

## SUMÁRIO - AULA 13

- ❑ Paradigma Funcional

- ❑ 2 aulas



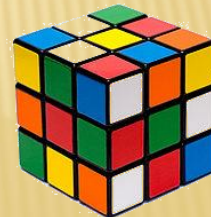
## PARADIGMAS DE LPS

- ❑ Paradigma Imperativo / Procedural
- ❑ Paradigma Estruturado
- ❑ Paradigma Orientado a Objeto
- ❑ Paradigma Orientado a Aspectos
- ❑ Paradigma Funcional
- ❑ Paradigma Lógico



## PARADIGMA FUNCIONAL

- ❑ Trata a computação como uma **avaliação** de **funções matemáticas**.
  - ❑  $f(x) = 2x + 4$
- ❑ Enxerga **todos** os **subprogramas** como **funções** que recebem argumentos e retornam **soluções simples**



## PARADIGMA FUNCIONAL

- ❑ Cada **porção** de um programa pode ser compilada, corrigida e testada **independentemente** das restantes.



- ❑ Permite a criação, teste e correção de programas de uma forma **incremental**, o que **facilita** muito a tarefa do programador.

## PARADIGMA FUNCIONAL

- Tomou forma via linguagem **LISP**

- LIS**t Processor

- possui váaaarios **dialetos**

- nem sempre compatíveis...



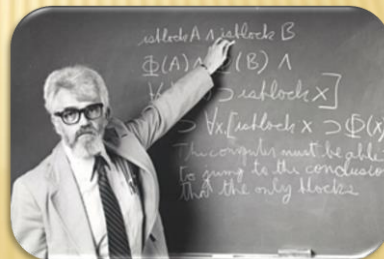
- LISP é frequentemente implementada por um **interpretador**, onde o **usuário** entra com uma **expressão** e o interpretador **avalia** a expressão e imprime o resultado.

## DO LISP AO RACKET: HISTÓRIA

- Lisp foi desenvolvido no final dos anos **50** por **John McCarthy**.

- É a segunda **mais antiga** linguagem de programação de alto nível.

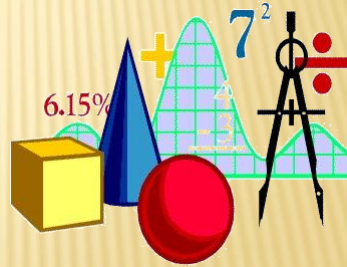
- A mais antiga é FORTRAN.



## DO LISP AO RACKET: HISTÓRIA

- Lisp nasceu como uma **ferramenta matemática**, independente de qualquer computador.

- Cálculo Lambda
  - Alonzo Church
  - matemático



- só depois foi adaptado a uma máquina

## PARADIGMA FUNCIONAL

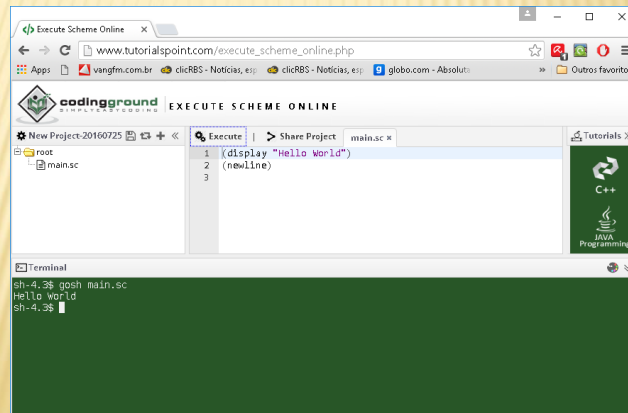
- LISP - *Lost In Stupid Parenthesis*

```
{defun repl-fun (noprint).
  (/show0 "entering REPL").
  (loop.
    (unwind-protect.
      (progn.
        (scrub-control-stack).
        (sb!thread::get-foreground).
        (unless noprint.
          (flush-standard-output-streams).
          (funcall *repl-prompt-fun* *standard-output*).
          (force-output *standard-output*).
          (let* ((form (funcall *repl-read-form-fun*.
                               *standard-input*.
                               *standard-output*)))
            (results (multiple-value-list (interactive-eval form))))).
        (unless noprint.
          (dolist (result results).
            (fresh-line).
            (princ result))))).
    (disable-stepping)))).
```



