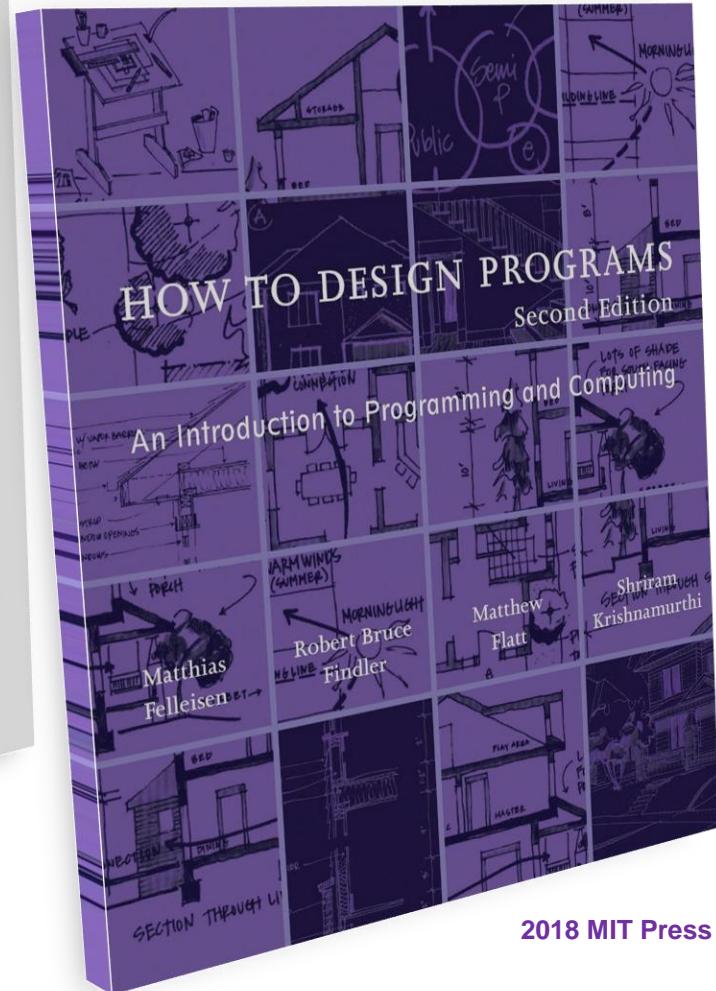
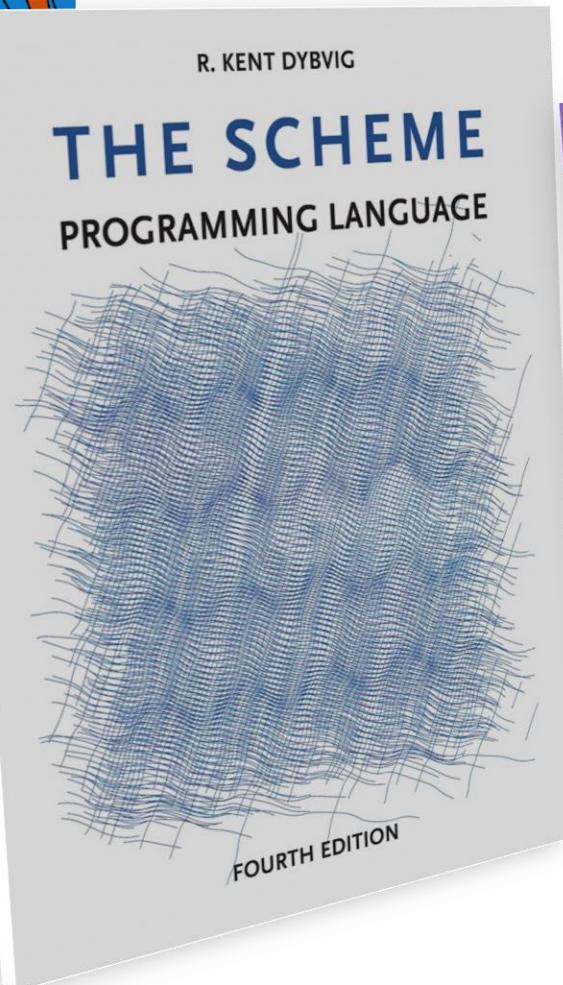
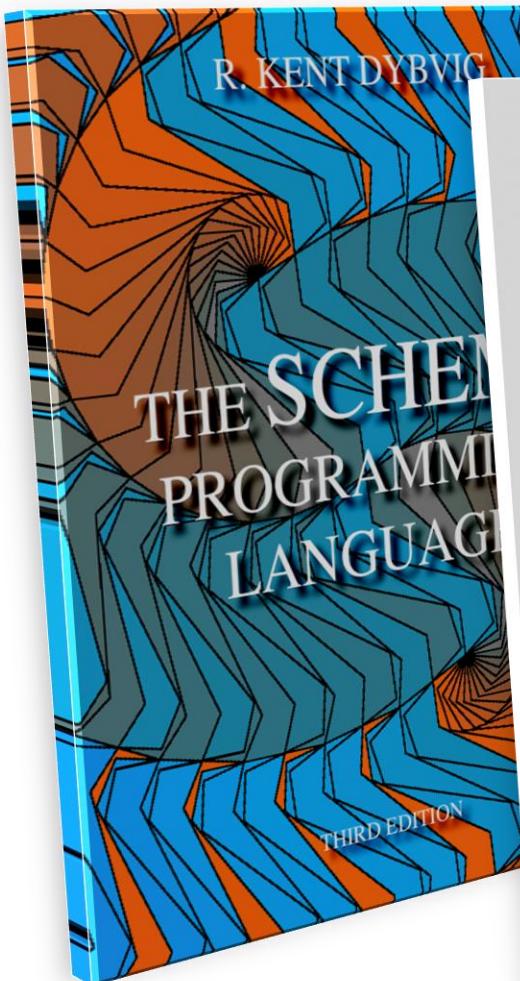
# Introdução à Linguagem Racket

Paradigma Funcional

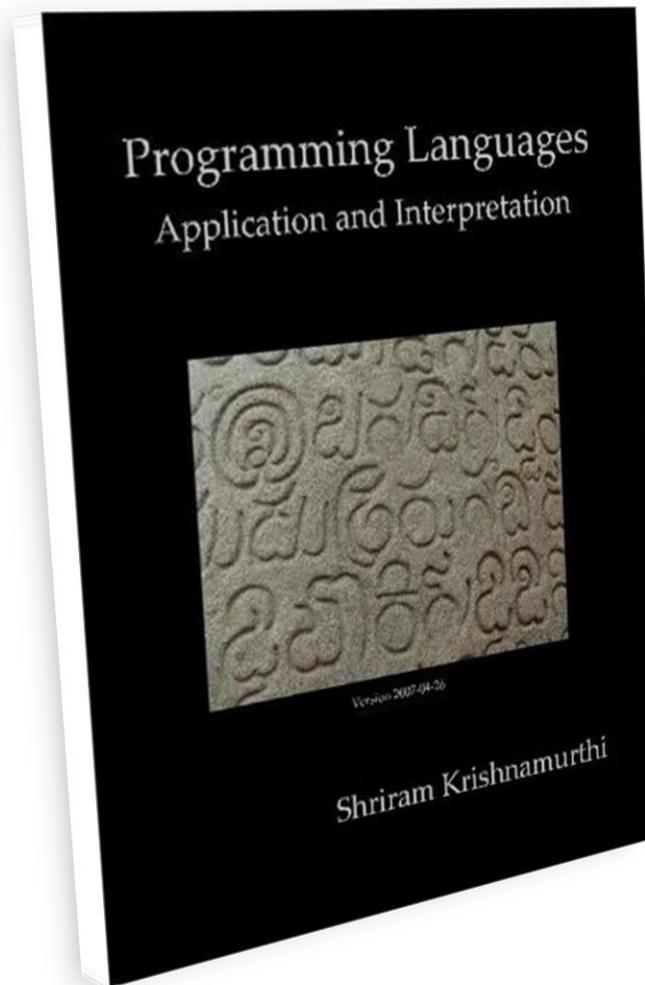


Prof. Ausberto S. Castro V.  
[ascv@uenf.br](mailto:ascv@uenf.br)

