Introdução

Programação Funcional

Marco A L Barbosa
malbarbo.pro.br

Departamento de Informática
Universidade Estadual de Maringá

O quê é?

O quê é?

O que é programação imperativa?

- Um paradigma de programação onde os programas são descritos com sentenças que modificam o estado do programa.

O que é programação funcional?

- Um paradigma de programação onde os programas são descritos com aplicação e composição de funções.
- Evita mudança de estado (mudança do valor das variáveis)
- Evita efeitos colaterais (qualquer efeito que seja observável além do valor de saída de função, como a mudança dos parâmetro e variáveis global, exceções, entrada e saída, etc).

Por quê?

Por quê?

Um paradigma (linguagem) de programação é uma ferramenta.

Conhecer várias ferramentas permite utilizar a mais adequada para cada problema.

Compartilhamento de dados junto com mudança de estado é difícil!

Mudança de estado

Qual o valor de `lst`?

```
>>> lst = [0] * 3
>>> lst
[0, 0, 0]
>>> lst[1] = 10
>>> lst
```

```
[0, 10, 0]
```

Qual o valor de `lst`?

```
>>> lst = [[]] * 3
>>> lst
[[], [], []]
>>> lst[1].append(2)
>>> lst
```

```
[[2], [2], [2]]
```

```
def adiciona_todos(
        dest: list[int],
        fonte: list[int]):
    '''
    Adiciona todos os elementos
    de *fonte* no final
    de *dest*.
    '''
    for x in fonte:
        dest.append(x)
```

Qual o valor de `lst`?

```
>>> lst = [4, 3, 1]
>>> adiciona_todos(lst, [6, 2])
>>> lst
```

```
[4, 3, 1, 6, 2]
```

```
>>> adiciona_todos(lst, lst)
>>> lst
```

A execução não para!

Efeitos colaterais

As duas definições a seguir são equivalentes?

```
def soma_indices(lst: list[int], a: int, b: int) -> int:
    return indice(lst, b) + indice(lst, a)

def soma_indices(lst: list[int], a: int, b: int) -> int:
    return indice(lst, a) + indice(lst, b)
```

Não é possível afirmar que as duas definições são equivalentes sem olhar o código da função `indice`. Se a função `indice` tem efeitos colaterais, então as definições podem não ser equivalentes.

Efeitos colaterais

A possibilidade de efeitos colaterais **dificulta pensar localmente** sobre o funcionamento do código.

A ausência de efeitos colaterais **permite pensar localmente** sobre o funcionamento do código.

Como?

Como?

1) Escolher uma linguagem.
2) Estudar as construções do paradigma e as referências da linguagem.
3) Praticar lendo e escrevendo código.

Como?

1) Escolher uma linguagem
    - Racket (variante moderna do Lisp)
    - Bom suporte ao paradigma funcional
    - Ambiente integrado DrRacket
    - Documentação extensa
    - Fácil instalação

2) Estudar as construções do paradigma e as referências da linguagem

   - A Tutorial Introduction to the Lambda Calculus
   - Livro How to Design Programs
   - Guia e Referência do Racket
   - Livro Structure and Interpretation of Computer Programs
   - Livro The Scheme Programming Languague

Como?

3) Praticar lendo e escrevendo código
    - Muitos exemplos
    - Muitos exercícios