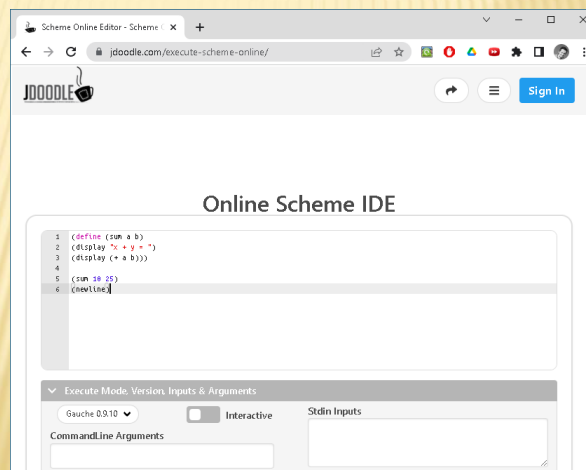
## AMBIENTE ON LINE

- ❑ [http://www.tutorialspoint.com/execute\\_scheme\\_online.php](http://www.tutorialspoint.com/execute_scheme_online.php)



## AMBIENTE ON LINE

- ❑ <https://www.jdoodle.com/execute-scheme-online>



## RACKET

- ❑ Linguagem Funcional que iremos utilizar.



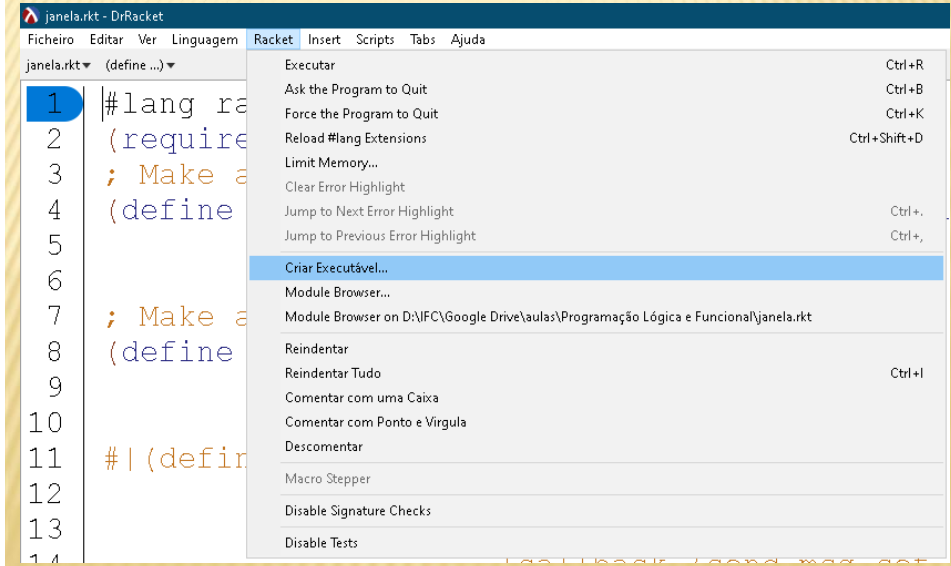
## RACKET: CARACTERÍSTICAS



- ❑ Pode ser executado de forma **interpretada** ou **compilada**.
- ❑ Pode-se gerar **executável**.
  - ❑ <https://docs.racket-lang.org/guide/running.html>
- ❑ Baixar em <http://racket-lang.org>