# Bibliografia - Textos



# Bibliografia – Texto Complementar



Shriram Krishnamurthi  
Brown University  
2007

# Bibliografia complementar

<https://learnxinyminutes.com/docs/racket/>

# Paradigma Funcional - Origem

- ❖ Programação Funcional (PF)
  - começou em 1960 para dar suporte à pesquisa em IA e computação simbólica
  - Estilo de programação baseada em funções ( $f(x)$ )
- ❖ Primeira linguagem funcional
  - LISP criada por John McCarthy
- ❖ Linguagens sucessoras de Lisp:
  - Lambda Calculus, Scheme, Racket, Miranda, ML, Haskell
- ❖ Linguagens funcionais continuam a serem fortemente utilizadas em aplicações
  - para prova de teoremas,
  - sistemas baseados em regras,
  - processamento de linguagem natural (IA)

# PF: Características

- ❖ Computação é vista como uma função matemática que mapeia entradas para saídas



- ❖ Um programa consiste inteiramente de *funções*
- ❖ *Avaliação de expressões* em vez de *execução de comandos*
- ❖ Transparência referencial
  - O valor de uma expressão depende unicamente dos valores das sub-expressões
- ❖ Gerenciamento de memória automático
  - O sistema deve gerenciar a memória - Requer **garbage collector**
  - Programas mais simples e curtos
  - Execução *mais lenta*

# PF: Características

- ❖ Funções de primeira ordem (**NomeFun paramet**)
  - (tradicional): (defun Somar3 (val) (+ 3 val))
- ❖ Funções de ordem superior
  - funções podem ser parâmetros ou valores de entrada para outras funções
  - funções podem ser os valores de retorno ou saída de uma função
- ❖ Não existe notação explícita de estado
  - Sem declaração de variáveis (estado, valor)
- ❖ Não existe comando de *atribuição* (em linguagens puramente funcionais)
- ❖ Laços (loops) são modelados por *recursão*
- ❖ Baseadas em  $\lambda$ -cálculo

( f a )

# $\lambda$ -cálculo

- ❖ Linguagem precursora de todas as linguagens funcionais
  - Uma função  $f$  definida sobre os números reais, por
$$f(x) = x^2$$
- ❖ Em  $\lambda$ -cálculo:  $f$  é anônima
  - $f = g = h = (\lambda x.x^2)$
  - $f(a) = (\lambda x.x^2)(a) \qquad \qquad f(3) = (\lambda x.x^2)(3)$
  - $f(a) = (f\ a) = ((\lambda x.x^2)\ a)$

$$(f\ a) = ((\lambda x.x^2)\ a)$$

função                  argumento

# Funções (f x)

Função	Notação Funcional
$F(x) = x$	(F x)
$G(x) = x + y$	(+ x y)
$H(y) = x^*y$	(* x y)
$F(x) = \text{sen}(x)$	(sen x)
$K(x) = \cos(x)$	(cos x)
$F(x) = x + 4$	(+ x 4)
$G(y) = 24 - 75$	(- 24 75)
$H(y) = y^2 + 6$	(+ (* y y ) 6)

Notação pré-fixada

(+ a b)

Notação infixa

(a + b)

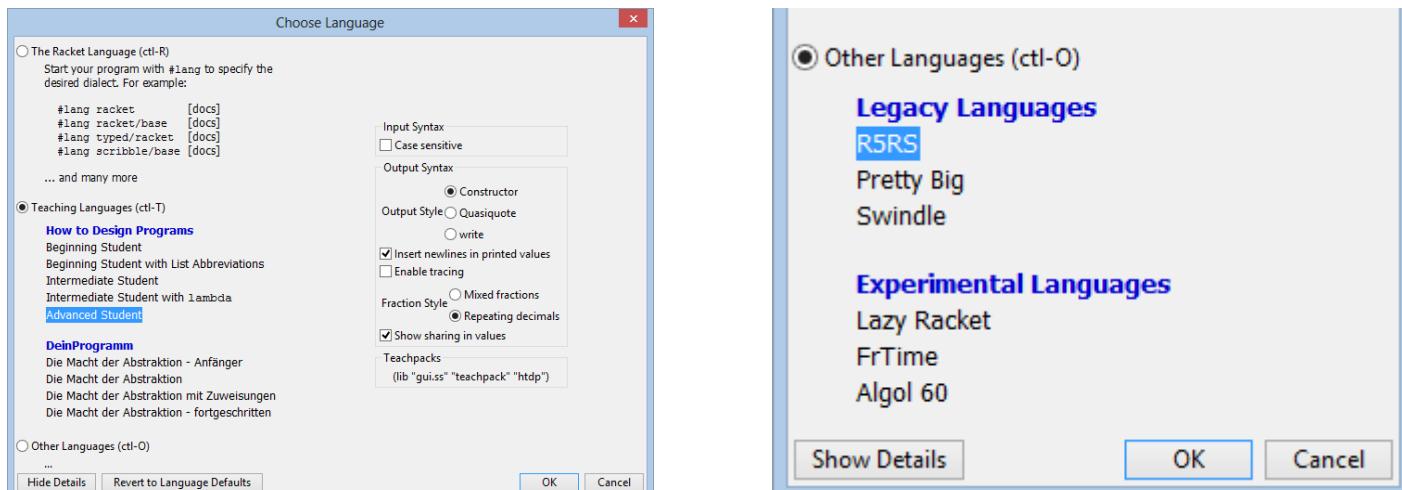
Notação pós-fixada

(a b +)

# Linguagem Racket



- É uma *linguagem de programação* - dialeto de Lisp e descendente de Scheme
- É uma *família de linguagens de programação* – as variantes de Racket
- Um *conjunto de ferramentas*
  - racket (compilador, interpretador e sistema em tempo real)
  - Dr.Racket (ambiente de programação)
  - raco (linha de comandos: instalação de pacotes, construção de bibliotecas)



# Racket

- ❖ Paradigma funcional
- ❖ Linguagem derivada do LISP: é um dialeto
- ❖ Utilizando Scheme e Racket:
  - 159 colleges/universidades nos EUA
  - 286 colleges/universidades ao redor do mundo
  - 65 colégios secundários em EUA
  - 75 colégios secundários ao redor do mundo

Dr Racket, 8.8



# Ferramenta: DrRacket v8.6.0.11



01-primeiro.rkt - DrRacket

File Edit View Language Racket Insert Scripts Tabs Help

01-primeiro.rkt (define ...) ▾

Pause || Go Step Over Out (Out)

```
;; Introdução à Linguagem Racket (Scheme)
;; Prof. Ausberto S. Castro Vera (ascv@uenf.br)
;; UENF-CCT-LCMAT - Curso de Ciencia da Computacao
;; 2021
;; Aluno: Fulano
;;
;; Liguagem Advanced Student
;; O primeiro programa Racket
;;
(begin
  (newline)
  (display "Bom dia, UENF.
  (newline))
```

Welcome to DrRacket, version 8.2 [cs].  
Language: Advanced Student; memory limit: 128 MB.

Bom dia, UENF. Bemvindo à Linguagem Racket-Scheme! 2021

All expressions are covered

Advanced Student ▾

R5RS ▾

Check Syntax Debug Run Stop

01-primeiro.rkt - DrRacket\*

File Edit View Language Racket Insert Scripts Tabs Help

01-primeiro.rkt (define ...) ▾

Macro Stepper

Pause || Go Step Over Out (Out)

```
;; Introdução à Linguagem Racket (Scheme)
;; Prof. Ausberto S. Castro Vera (ascv@uenf.br)
;; UENF-CCT-LCMAT - Curso de Ciencia da Computacao
;; 2021
;; Aluno: Fulano <===== escreva seu nome aqui
;;
;; Liguagem Advanced Student ; define a linguagem default
;; O primeiro programa Racket
;;
(begin
  (newline)
  (display "Bom dia, UENF. Bemvindo à Linguagem Racket-Scheme! 2021") ;<-
  (newline))
```

Welcome to DrRacket, version 8.2 [cs].  
Language: Advanced Student; memory limit: 128 MB.

