

### **INTRODUÇÃO**

**1 - O ESSENCIAL DO SCHEME**

**2 - RECURSIVIDADE**

**3 - ABSTRACÇÃO DE DADOS**

**4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE**

**5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS**

**6 - SCHEME E OUTRAS  
TECNOLOGIAS**

**7-EXERCÍCIOS  
E  
PROJECTOS**

**ANEXOS**

## **Introdução**

Uma parte substancial desta obra tem sido utilizada na primeira unidade curricular de programação do curso de informática da Faculdade de Engenharia da Universidade do Porto (FEUP). Os trabalhos realizados no âmbito desta unidade curricular apoiam-se em plataformas onde se evidencia a linguagem Scheme, uma linguagem de programação pouco difundida em Portugal, apesar da sua larga utilização em importantes universidades americanas e europeias. Em Portugal, para além da FEUP, só conhecemos a sua utilização em cursos de informática do Instituto Superior Técnico (IST) da Universidade Técnica de Lisboa.

Esta obra apresenta algumas características inovadoras como manual de apoio aos estudantes de programação, pois os autores tiveram a preocupação de ir ao encontro de novas atitudes pedagógicas, apostando no paradigma da aprendizagem onde o estudante é incentivado a ser o actor principal.

Não foi descurada a crítica que classifica o Scheme como uma linguagem apenas para uso no ensino da programação, tendo sido reservado espaço para o estabelecimento de pontes entre esta linguagem e outras tecnologias importantes para a construção de software (testes unitários, desenvolvimento de GUI, programação OO, aplicações multimédia e jogos).

Em termos de público-alvo, identificam-se os estudantes com interesse na iniciação à programação, sobretudo estudantes do ensino superior, mas também estudantes do ensino secundário da área da informática.



# **S C H E M E**

## **na descoberta da programação**

### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

Aprender programação com a linguagem Scheme é uma opção já com alguns anos em importantes universidades americanas (MIT, Yale e Indiana) e europeias. Na Faculdade de Engenharia da Universidade do Porto (FEUP), a utilização do Scheme iniciou-se em 1991, na Licenciatura de Gestão Industrial, e em 1994 na Licenciatura de Engenharia Informática e Computação. Em Portugal, fora da FEUP, apenas se conhece o seu uso no Instituto Superior Técnico da Universidade Técnica de Lisboa.

A simplicidade sintáctica desta linguagem permite que o esforço do estudante incida sobretudo na programação e não nas ferramentas de apoio à aprendizagem. Para combater o argumento de que o Scheme é sobretudo uma linguagem de aprendizagem de programação, procurou-se estabelecer pontes entre o Scheme e outras tecnologias importantes para construção de software (testes unitários, desenvolvimento de GUI, programação OO, aplicações multimédia e jogos). Ficam assim garantidas a ligação do Scheme a outras disciplinas desta área e também a sua quota de espaço na resolução de problemas em ambiente "menos académico".

A versatilidade desta linguagem é aproveitada para introduzir, naturalmente, o paradigma da programação funcional, mas também outros paradigmas de programação como, por exemplo, a orientação por objectos. É, sem dúvida, uma oportunidade rara para o estudante se envolver, mesmo que de uma forma breve, na implementação destes paradigmas e não só na sua utilização, como acontece com a generalidade de outras linguagens.

Quando tanto se fala na mudança de paradigma de ensino, aqui apostava-se no paradigma da aprendizagem, através do incentivo ao estudante para que "aprenda fazendo". Este objectivo é procurado através de um estilo de apresentação de conteúdos que promove uma aprendizagem activa, estimulando a atenção, a experimentação e a procura de respostas.



**A inserção de perguntas ou pequenos desafios, relacionados com os temas em presença, tem por objectivo despertar a atenção do estudante.  
O que acha desta estratégia de actuação?**



Os conteúdos surgem organizados em Partes, compostas por módulos, normalmente de uma a três dezenas de páginas, tanto quanto possível autónomos, para possibilitar uma sequência personalizada de aprendizagem e de aprofundamento de conhecimentos. Em sintonia com este estilo de apresentação, é também incluído um espaço de interacção ligado ao texto, designado por Laboratório Scheme, que permite o desenvolvimento e teste dos numerosos exemplos e exercícios propostos, sem que a atenção do estudante seja muito perturbada pela entrada em cena de outra plataforma de trabalho.

Se está disposto e com vontade de enfrentar este desafio, vai ter que começar por instalar uma implementação do Scheme no seu computador. Combinado?

Nós vamos utilizar o DrScheme, a versão 372, uma plataforma onde foram testados os muitos exemplos que irá encontrar neste percurso que agora inicia, uma plataforma de domínio público disponível para sistemas *Linux*, *Mac* e *Windows*.