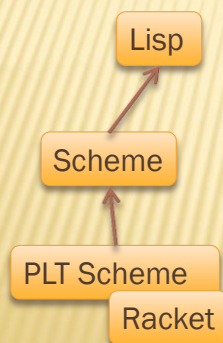


# RACKET: CARACTERÍSTICAS

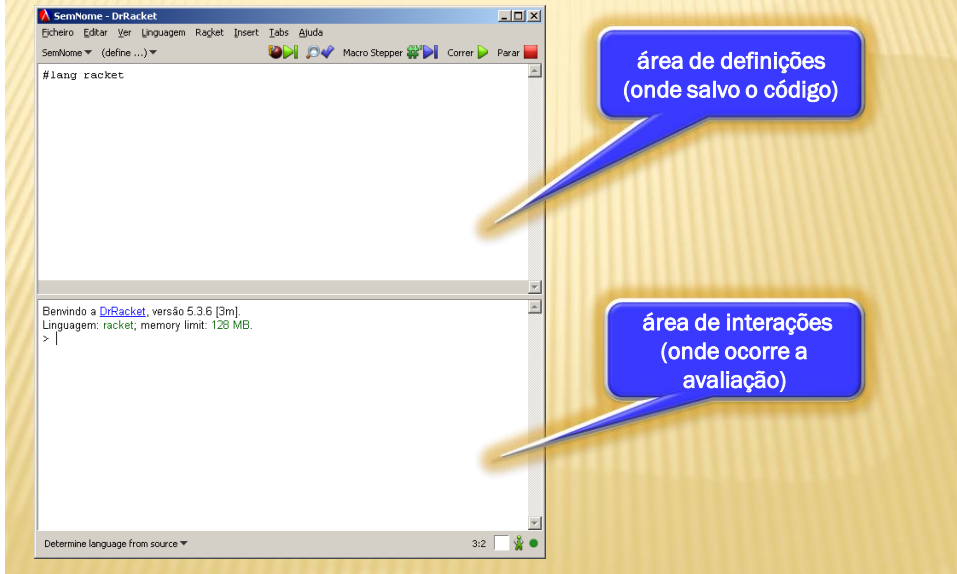


## DO LISP AO RACKET: HISTÓRIA

- ❑ Racket chamava-se PLT Scheme
  - ❑ Dialetto de Scheme
    - ❑ Linguagem da família Lisp



## Ambiente DrRacket

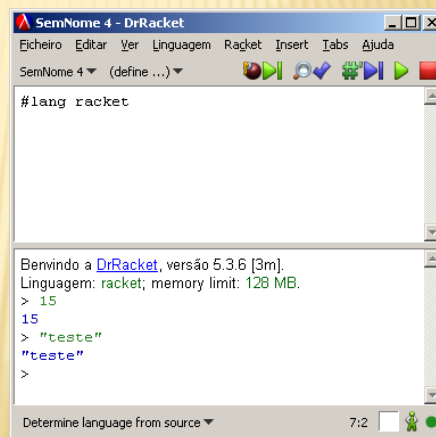


## RACKET: CARACTERÍSTICAS

❑ Avaliação instantânea de expressões. Ex:

❑ 15

❑ "teste"





## ÁLGEBRA X RACKET

Álgebra	Racket
$2+3$	$(+ 2 3)$
$3*4$	$(* 3 4)$
$(2 + 3) * (4 + 5)$	$(* (+ 2 3) (+ 4 5))$
$2 * 3 + 4 * 5$	$(+ (* 2 3) (* 4 5))$
$53 + 48$	$(+ 53 48)$

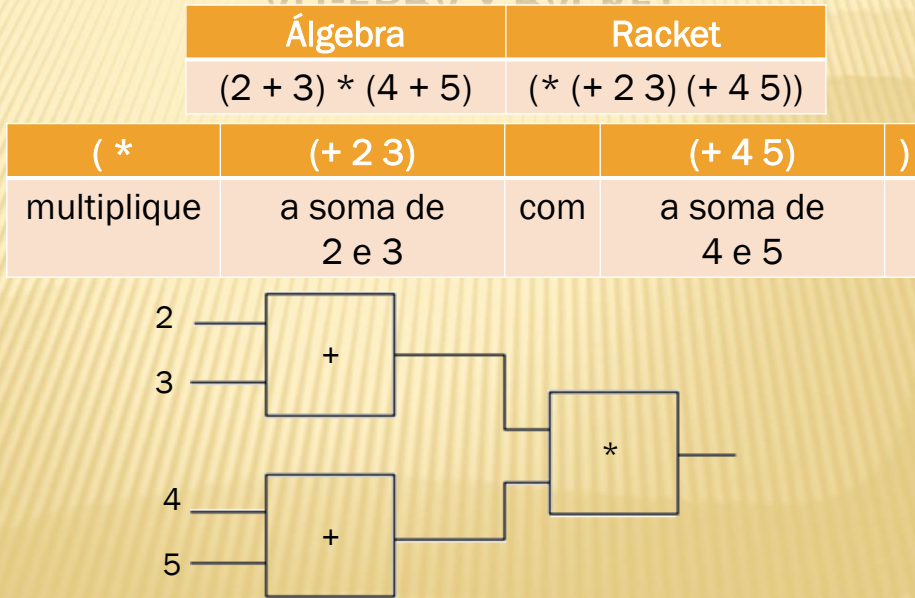
- ❑ Em Racket:
  - ❑ cada operação é colocada entre **parênteses**
  - ❑ o **operador** vem sempre **antes** dos operandos:
    - ❑ Notação pré-fixada (**prefix**)