Bom dia, UENF. Bemvindo à Linguagem Racket-Scheme! 2021

All expressions are covered

OK Cancel

Racket

Version 8.6 (August 2022)

Platform: Windows (x64, 64-bit)

racket-8.6-x86\_64-win32-cs.exe (166M)

- The Racket Language
- Start your desired direction
- #lang
- #lang
- #lang
- #lang
- scribble/base [docs]
- ... and many more

- Teaching Languages (ctl-T)
- How to Design Programs**
- Beginning Student
- Beginning Student with List Abbreviations
- Intermediate Student
- Intermediate Student with Lambda
- Advanced Student**

- DeinProgramm
- Die Macht der Abstraktion - Anfänger
- Die Macht der Abstraktion
- Die Macht der Abstraktion mit Zuweisungen
- Die Macht der Abstraktion - fortgeschritten

- Other Languages (ctl-O)
- Legacy Languages**
- R5RS
- Pretty Big
- Swindle

- Experimental Languages**
- Lazy Racket
- FrTime
- Algol 60

Show Details

# Ferramenta: DrRacket v8.6.0.11

The screenshot shows the DrRacket interface. The top menu bar includes File, Edit, View, Language, Racket, Insert, Scripts, Tabs, and Help. A toolbar below the menu has buttons for Check Syntax, Step, Run, and Stop. The main window title is "A-teste.rkt - DrRacket". The left pane, labeled "Área de Edição (definições)", contains the following Racket code:

```
(+ 6 2)
(* 4 7)
(- 6 1)
(/ 8 2)
(sqrt 3)
(expt 3 4)
(sin 0)
(cos pi)
```

The right pane, labeled "Área de Intereração", shows the output of the code execution:

```
Welcome to DrRacket, version 8.2 [cs].
Language: Beginning Student; memory limit: 128 MB.
8
28
5
4
9
81
0
#i-1.0
>
```

At the bottom, status information includes "All expressions are covered", "Beginning Student", "Show next time?", a timestamp of "9:0", memory usage of "P 573.43 MB", and a count of "0" errors.

# Ferramenta: DrRacket v8.6.0.11

## ❖ HomePage

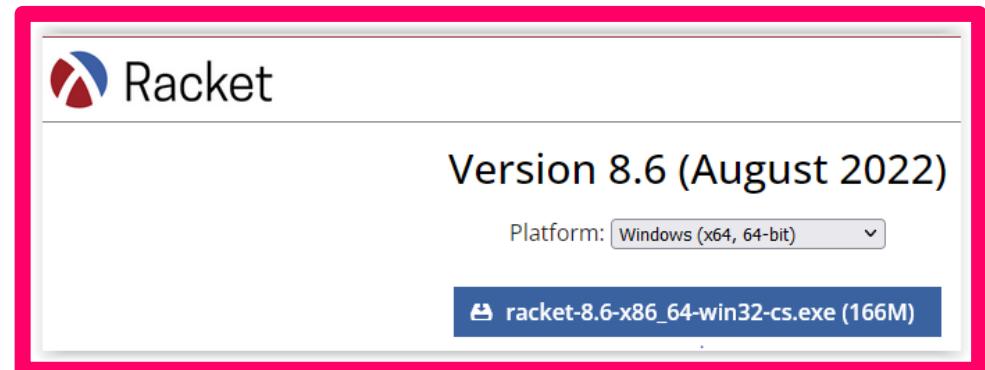
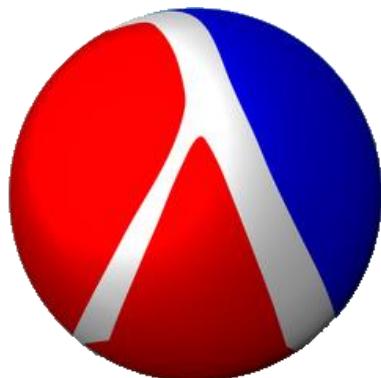
- <http://racket-lang.org/>

## ❖ Quick: An Introduction to Racket with Pictures

- <http://docs.racket-lang.org/quick/index.html>

## ❖ Schemers

- <http://www.schemers.org/>



# Racket-Scheme

- ❖ **Baseada em funções (f x)**
- ❖ **Expressões Primitivas**
  - Números, booleanos, strings, caracteres, símbolos, etc
  - 23, 17.62, “UENF”, a
- ❖ **Formas de Combinação**
  - Baseada em notação pré-fixada
  - (+ 4 7), (\* 5 30)
- ❖ **Formas de Abstração**
  - Variáveis e procedimentos
  - x, y teste,
  - (display “O resultado total = ”)



(Oper Param)

# Primitivas

- ❖ Números
  - 5, 8.6
- ❖ Caracteres
  - ‘a’, ‘x’
- ❖ Strings
  - Utiliza aspas: “esta é uma cadeia de caracteres”
- ❖ Símbolos (um tipo de dado)
  - ‘a, ‘ teste, ‘turma
- ❖ Booleanos
  - #t (verdadeiro) e #f (falso)

# Combinação

- ❖ Utiliza parênteses ao redor de qualquer expressão que deve ser avaliada
    - (soma 3 7), (quadrado x), (area base altura)
  - ❖ Utilizadas na forma pre-fixada

# (nome-função parâmetros)

- (+ 3 7) para 3+7
  - (+ 2 4 6 8 10) para  $2 + 4 + 6 + 8 + 10$
  - (fun 4 9) para executar fun(4,9)

# Abstração

## ❖ Variáveis

- Nomes ou identificadores
- Não precisam ser declaradas
- Letras e símbolos
  - `remove-first`, `zero?`, `mudar!`, `isto+isto`, `+`
  - `maiorque8emenorque15`

## ❖ Procedimentos

- Função pré-definidas (biblioteca)
  - `(+ a, b)`, `(* x y)`,
  - `(list 2 5 8 9)`
- Funções definidas pelo usuário

# Comentários

## ❖ Uma linha

- Utilizando ponto e vírgula
- Exemplo

```
(define pi 3.141516) ; para utilizar com Seno
```

## ❖ Um bloco de linhas

- Utilizando #| no início e |# no final
- Exemplo

```
#|
```

UENF/CCT/LCMAT – C. Computacao, 2022

Prof. Ausberto Castro Vera

```
| #
```

# Definições Globais

- ❖ Funcionam em todo o programa (global)
- ❖ Utiliza o comando **define**
- ❖ Usado para definir variáveis e procedimentos

```
(define <NomeVariavel> <expressão> )
```

```
(define ( <NomeProc> <parâmetros> ) <definição> )
```

# Definindo variáveis

- ❖ Comando **define**
- ❖ Formato

```
(define <NomeVariavel> <expressão> )
```

- ❖ A expressão é avaliada e logo atribuída o resultado à variável

- ❖ Exemplos

- (**define** x 3) ; para x=3
- (**define** z (+ m 6)) ; para z = m+ 6
- (**define** Pi 3.141516) ; para Pi=3.141516

# Definindo variáveis

Bháskara:

$$Ax^2 + Bx + C = 0$$

$$x = \frac{-B \mp \sqrt{B^2 - 4 * A * C}}{2 * A}$$

$$x^2 + 3x - 28 = 0$$

```
(define A 1)
(define B 3)
(define C -28)
(define V1 (* B B) )
(define V2 (* A C) )
(define V3 (* 4 V2) )
(define V4 (- V1 V3) )
(define V5 (sqrt V4) )
(define V6 (* 2 A) )
(define V7a (- (- B) V5) )
(define V7b (+ (- B) V5) )
(define Soluc1 (/ v7a V6) )
(define Soluc2 (/ v7b V6) )
```

The screenshot shows the DrRacket IDE interface with the file "DefVar.rkt" open. The code defines variables A, B, and C, calculates intermediate values V1 through V7, and solves for two solutions using division. The R5RS language is selected at the bottom.

```
; Introdução à Linguagem Scheme-Racket
;; Prof. Ausberto S. Castro Vera           (ascv@uenf.br)
;; UENF-CCT-LCMAT - Curso de Ciencia da Computacao
;; Setembro - 2021
;
; Resolver X^2 + 3X -28 = 0

(define A 1)
(define B 3)
(define C -28)
(define V1 (* B B) )
(define V2 (* A C) )
(define V3 (* 4 V2) )
(define V4 (- V1 V3) )
(define V5 (sqrt V4) )
(define V6 (* 2 A) )
(define V7a (- (- B) V5) )
(define V7b (+ (- B) V5) )
(define Soluc1 (/ V7a V6) )
(define Soluc2 (/ V7b V6) )

Welcome to DrRacket, version 8.2 [cs].
Language: R5RS; memory limit: 128 MB.
-7
4
>
```

# Definindo procedimentos simples

- ❖ Comando **define**
- ❖ Formato

```
(define ( <NomeProc> <parâmetros> ) <definição> )
```

- ❖ Exemplos

- **(define (quadrado x) (\* x x) )** ; para função quadrado
  - **(quadrado 6) → 36**
- **(define (duplo a) (+ a a) )** ; para duplo(a)=2a
  - **(duplo 6) → 12**
- **(define (maiscinco x) (+ x 5) )** ; para maiscinco = x +5
  - **(maiscinco 22) → 27**

# Expressão let

## DEFINIÇÃO

- Uma expressão `let` é utilizada para simplificar uma expressão que deveria conter duas sub-expressões idênticas
- SINTAXE:

```
( let ( (var val) ...) exp1 exp2 ...)  
      |  
      |  
( let <ligação> <corpo> )
```