Recomendamos que também instale a versão 372, disponível localmente para **Windows**, Mac OS X (**Intel** e **PPC**) e Linux (**Ubuntu**, **Ubuntu Feisty**, **Fedora Core 6** e **Fedora 7**).

No entanto, em [\*\*PLT Scheme\*\*](#), pode encontrar outras versões do DrScheme e muitos outros recursos para utilizar neste ambiente.

Depois da instalação da versão 372 do DrScheme, deve descomprimir e copiar para o seu directório PLT/collects/ do DrScheme, o directório **user-feup**, com código preparado por professores e estudantes da FEUP e que lhe será muito útil.



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

E agora está tudo preparado para poder experimentar o primeiro, entre muitos, exercício de laboratório, no espaço que se segue.



Teste o Laboratório Scheme com um programa de ajuda ao preenchimento de boletins do Totoloto.

The screenshot shows the DrScheme interface with a window titled "preenche\_simplesfinal.scm - DrScheme". The code in the editor is:

```
; PROGRAMA PARA PREENCHER BOLETINS DE TOTOLOTO
;
; F. Nunes Ferreira - fnf@fe.up.pt
; 4 de Novembro de 2000
;
; Modo de utilização
; Gerar um número desejado de apostas de totoloto.
; Ex: Para gerar 10 apostas
; (preenche-totoloto 10)
;
(define preenche-totoloto
  (lambda (num-apostas)
    (let ((registro (faiz-registro)))
      ; ... (code omitted) ...
      ))
```

The interaction pane shows:

```
Welcome to DrScheme, version 299.400p1.
Language: Graphical (MrEd, includes MzScheme).
> (preenche-totoloto 3)
```

Apostas seleccionadas:

1	17	23	33	36	45
7	10	12	34	46	47
19	28	31	37	40	43

Below the interaction pane, there is a callout bubble with the text: "Entre parêntesis curvos, escreva o nome do programa e o número de apostas... (preenche-totoloto 4)".



Actue o botão run... tudo recomeça e pode continuar a pedir apostas...



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Panorâmica da obra

Esta obra é composta por um conjunto de Partes e Anexos que agora são apresentadas de uma forma resumida.

**Parte 1**- O essencial do Scheme

**Parte 2**- Recursividade

**Parte 3**- Abstracção de dados

**Parte 4**- Procedimentos como objectos de 1<sup>a</sup> classe

**Parte 5**- Abstracções com dados mutáveis

**Parte 6**- Scheme e outras tecnologias

**Parte 7**- Exercícios e projectos

**Anexo A**- Principais procedimentos do Scheme

**Anexo B**- Abstracção Janela Gráfica

**Anexo C**- Reutilização de código

**Anexo D**- Abstracção Tabuleiro

A **Parte 1**, O essencial sobre Scheme, faz uma breve introdução à linguagem Scheme, considerando os números, expressões, símbolos, definição de procedimentos e estruturas de selecção. No final desta parte, encontrará uma introdução à programação através do uso de abstracções, com exemplos de programas que produzem sons e resolvem labirintos.

Para além desta Parte, a sintaxe do Scheme resumir-se-á, fundamentalmente, a notas pontuais ao longo de outras partes da obra e a referências a um dos anexos (**Anexo A**- Principais procedimentos do Scheme).



### INTRODUÇÃO

#### 1 - O ESSENCIAL DO SCHEME

#### 2 - RECURSIVIDADE

#### 3 - ABSTRACÇÃO DE DADOS

#### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

#### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

#### 6 - SCHEME E OUTRAS TECNOLOGIAS

#### 7-EXERCÍCIOS E PROJECTOS

#### ANEXOS



Aqui apenas se sugere um pequeno desvio até ao [Anexo A](#).

O que é que acha de uma linguagem cuja sintaxe se resume num anexo de uma dezena de páginas, ocupadas, em grande parte, por exemplos?

A **Parte 2**, Recursividade, considera o conceito da recursividade e a definição de procedimentos recursivos, procedimentos que se chamam a si próprios. É através deste tipo de procedimentos que se ultrapassa, no Scheme, a ausência dos ciclos que existem na generalidade das outras linguagens de programação. Mas a recursividade é muito mais rica do que isto, pois habilitar-nos-á a encontrar ideias para resolver problemas, alguns de grande complexidade, que de outra forma não seriam fáceis de resolver. As chamadas dos procedimentos geram processos no computador. A noção de processo, seja ele recursivo ou iterativo, é também considerada, sendo ainda analisada a forma como consome recursos computacionais, tempo de CPU e espaço de memória. Estes recursos consumidos são avaliados através da Ordem de crescimento ou Ordem de complexidade. Esta parte termina com os procedimentos vistos como blocos ou caixas-pretas, sendo ainda apresentadas as vantagens e desvantagens que daí podem resultar.