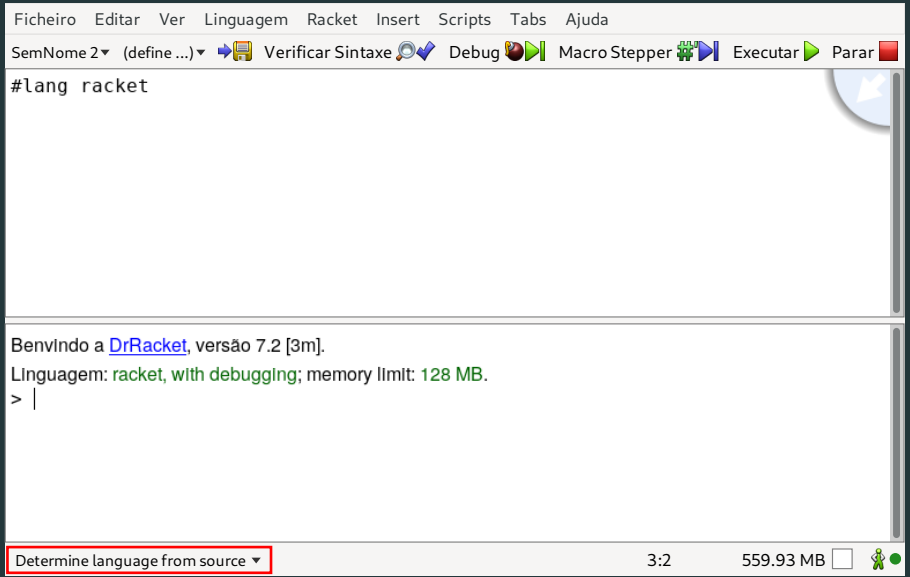
Primeiros passos

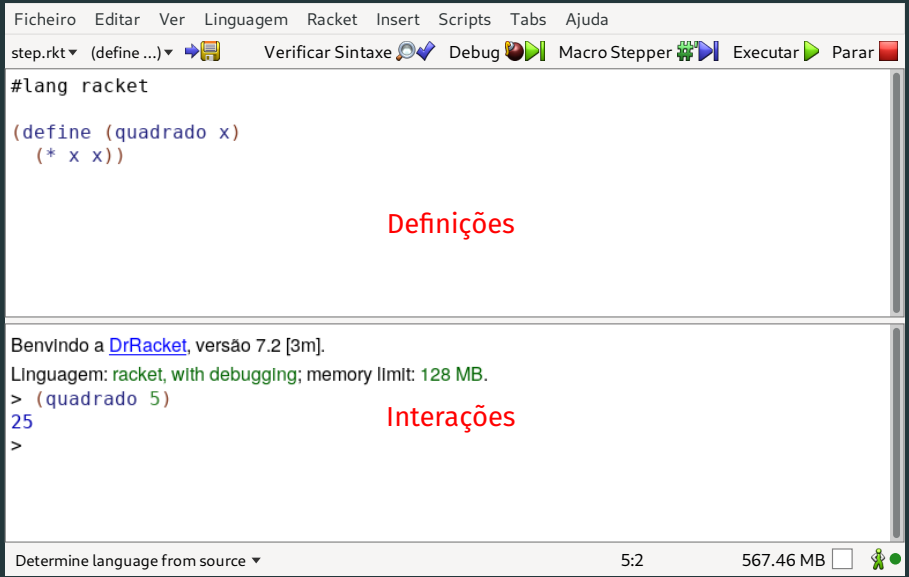
Instalação

```
$ apt-get install racket
```

Execução

```
$ drracket
```


Ficheiro Editar Ver Linguagem Racket Insert Scripts Tabs Ajuda
SemNome 2
(define ...)
Verificar Sintaxe
Debug
Macro Stepper
Executar
Parar
#lang racket
Benvindo a DrRacket, versão 7.2 [3m].
Linguagem: racket, with debugging; memory limit: 128 MB.
>
Determine language from source
3:2
559.93 MB


Ficheiro Editar Ver Linguagem Racket Insert Scripts Tabs Ajuda
step.rkt (define ...) Verificar Sintaxe Debug Macro Stepper Executar Parar
#lang racket
(define (quadrado x)
(* x x))
Definições
Benvindo a DrRacket, versão 7.2 [3m].
Linguagem: racket, with debugging; memory limit: 128 MB.
> (quadrado 5)
25
>
Interações
Determine language from source
5:2
567.46 MB

Uso

Na área de definições

- Digite o código do programa
- Pressione o botão executar/correr (ctrl + r))

Na área de interações

- Teste as definições usando REPL (*Read Eval Print Loop*)
- Entre com uma expressão (*Read*)
- A expressão é avaliada (*Eval*)
- O resultado da expressão é impressa (*Print*)
- Repetida o processo (*Loop*)

Exemplos de interações

```
> 34
34
> (+ 3 4)
7
> (* 2 6)
12
> +
#<procedure:+>
```

Leitura

Leitura

Recomendada

- Introdução rápida ao Racket
- Programação funcional

Extra

- The Python paradox
- Revenge of the Nerds
- Beating the averages