- Exemplos:

```
( let (( x 2))  
    (+ x 3) )
```

```
( let (( y 3))  
    (+ 2 y) )
```

```
( let ( (x 2)( y 3) )  
    (+ x y) )
```

**(var val)**

O *valor* de uma *variável* vale dentro do *corpo* da expressão `let`

# Expressão let

```
( let ((x 2) (y 3))  
      (* x y) )           => 6
```

```
( let ((x 2) (y 3))  
      (let ((x 7)  
            (z (+ x y)))  
        (* z x)) )           => 35
```

x =7 vale no corpo do segundo let

# Avaliação de Expressões

- ❖ É o coração de qualquer linguagem FUNCIONAL !
- ❖ Expressão:

**(operador operando1 operando2... operandoN )**

- ❖ Algoritmo
  1. Avalia cada uma das sub-expressões
  2. Aplica o resultado mais à esquerda ao resto (**exp resto**)

## □ Exemplo

- $(+ (* 4 5) (/ 48 16)) = (+ 20 3) = 23$
- $(c (\text{quadrado } a) (\text{quadrado } b)) = (c aq qb)$ 
  - $(c (\text{quadrado } 5) (\text{quadrado } 8)) = (c 25 64)$

# Condicional if

## ❖ (if condição consequente alternativa)

### ■ Exemplo

```
(define minquadrado
  (lambda (a b)
    (if (< a b)
        (quadrado a)
        (quadrado b) )))
)
```

```
>(minquadrado 4 5)
16
>(minquadrado 9 7)
49
```

# Expressões Condicionais

## ❖ Sintaxe

```
(cond [<p1> <e1>]  
      [<p2> <e2>]  
      . . .  
      [<pn> <en>] )
```

## ❖ Exemplo

```
(cond ((= x 0) #t)  
      ((< x 0) #f)  
      ((> x 0) #f))
```

```
(cond  
      [(< n 10) 20]  
      [(> n 20) 0]  
      [else 1])
```

```
(cond  
      [(<= n 1000) .040]  
      [(<= n 5000) .045]  
      [(<= n 10000) .055]  
      [(> n 10000) .060])
```

# Expressões Condicionais

```
(define (taxa quantidade)
  (cond
    [(<= quantidade 1000) 0.040]
    [(<= quantidade 5000) 0.045]
    [(> quantidade 5000) 0.050]))
```

```
> (taxa 4000)  
0.045  
> (taxa 6500)  
0.050
```

# Pares

## ❖ Utilizando o operador **cons**

## ❖ Exemplo

> (**cons** 7 9)

(7 . 9)

> (**cons** 23 67)

(23 . 67)

The screenshot shows the DrRacket IDE interface. The top window title is "A-uenf-cons.rkt - DrRacket". The menu bar includes File, Edit, View, Language, Racket, Insert, Scripts, Tabs, and Help. The toolbar includes Check Syntax, Debug, Macro Stepper, Run, and Stop. The code editor contains the following Racket code:

```
;; Introdução a Linguagem Racket-Scheme
;; Prof. Ausberto S. Castro Vera (ascv@computer.org)
;; UENF-CCT-LCMAT - Curso de Ciencia da Computacao
;; Setembro - 2021
;; Aluno: (escreva aqui seu nome)

;; O operador cons
;-----

(cons 7 9)
(cons 23 67)
(cons "UENF-CCT-LCMAT-CC" "Campos,RJ")
(cons "Flamengo Campeao" 2021)
(cons 123.543 321.876)
```

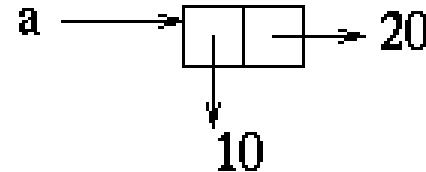
The bottom window displays the R5RS language environment with the following output:

```
Welcome to DrRacket, version 8.2 [cs].
Language: R5RS; memory limit: 128 MB.
(7 . 9)
(23 . 67)
("UENF-CCT-LCMAT-CC" . "Campos,RJ")
("Flamengo Campeao" . 2021)
(123.543 . 321.876)
> |
```

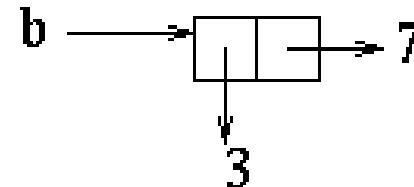
The status bar at the bottom shows "R5RS" and "8:2 P 402.65 MB".

# Pares

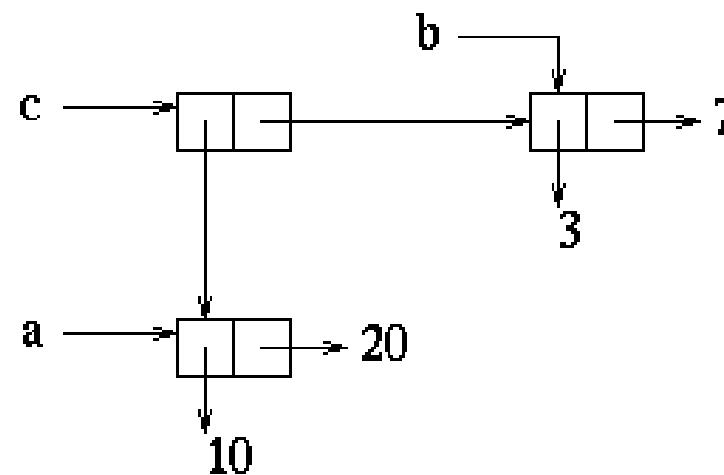
```
(define a (cons 10 20))
```



```
(define b (cons 3 7))
```



```
(define c (cons a b))
```



# Pares : car e cdr

## ❖ Acesso aos elementos de um par

- **car:** elemento da esquerda
- **cdr:** elemento da direita

(**car**    **par**)

(**cdr**    **par**)

The screenshot shows the DrRacket IDE interface with the title bar "A-car-cdr.rkt - DrRacket\*". The menu bar includes File, Edit, View, Language, Racket, Insert, Scripts, Tabs, Help. The toolbar includes Check Syntax, Debug, Macro Stepper, Run, and Stop. There are four tabs open: 1: 04-areas.rkt, 2: teste00.rkt, 3: uenf-cons.rkt, and 4: A-car-cdr.rkt (marked with a star). The code in the editor is:

```
;; Introdução a Linguagem Racket-Scheme
;; Prof. Ausberto S. Castro Vera (ascv@computer.org)
;; UENF-CCT-LCMAT - Curso de Ciencia da Computacao
;; Abril - 2019
;; Aluno: (escreva aqui seu nome)

;; car - cdr
;;

(define x (cons 1 2))

(define y (cons 678 321))

(define w (cons 'a 3))

(define z (cons x y))

(define v (cons z w))
|
(car x)

(car v)

(cdr z)
```

The interaction window at the bottom shows the welcome message and the results of the evaluated expressions:

```
Welcome to DrRacket, version 7.2 [3m].
Language: R5RS; memory limit: 128 MB.
1
((1 . 2) 678 . 321)
(678 . 321)
(a . 3)
>
```

At the bottom right, there are status indicators: R5RS ▾, CRLF 19:0, 603.01 MB, and a small green person icon.