A complexidade dos problemas poderá exigir uma análise que identifique as suas principais tarefas e, entre estas, as que ainda forem consideradas complexas também serão sujeitas a uma análise idêntica. E o processo repete-se enquanto a complexidade de alguma tarefa o exigir. Trata-se da abordagem **de-cima-para-baixo**, em que se decompõe o problema em problemas cada vez mais simples.



**Hoje, depois da leitura desta introdução, imagine que vai tirar o seu carro da garagem para dar uma volta.  
No entanto, o carro tem uma roda furada.**

**Identifique e escreva num papel a sequência de operações que deve realizar para tirar o carro da garagem.**



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



**Na sequência das operações, vou imaginar que indicou uma designada por "mudar a roda furada". Por se tratar de uma operação relativamente complexa, dedique-lhe um pouco mais de atenção. E agora, só para esta operação, identifique e escreva num papel a sequência de operações elementares em que ela se pode decompor.**



**Se tiver conseguido tudo isto terá escrito um programa. Se não tiver conseguido, não desanime, pois ainda agora está no início... Mesmo que tenha escrito um programa para tirar o seu carro da garagem, explique por que razão o computador não o entenderia.**

A Parte 3, Abstracção de dados, mostra como, a partir de dados simples, podemos compor os dados que se adequam à representação dos problemas que pretendemos resolver. Por exemplo, num problema de características gráficas a 2 dimensões (2D), a criação da entidade ponto é perfeitamente justificável, sendo constituída por dois valores numéricos que representam as suas coordenadas x e y no plano. Para este caso, será necessário agrupar dois valores numéricos criando objectos do tipo `ponto-2D`. Na criação de novas entidades, a partir de dados simples ou de outras entidades já criadas, utilizam-se os construtores e o acesso aos elementos das novas entidades é realizado com os selectores. E é em torno destes construtores e selectores que normalmente se desenvolve uma certa funcionalidade que permite tratar as entidades criadas através de uma linguagem própria, orientada para o problema a resolver, surgindo assim a noção de abstracção de dados.



## INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A **Parte 4**, Procedimentos como objectos de 1<sup>a</sup> classe, começa por mostrar que os objectos de 1<sup>a</sup> classe são, normalmente, os operandos (números, booleanos, pares e símbolos), entidades que podem ser processadas e, por isso, reconhecidas como objectos passivos. Os procedimentos são objectos activos, pois associam-se-lhes actividades de processamento. As propriedades dos objectos que os caracterizam como objectos de 1<sup>a</sup> classe são: 1- Ligam-se a símbolos; 2- são passados como argumentos de procedimentos; 3- são devolvidos como valores de retorno de procedimentos; 4- são elementos de estruturas compostas de dados. Nesta parte, poderá verificar que os procedimentos também apresentam estas quatro características e, por isso, eles próprios são também objectos de 1<sup>a</sup> classe.

A **Parte 5**, Abstracções com dados mutáveis, mostra que, para além dos construtores e selectores, também existem modificadores. Começam por surgir as chamadas listas mutáveis que estarão na base da implementação das abstracções filas, pilhas e tabelas. Os vectores e as cadeias de caracteres (*string*) são abstracções de dados mutáveis directamente disponibilizadas pelo Scheme, e mostra-se o que trazem de novo em relação às restantes abstracções. Nesta parte é ainda introduzido o tema ficheiros, tendo em conta a necessidade de guardar em disco dados de uma sessão de trabalho para outra.

Perante várias abstracções, as fornecidas pela linguagem e as criadas pelo programador, poderá não ser fácil decidir qual a melhor em cada situação ou até se não será necessário criar uma nova. No decurso desta parte procura-se analisar as grandes diferenças entre vectores e listas mutáveis. As listas são estruturas de dados dinâmicas, de grande flexibilidade, pois é possível juntar-lhes ou retirar-lhes elementos. As listas têm um acesso do tipo sequencial, pois para chegar a um dos seus elementos é necessário passar por todos os elementos que o antecedem. Os vectores são normalmente estruturas estáticas, de dimensão fixa, mas todos os seus elementos são igualmente acessíveis, através do respectivo índice. Por seu turno, as árvores de pesquisa binária surgem como uma estrutura que procura reunir o melhor dos dois mundos, o melhor dos vectores e das listas.