## INFIX PREFIX POSTFIX

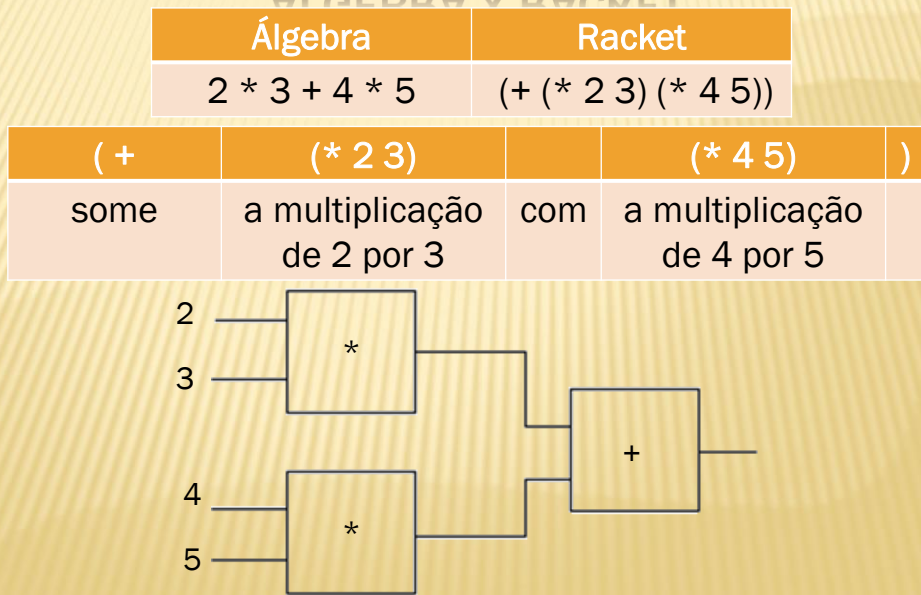
- ❑ Comparando:

Infix	Prefix	Postfix
$A+B$	$+AB$	$AB+$
$A+B-C$	$-+ABC$	$AB+C-$
$(A+B)*C-D$	$-.+ABCD$	$AB+C*D-$

## ÁLGEBRA X RACKET



## ÁLGEBRA X RACKET



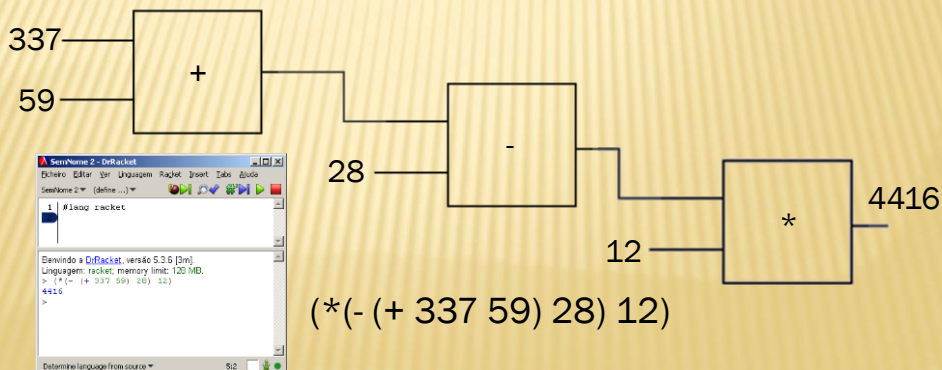
## ÁLGEBRA X RACKET

Álgebra	Racket
$(5*2)+1$	? $(+(* 5 2)1)$
$5*(2+1)$	? $(* (+ 2 1)5)$
$3.14^2$	? $(expt 3.14 2)$
Raiz de 16	? $(sqrt 16)$

## ÁLGEBRA X RACKET



- Um mercado possuía 337 dúzias de ovos. Comprou mais 59 dúzias e depois vendeu 28 dúzias. Qual o **total de ovos**? R = 4416