# Listas

- ❖ Uma lista é uma sequência ordenada de elementos
- ❖ Os elementos de uma lista podem ser de qualquer tipo
- ❖ Listas podem ser construídas:
  - A partir de pares
  - Em forma recursiva
    - Uma lista é uma lista vazia
    - Um lista é um par de um elemento junto com uma lista
  - Utilizando o procedimento `list`
- ❖ Exemplo
  - `null` é lista vazia
  - `(define list1 (cons 1 null))`
  - `(define list2 (cons 2 list1))`
  - `(list 3 2 5 7)`

# Funções de Listas

- ❖ **(list item1 item2 ...)**
  - Cria uma lista
- ❖ **(cons item lst)**
  - Retorna o resultado de agregar um novo item ao inicio da lista
- ❖ **(car lst)**
  - Devolve o primeiro elemento da lista
- ❖ **(cdr lst)**
  - Devolve a lista depois do primeiro elemento
- ❖ **(append lst1 lst2)**
  - Concatena duas listas e devolve o resultado

# Funções de Listas

- ❖ **(reverse lst)**
  - **Retorna o reverso da lista lst**
- ❖ **(length lst)**
  - **Devolve o comprimento da lista lst**
- ❖ **(member item lst)**
  - **Testa se um item está na lista**
- ❖ **(null? lst)**
  - **Testa se a lista esta vazia**

# Listas Exemplos

```
> (define ListaNO (list 23 4 78 45 11 62 7 88 2 71))

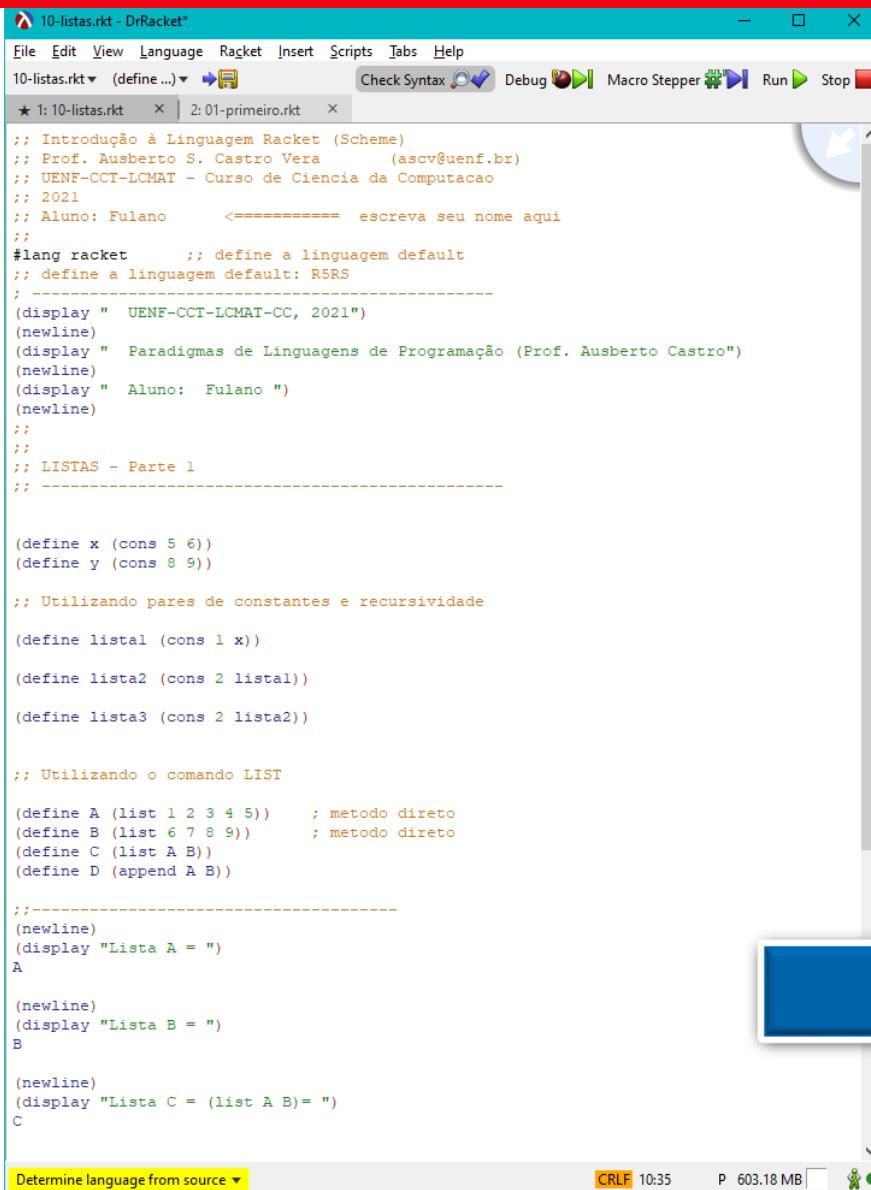
> ListaNO
(list 23 4 78 45 11 62 7 88 2 71)

> (quicksort ListaNO >)
(list 88 78 71 62 45 23 11 7 4 2)

> (quicksort ListaNO <)
(list 2 4 7 11 23 45 62 71 78 88)

>
```

# Listas Exemplos



10-listas.rkt - DrRacket\*

File Edit View Language Racket Insert Scripts Tabs Help

10-listas.rkt (define ...) ▾ Check Syntax Debug Macro Stepper Run Stop

★ 1: 10-listas.rkt x 2: 01-primeiro.rkt x

```
;; Introdução à Linguagem Racket (Scheme)
;; Prof. Ausberto S. Castro Vera (ascv@uenf.br)
;; UENF-CCT-LCMAT - Curso de Ciencia da Computacao
;; 2021
;; Aluno: Fulano <===== escreva seu nome aqui
;;
#lang racket ; define a linguagem default
;; define a linguagem default: R5RS
; -----
(display " UENF-CCT-LCMAT-CC, 2021")
(newline)
(display " Paradigmas de Linguagens de Programação (Prof. Ausberto Castro")
(newline)
(display " Aluno: Fulano ")
(newline)
;;
;;
;; LISTAS - Parte 1
;;

(define x (cons 5 6))
(define y (cons 8 9))

;; Utilizando pares de constantes e recursividade
(define listal (cons 1 x))
(define lista2 (cons 2 listal))
(define lista3 (cons 2 lista2))

;; Utilizando o comando LIST
(define A (list 1 2 3 4 5)) ; metodo direto
(define B (list 6 7 8 9)) ; metodo direto
(define C (list A B))
(define D (append A B))

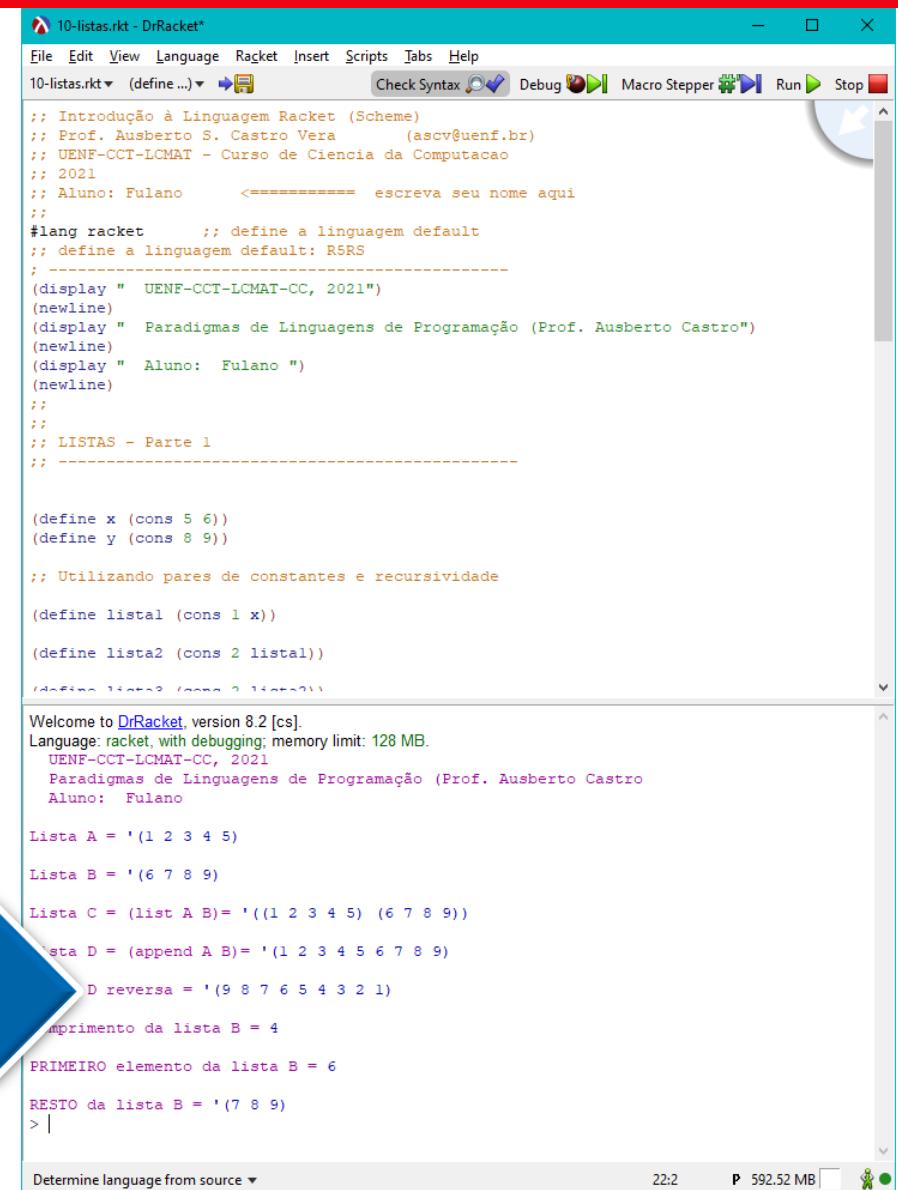
;-----
(newline)
(display "Lista A = ")
A

(newline)
(display "Lista B = ")
B

(newline)
(display "Lista C = (list A B)= ")
C
```

Determine language from source ▾

CRLF 10:35 P 603.18 MB



10-listas.rkt - DrRacket\*

File Edit View Language Racket Insert Scripts Tabs Help

10-listas.rkt (define ...) ▾ Check Syntax Debug Macro Stepper Run Stop

```
; Introdução à Linguagem Racket (Scheme)
; Prof. Ausberto S. Castro Vera (ascv@uenf.br)
; UENF-CCT-LCMAT - Curso de Ciencia da Computacao
; 2021
; Aluno: Fulano <===== escreva seu nome aqui
;;
#lang racket ; define a linguagem default
;; define a linguagem default: R5RS
; -----
(display " UENF-CCT-LCMAT-CC, 2021")
(newline)
(display " Paradigmas de Linguagens de Programação (Prof. Ausberto Castro")
(newline)
(display " Aluno: Fulano ")
(newline)
;;
;;
;; LISTAS - Parte 1
;;

(define x (cons 5 6))
(define y (cons 8 9))

;; Utilizando pares de constantes e recursividade
(define listal (cons 1 x))
(define lista2 (cons 2 listal))
(define lista3 (cons 2 lista2))

Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
UENF-CCT-LCMAT-CC, 2021
Paradigmas de Linguagens de Programação (Prof. Ausberto Castro
Aluno: Fulano

Lista A = '(1 2 3 4 5)
Lista B = '(6 7 8 9)
Lista C = (list A B) = '((1 2 3 4 5) (6 7 8 9))
Lista D = (append A B) = '(1 2 3 4 5 6 7 8 9)
D reversa = '(9 8 7 6 5 4 3 2 1)
imprimento da lista B = 4
PRIMEIRO elemento da lista B = 6
RESTO da lista B = '(7 8 9)
> |
```

Determine language from source ▾

22:2 P 592.52 MB

# Vetores

- ❖ Vectors são listas onde cada elemento tem um índice através do qual pode ser acessado.
- ❖ Para criar vetores utiliza-se o procedimento vector
- ❖ Exemplo:
  - > (vector 6 7 8)
- ❖ (vector-ref vector index)
  - Devolve o item no índice
- ❖ (vector-set! vector index val)
  - Coloca o valor val no índice index
- ❖ Example:

```
(define v (vector 1 2 3))
  ; cria um vetor [1 2 3]
(vector-ref v 0)
  ; devolve v[0]
(vector-set! v 2 35)
  ; v[2] = 35
```

# Comparações

- ❖ Scheme suporta `=, <, >, >=, e >=` para comparar dados numéricos
- ❖ Scheme também tem dois operadores de comparação para qualquer tipo de dado:
  - `(eq? a b)`
    - Testa se duas expressões se referem ao mesmo objeto
  - `(equal? a b)`
    - Compara se dois items tem o mesmo valor

# Procedimentos definidos pelo Usuário

## ❖ Utilizando **define**

- `(define (incremento x) (+ x 1))`
- `(define MercoSul (list "Brasil", "Argentina"  
"Paraguai" "Uruguai"))`

## ❖ Utilizando **lambda**

- Maneira mais elegante e poderosa

`(lambda (lista de parâmetros) corpo )`

## ❖ Exemplos

```
(define incremento  
  (lambda (x) (+ x 1) )  
)  
  
(define quadrado  
  (lambda (x) (* x x) )  
)
```

$(\lambda x.x^2)$

← (incremento 6)

# Procedimentos definidos pelo Usuário

```
(define soma
  (lambda (x y)
    (begin
      (newline)
      (display "A soma = ")
      (+ x y)
    )
  )
)
```

**(texto )**

**(soma 3 8)**

```
(define texto          ; nome do procedimento
  (lambda ()           ; sem argumentos
    (begin
      (newline)
      (display "UENF,2022 - Paradigmas de")
      (display "Linguagens de Programacao")
      (newline)
      (display "Linguagem RACKET")
      (newline)
      (display "Prof. Ausberto Castro")
      (newline)
      (newline)
    )    )  )
```

# Procedimentos definidos pelo Usuário

A-somalista.rkt - DrRacket\*

File Edit View Language Racket Insert Scripts Tabs Help  
A-somalista.rkt (define ...) Run Stop

Check Syntax Debug Run Stop

Salvo neste PC

```
10 (newline)
11 (display " UENF-CCT-LCMAT-CC, 2022")
12 (newline)
13 (display " Paradigmas de Linguagens de Programação (Prof. Ausberto Castro)")
14 (newline)
15 (display " Aluno: Fulano ")
16 (newline)
17 (newline)
18 ;;
19 ; soma os elementos de uma lista, utilizando recursividade
20 (define somalista
21   (lambda (lista)
22     (if (empty? lista)
23       (car (list 0))
24       (+ (car lista) (somalista (cdr lista)))))
25   )
26 )
27
28 (define LISTA01 (list 24 82 13 56 42 102 36))
29
30 (display "Lista1 : ") LISTA01
31 (display "A soma dos elementos da Lista : ") (somalista LISTA01)
```

Language: Advanced Student; memory limit: 128 MB.

UENF-CCT-LCMAT-CC, 2022  
Paradigmas de Linguagens de Programação (Prof. Ausberto Castro)  
Aluno: Fulano

List1 : (list 24 82 13 56 42 102 36)  
A soma dos elementos da Lista : 355  
>

All expressions are covered

Show next time?

Advanced Student

10:2 438.80 MB

# Variáveis locais

- ❖ Criadas utilizando a expressão `let`
- ❖ Formato:

```
(let (( <var> <exp>)  
        (<var> <exp>)  
        ...  
        (<var> <exp>)  
        )  
<corpo>)
```

- ❖ Cada nome de variável `<var>` é limitada à `<exp>` associada
- ❖ Variáveis só existem dentro do escopo de `let`

# Variáveis locais

```
(let ((quadrado  (lambda (x) (* x x)))
      (cubo       (lambda (x) (* x x x)))
      )
  (+ (quadrado 3) (cubo 4)))
```

```
>(define x 10)           x=10
>(+ (let  ( (x 5) )      x=5
      (* x (+ x 2) )    )
     )
     x           x=10
```

45

# Operadores Lógicos

- ❖ **and**

```
(and <exp_1> <exp_2> ... <exp_n>)
```

- ❖ **or**

```
(or <exp_1> <exp_2> ... <exp_n>)
```

- ❖ **not**

```
(not <exp>)
```

- ❖ **Exemplos**

```
(or (< 3 5) (> x 8))
```

```
(and (> x y) (≤ a b))
```

```
(not (equal (+ a b) (- x y)))
```