## RACKET: CARACTERÍSTICAS

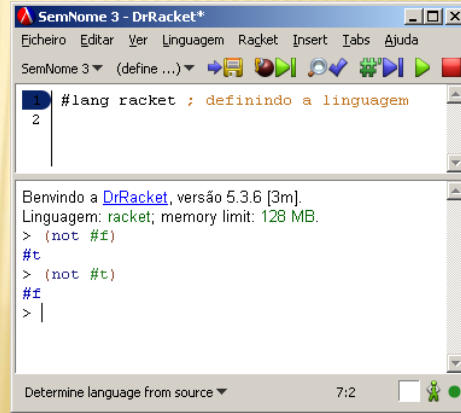
- ; comentários iniciam com ponto e vírgula
- #| comentários de bloco |#

- valores booleanos:

- #f = false
- #t = true

- not inverte a condição

- (not #f)
- (not #t)



The screenshot shows the DrRacket IDE window titled "SemNome 3 - DrRacket\*". The menu bar includes "Ficheiro", "Editar", "Ver", "Linguagem", "Racket", "Insert", "Tabs", and "Ajuda". The editor shows a Racket script with the following code:

```
#lang racket ; definindo a linguagem
2
```

The output window displays the following text:

```
Bemvindo a DrRacket, versão 5.3.6 [3m].
Linguagem: racket; memory limit: 128 MB.
> (not #f)
#t
> (not #t)
#f
> |
```

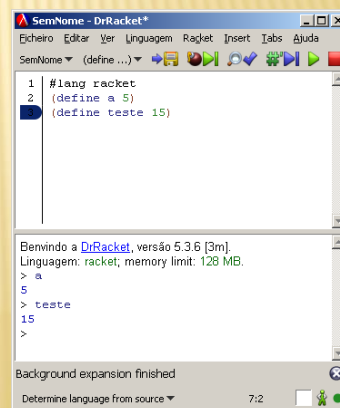
The status bar at the bottom indicates "Determine language from source" and "7:2".

## RACKET: CARACTERÍSTICAS

- Definido variáveis globais via define

- (define a 5) ; a = 5

- (define teste 15) ; teste = 15



The screenshot shows the DrRacket IDE window titled "SemNome - DrRacket\*". The menu bar includes "Ficheiro", "Editar", "Ver", "Linguagem", "Racket", "Insert", "Tabs", and "Ajuda". The editor shows a Racket script with the following code:

```
1 #lang racket
2 (define a 5)
3 (define teste 15)
```

The output window displays the following text:

```
Bemvindo a DrRacket, versão 5.3.6 [3m].
Linguagem: racket; memory limit: 128 MB.
> a
5
> teste
15
>
```

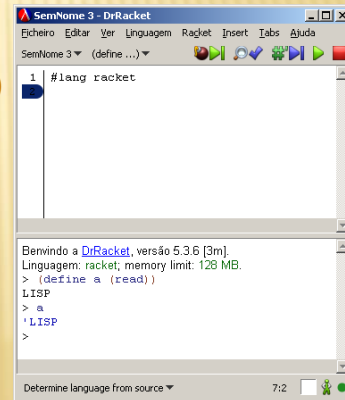
The status bar at the bottom indicates "Background expansion finished" and "Determine language from source".



## RACKET: CARACTERÍSTICAS

- ❑ Solicitando valores do usuário via `read` e atribuindo a uma variável

❑ `(define a (read)) ; a = (read)`



## RACKET: CARACTERÍSTICAS

- ❑ **Figuras.** Basta definir a linguagem. Ex:

❑ `#lang slideshow`

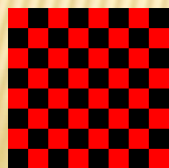


- ❑ **importar módulos/bibliotecas.** Ex:

❑ `#lang slideshow`

`; para desenhos`

❑ `(require slideshow/flash)`



# RACKET: CARACTERÍSTICAS

```

1 #lang slideshow
2 (jack-o-lantern 100)
3 (printf "\n\n")
4 (standard-fish 100 50 #:open-mouth #t #:color "olive")

```

Bem-vindo a [DrRacket](#), versão 6.10.1 [3m].  
 Linguagem: **slideshow**, with debugging; memory limit: 512 MB.

>

Determine language from source ▾ 3:11 460.39 MB

# RACKET: CARACTERÍSTICAS

❑ **Slides.** Basta definir a linguagem.