# Algebra vs Racket vs Pascal

## Algebra

$$f(x) = x + 5$$

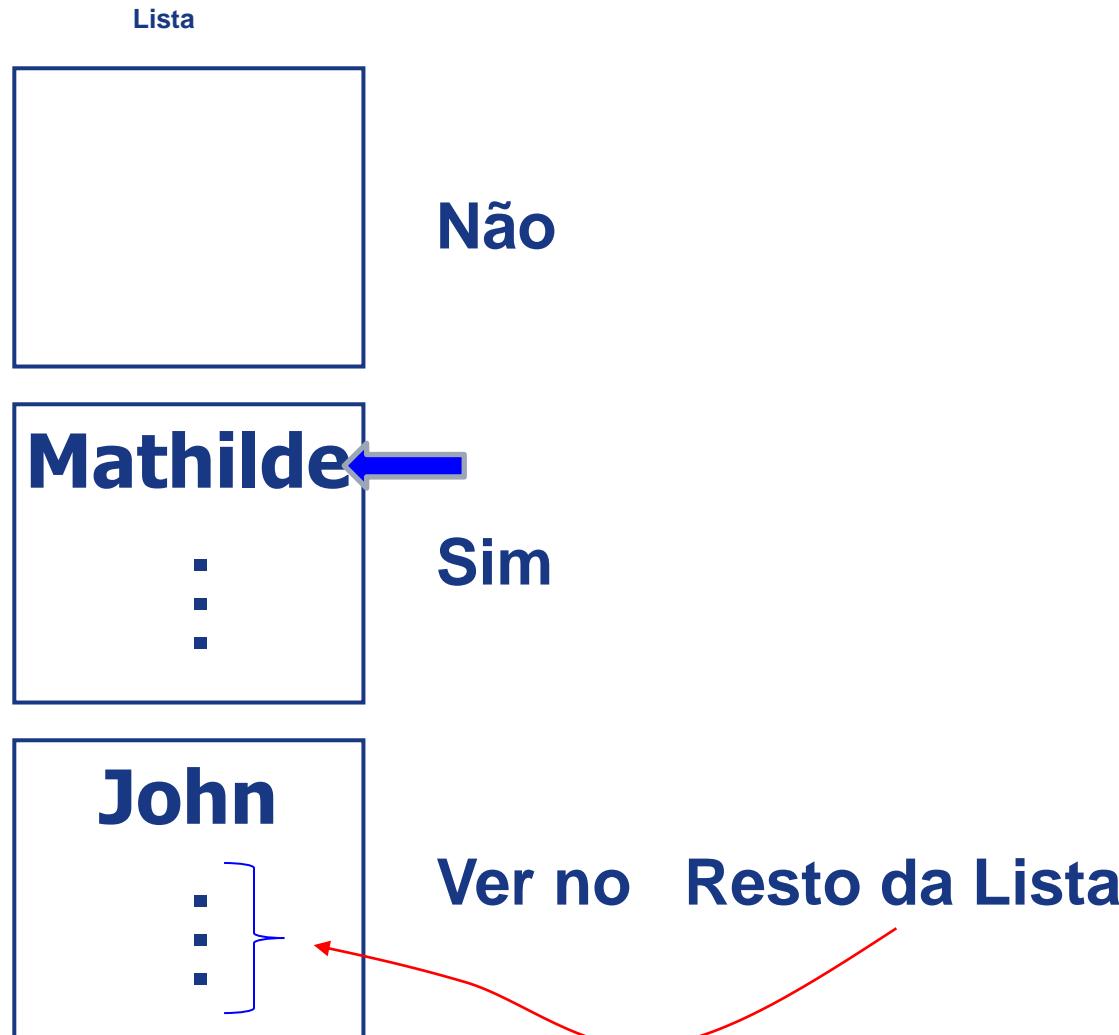
## Racket

```
( define (f x )
  ( + x 5 ) )
```

## Pascal

```
Program (Input, Output) ;
Var
  x : Integer ;
Begin
  Readln ( x ) ;
  Writeln ( x + 5 ) ;
End .
```

# Esta *Mathilde* na lista?



# Esta *Mathilde* no Resto da Lista?

Resto da Lista

Não

Mathilde

Sim

Elize

Ver no Resto da Lista



# Algebra vs Racket (Scheme)

*guest( name, list ) =*

{	<i>no</i>	<i>if list is empty</i>
	<i>yes</i>	<i>if name = first ( list )</i>
	<i>guest( name, rest ( list ) )</i>	<i>otherwise</i>

```
( define ( guest  name  list )
  ( cond
    ( ( empty?  list )           'no
    ( ( equal?  name ( first  list ))   'yes
    ( else                         ( guest name (rest list ) ) ) ) )
```

## Racket

```
(define (guest name list)
  (cond
    ((empty? list) 'no)
    ((equal? name (first list)) 'yes)
    (else (guest name (rest list))))
```

```
#include <stdio.h>
typedef struct listCell * list;
struct listCell {
    int first;
    list rest;
};
bool guest (int x, list l) {
    if (l == NULL)
        return false;
    else if (x == (l -> first))
        return true;
    else
        return guest (x, l -> rest);
}
int main (int argc, char ** argv) {
    list l1, l2, l3 = NULL; int x;
    l1 = (list) malloc (sizeof (struct listCell));
    l2 = (list) malloc (sizeof (struct listCell));
    l2 -> first = 3; l2 -> rest = l3;
    l1 -> first = 2; l1 -> rest = l2;
    scanf ("%d", &x);
    printf ("%d\n", member (x, l1));
}
```

# Scheme

```
( define ( guest name list )
  ( cond
    ( ( empty? list ) 'no )
    ( ( equal? name ( first list ) ) 'yes )
    ( else ( guest name ( rest list ) ) ) ) )
```

# Pascal

```
Program NameOnList (Input, Output);
Type
  ListType = ^NodeType;
  NodeType = Record
    First : String;
    Rest : ListType
  End;
Var
  List : ListType;
  Name : String;
Procedure GetList (Var List: ListType); ...
Function Member (Name : String; List : ListType) : String;
Begin
  If List = nil
    Then Member := 'no'
  Else If Name = List^.First
    Then Member := 'yes'
  Else Member := Member ( Name, List^.Rest)
End;
Begin
  ReadIn ( Name );
  GetList ( List );
  Writeln (Member ( Name, List ) )
End .
```

# Exercícios (Laboratório)

## ❖ Programar em Racket:

- $(3 + x)/(7y - 2) - (xy + 9)$
- Raiz quadrada de  $x^2 + 3x - 5$
- Criar uma lista com 5 elementos
- Determinar o segundo elemento de uma lista
- Determinar o antepenúltimo elemento de uma lista dada
- Consultar se um elemento pertence a uma lista dada
- Adicionar o terceiro elemento de uma lista
  - No final de outra lista A
  - No início de outra lista B
- Calcular o perímetro de um quadrado, circulo ou triangulo
- Calcular o k-ésimo numero inteiro (par ou ímpar)

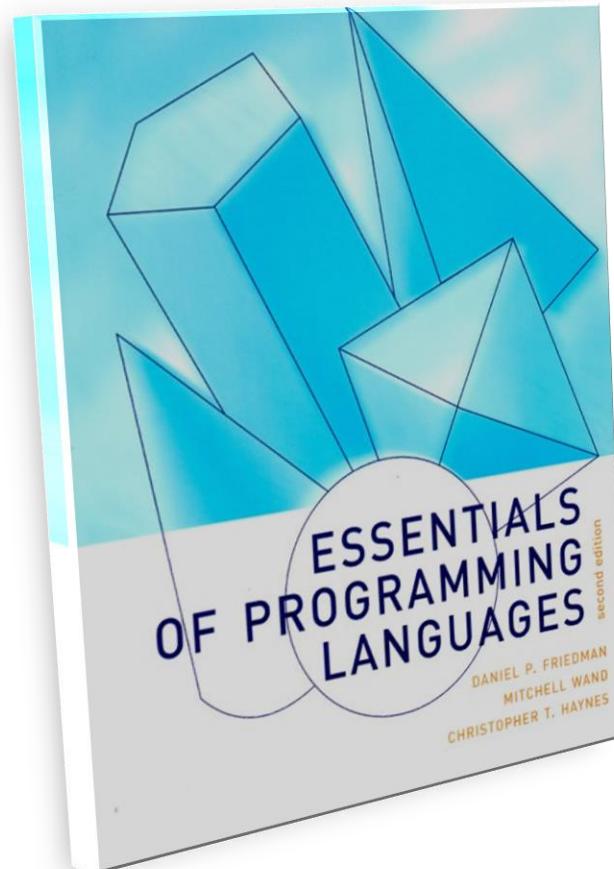
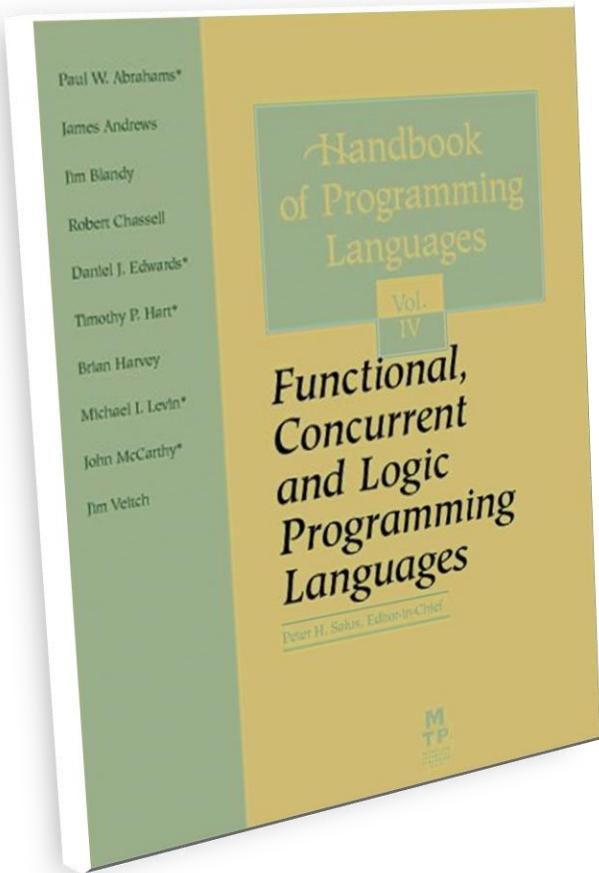


**Prof. Dr. Ausberto S. Castro Vera**  
**Ciência da Computação**  
**UENF-CCT-LCMAT**  
**Campos, RJ**

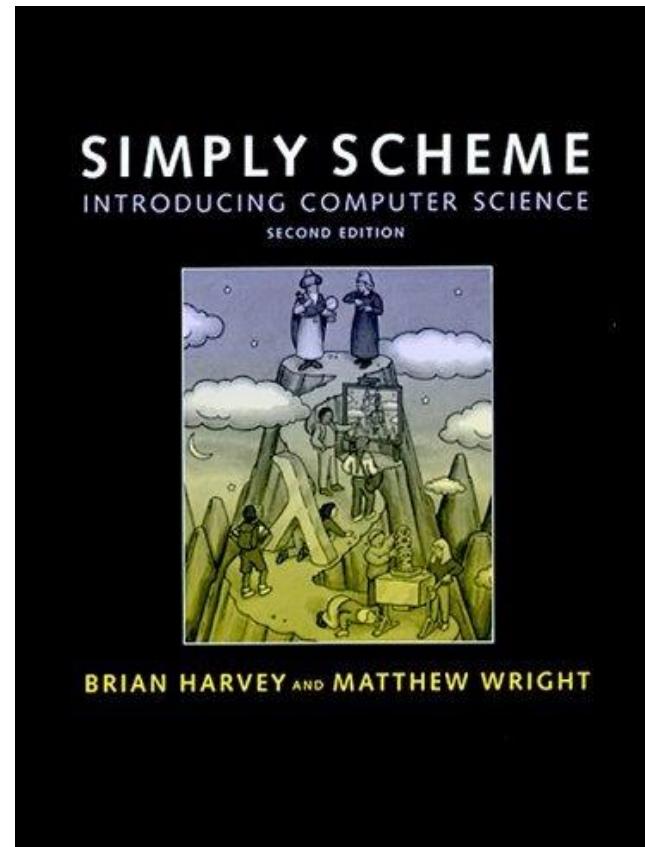
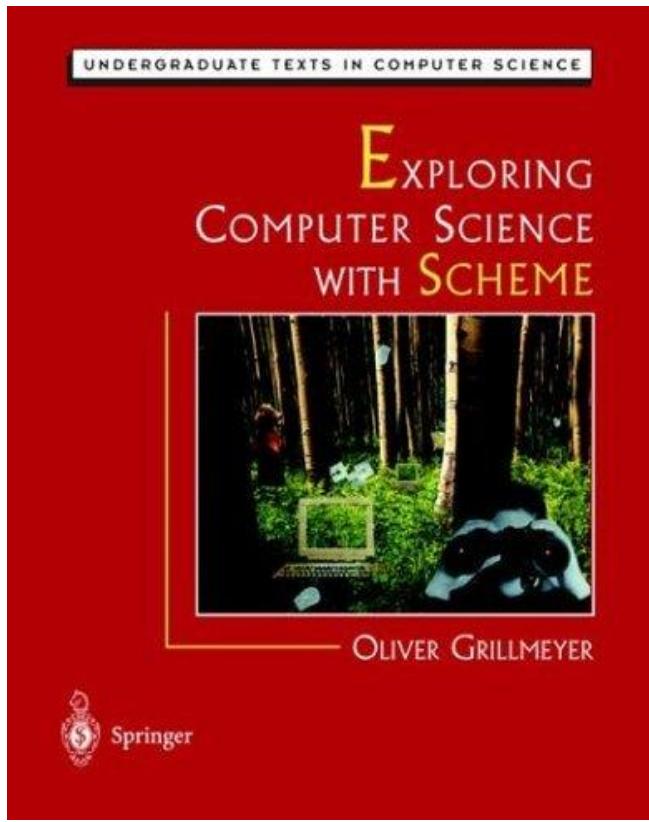
[ascv@uenf.br](mailto:ascv@uenf.br)  
[ausberto.castro@gmail.com](mailto:ausberto.castro@gmail.com)

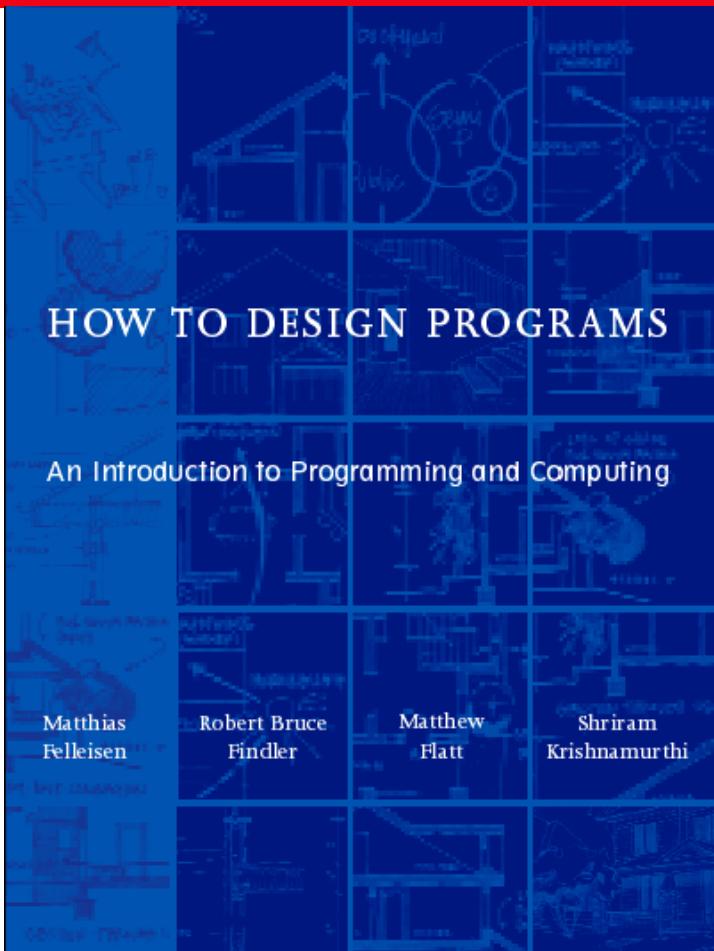


# Bibliografia Complementar



# Bibliografia Complementar





**How to Design Programs  
Second Edition, 2012**

▼ [How to Design  
Programs, Second  
Edition](#)

- 1 Prologue: How to Program
- 2 Fixed-Size Data
- 3 Intermezzo: BSL
- 4 Arbitrarily Large Data
- 5 Intermezzo: Quote, Unquote
- 6 Abstraction
- 7 Intermezzo: Scope
- 8 Intertwined Data
- 9 Intermezzo: Pattern Matching
- 10 Generative Recursion
- 11 Intermezzo: Vectors
- 12 Accumulators
- 13 Epilogue

*On this page:*

- 1.1 Arithmetic and Arithmetic
- 1.2 Inputs and Output
- 1.3 Many Ways to Compute
- 1.4 One Program, Many Definitions
  - 1.4.1 Magic Numbers
  - 1.5 One More Definition
- 1.6 You are a Programmer Now
- 1.7 Not!

*On this page:*

- 2.1 Arithmetic
  - 2.1.1 The Arithmetic of Numbers
  - 2.1.2 The Arithmetic of Strings
  - 2.1.3 Mixing It Up
  - 2.1.4 The Arithmetic of Images
  - 2.1.5 The Arithmetic of Booleans
  - 2.1.6 Mixing It Up with Booleans
  - 2.1.7 Predicates: Know Thy Data
- 2.2 Functions and Programs
  - 2.2.1 Functions
  - 2.2.2 Composing Functions
  - 2.2.3 Programs
- 2.3 How to Design Programs
  - 2.3.1 Designing Functions
  - 2.3.2 Finger Exercises
  - 2.3.3 Domain Knowledge
  - 2.3.4 From Functions to Programs
  - 2.3.5 On Testing
  - 2.3.6 Designing World Programs
  - 2.3.7 Virtual Pet Worlds
- 2.4 Intervals, Enumerations, etc.
  - 2.4.1 Conditional Computations
  - 2.4.2 How It Works
  - 2.4.3 Enumerations
  - 2.4.4 Intervals
  - 2.4.5 Itemizations
  - 2.4.6 Designing with Itemizations

# Bibliografia Complementar