❑ **#lang slideshow**

```

1 #lang slideshow
2
3 (slide
4   #:title "How to Say Hello"
5   (item "If you want to create an example, you"
6         "can always do something with" (bt "Hello World!"))
7   (item "It's a bit silly, but a follow-up example"
8         "could be" (bt "Goodbye Drow!")))

```

Bem-vindo a [DrRacket](#), versão 6.10.1 [3m].  
 Linguagem: **slideshow**, with debugging; memory limit: 512 MB.

>

Determine language from source ▾ 3:2 334.17 MB

[https://download.racket-lang.org/docs/5.1.3/html/slideshow/Creating\\_Slide\\_Presentations.html](https://download.racket-lang.org/docs/5.1.3/html/slideshow/Creating_Slide_Presentations.html)

## RACKET: CARACTERÍSTICAS

### How to Say Hello

- If you want to create an example, you can always do something with **Hello World!**
- It's a bit silly, but a follow-up example could be **Goodbye Dirow!**

## RACKET: CARACTERÍSTICAS

- ❑ Usar funções sempre **entre parênteses**. Ex:

❑ (circle 20)

❑ (rectangle 10 20)



( ... )

- ❑ Podemos **criar** outras funções via **define**
- ❑ Podemos **reusar** as já disponíveis...

## RACKET: CARACTERÍSTICAS

- Definindo como desenhar um círculo e um quadrado:

- #lang slideshow
- (define c (circle 20))
- (define r (rectangle 10 20))

Rodar antes!

- Invocando:

- c
- r

Sem passagem de parâmetros não exige parênteses

```

1 #lang slideshow
2 (define c (circle 20))
3 (define r (rectangle 10 20))

```

Bem-vindo a DrRacket, versão 5.3.6 [3m].  
Linguagem: slideshow, memory limit: 128 MB.

> c  
> r  
> |

## RACKET: CARACTERÍSTICAS

- Criando uma nova função:

Nome da função criada

- #lang slideshow
- (define (square n)
- ; abaixo o corpo da função.
- (filled-rectangle n n)
- )

- Invocando:

- (square 50)

```

1 #lang slideshow
2 (define (square n)
3   ; abaixo o corpo da função.
4   (filled-rectangle n n)
5 )

```

Bem-vindo a DrRacket, versão 5.3.6 [3m].  
Linguagem: slideshow, memory limit: 128 MB.

> (square 50)  
>



## ESCRITA

- write
  - (write "Um texto")
- print
  - (print "Um texto")
- error
  - (error "Atenção! Há um erro na sua expressão!")
- display
  - (display "Escreve sem aspas")

```

SemNome 6 - DrRacket
Eicheiro  Editor  Ver  Linguagem  Racket  Insert  Tabs  Ajuda
SemNome 6  (define ...)
1 | #lang racket

Bemvindo a DrRacket, versão 5.3.6 [3m].
Linguagem: racket; memory limit: 128 MB.
> (write "Um texto")
"Um texto"
> (print "Outro texto")
"Outro texto"
> (error "Atenção! Há um erro na sua expressão!")
Atenção! Há um erro na sua expressão!
>
Determine language from source 2:0

```

## ESCRITA COM PARÂMETROS

- converte parâmetro para string e concatena

□ ~a

□ (~a pi)

□ (~a "π vale: " pi)

□ (~a "π vale: " pi " unidades" " de medida")

□ (display (~a "π vale: " pi))

```

Untitled - DrRacket
File Edit View Language Racket Insert Tabs Help
Untitled (define ...)
1 | #lang racket

> (~a pi)
"3.141592653589793"
> (~a "n vale: " pi)
"n vale: 3.141592653589793"
> (~a "n vale: " pi " unidades" " de medida")
"n vale: 3.141592653589793 unidades de medida"
> (display (~a "n vale: " pi))
n vale: 3.141592653589793
> |
Determine language from... 11:2 418.48 MB

```

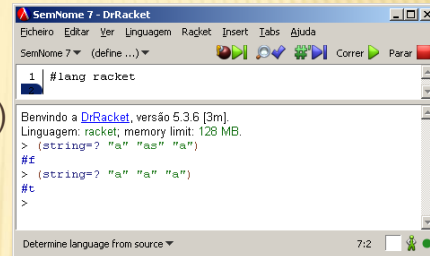
# STRINGS

## ❑ Comparação

- ❑ (string=? "Apple" "apple")
- ❑ (string=? "a" "as" "a")

## ❑ Capitalização

- ❑ (string-upcase "abc!")
- ❑ (string-downcase "aBC!")
- ❑ (string-titlecase "aBC twO")



```

1 | #lang racket

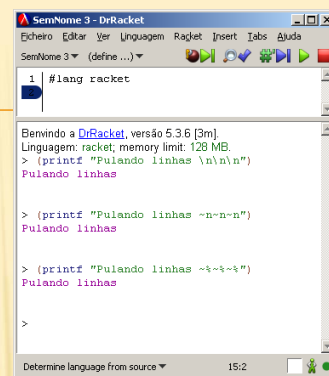
Bemvindo a DrRacket, versão 5.3.6 [3m].
Linguagem: racket; memory limit: 128 MB.
> (string=? "a" "as" "a")
#f
> (string=? "a" "a" "a")
#t
>
  
```

# STRINGS

## ❑ Caracteres Especiais

### ❑ Nova linha

- ❑ (printf "Pulando linhas \n\n\n")
- ❑ (printf "Pulando linhas ~n~n~n")
- ❑ (printf "Pulando linhas ~%~%~%")



```

1 | #lang racket

Bemvindo a DrRacket, versão 5.3.6 [3m].
Linguagem: racket; memory limit: 128 MB.
> (printf "Pulando linhas \n\n\n")
Pulando linhas

> (printf "Pulando linhas ~n~n~n")
Pulando linhas

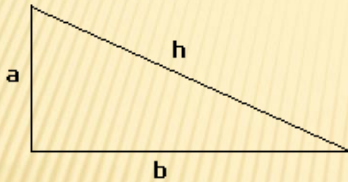
> (printf "Pulando linhas ~%~%~%")
Pulando linhas

>
  
```

## EXERCÍCIO



- Defina em Racket uma programa para calcular a **hipotenusa** de um triângulo.



- $h^2 = a^2 + b^2$
- $5^2 = 3^2 + 4^2$

(expt 3.14 2)

(sqrt 16)

```

1 #lang racket
2 (define (hipotenusa a b)
3   (sqrt
4     (+ (expt a 2) (expt b 2))
5   )
6 )

```

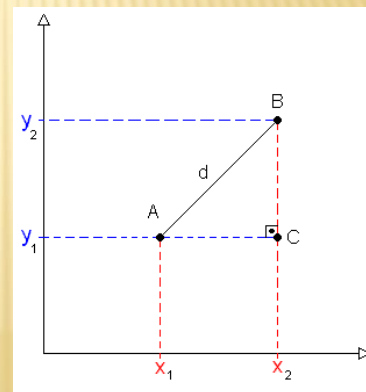
Bemvindo a [DrRacket](#), versão 5.3.6 [3m].  
Linguagem: racket; memory limit: 128 MB.  
> (hipotenusa 4 5)

## EXERCÍCIO

- Defina em Racket um programa para **calcular a distância** entre dois pontos.

$$D_{AB}^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$$

$$D_{AB} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

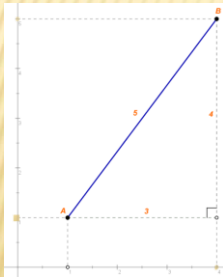


## RESPOSTA DO EXERCÍCIO

□ Para os pontos:

□ A = (1,1)

□ B = (4,5)



```

1 | #lang racket
2 | #| A(1,1) e B(4,5) |#
3 | (define (pontos x1 x2 y1 y2)
4 |   (sqrt (+
5 |     (expt
6 |       (- x2 x1)
7 |       2)
8 |     (expt
9 |       (- y2 y1)
10 |      2)
11 |   ))
12 | )
13 | )

```

Bemvindo a DrRacket, versão 5.3.6 [3m].  
 Linguagem: racket; memory limit: 128 MB.  
 > (pontos 1 4 1 5)  
 5  
 >

Determine language from source ▼ 2:22

$$d_{AB} = \sqrt{(4-1)^2 + (5-1)^2} = \sqrt{3^2 + 4^2} = \sqrt{9+16} = \sqrt{25} = 5 \text{ unidades de medida}$$