**Do que acabou de ser dito, conseguirá já identificar o que será o melhor dos vectores? E o melhor das listas?**  
**Nesta Parte 5 poderá ver como se comparam as árvores de pesquisa binária em termos do que têm de melhor os vectores e as listas.**



### INTRODUÇÃO

#### 1 - O ESSENCIAL DO SCHEME

#### 2 - RECURSIVIDADE

#### 3 - ABSTRACÇÃO DE DADOS

#### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

#### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

#### 6 - SCHEME E OUTRAS TECNOLOGIAS

#### 7-EXERCÍCIOS E PROJECTOS

#### ANEXOS

Na **Parte 6**, Scheme e outras tecnologias (GUI, Jogos, Testes unitários, Executáveis e Programação OO), o objectivo principal é mostrar que o Scheme não é só aquela linguagem essencialmente usada para aprender a programar. Isso fica provado com: o desenvolvimento de interfaces gráficas (GUI) para os programas Scheme; o desenvolvimento de programas multimédia, em especial jogos, sobre plataformas multimédia de acesso gratuito; o teste ao funcionamento do código desenvolvido, pela associação de testes unitários sempre prontos a ajudar na identificação de falhas; a geração de um código executável a partir de programas Scheme, permitindo a execução desse código em computadores que não tenham o Scheme instalado; e com a demonstração da flexibilidade do Scheme para introduzir conceitos associados ao paradigma da programação orientada por objectos, tanto do ponto de vista da implementação desses conceitos como do respectivo uso.

A **Parte 7**, Exercícios e projectos, não é mais do que um conjunto de enunciados de exercícios e projectos, poucos com pistas de solução, de forma a colocar o estudante perante problemas de programação completamente desligados deste ou daquele tema. Quando se coloca um exercício no final de uma Parte, uma primeira pista de solução fica implicitamente estabelecida, mas no mundo real os problemas não aparecem rotulados ou associados a partes ou a capítulos de qualquer livro. Perante um problema, torna-se necessário encontrar uma ideia para o resolver, vaga no início, mas sucessivamente mais refinada. Só à medida que as várias tarefas e sub-tarefas do problema vão sendo identificadas é que se clarificam as associações com as matérias tratadas. Surge a forma de representar os dados do problema, acompanhada de um certo conjunto de construtores, selectores e até modificadores, ou seja, surge a abstracção de dados. E é sobre a abstracção idealizada, com as tarefas e sub-tarefas identificadas, que se refina o desenvolvimento da solução.

### Programas desenvolvidos em Scheme e alguns dos projectos propostos

A aprendizagem da programação deverá ser sistematicamente praticada no computador, o que justifica a inclusão de numerosos exemplos, exercícios e também alguns projectos de pequena/média dimensão. No caso particular dos projectos, procurou-se que fossem particularmente atractivos, pois exigem algum tempo e esforço e também alguma imaginação. É um desafio que se coloca e que transmitirá, a quem o vencer, um elevado grau de autoconfiança e gosto pela programação.

Desde já se apresentam exemplos de programas desenvolvidos em Scheme, alguns deles da autoria de estudantes, onde se mostra o que se poderá fazer com o Scheme num curso de iniciação à programação.



## INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Programa 1 -Troca quantia expressa em cêntimos e calcula número de trocas distintas

```
; (conta-trocas quantia)
; devolve numero de hipóteses de troca de uma quantia

; (troca quantia)
; devolve hipótese de troca com menos unidades monetarias

(define unidades-monetarias
  ; lista das unidades monetarias especificadas em centimos
  (list 50000 20000 10000 5000 2000 1000 500 200 100 50 20 10 5 2 1))

Welcome to DrScheme, version 299.400p1.
Language: Graphical (MrEd, includes MzScheme).
> (conta-trocas 3)
2
> (conta-trocas 5)
4
> (troca 3)
2 centimo
1 centimo
ok
> (troca 5)
5 centimo
ok
>
```



Teste os programas troca e conta-trocas.





Teste o programa adivinhar.  
Também pode jogar com um número diferente de 5 perguntas. Tente...

## Programa 2 - Adivinha um número pensado pelo utilizador

> (adivinar 5)

Pense num numero entre 0 e 31!

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31

P1- O numero pensado esta no conjunto (1=sim, 0=nao)?

2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31

P2- O numero pensado esta no conjunto (1=sim, 0=nao)?

4 5 6 7 12 13 14 15 20 21 22 23 28 29 30 31

P3- O numero pensado esta no conjunto (1=sim, 0=nao)?

8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31

P4- O numero pensado esta no conjunto (1=sim, 0=nao)?

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

P5- O numero pensado esta no conjunto (1=sim, 0=nao)?

O numero pensado foi: 13

2

# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Programa 3 - Um pequeno animal percorre um labirinto, com visualização num tabuleiro

Normalmente, o pequeno animal parte da célula do canto superior-esquerdo do tabuleiro e tem como objectivo alcançar a célula do canto inferior-direito... mas o melhor é experimentar. E atenção ao som introduzido na versão do estudante Luís Santos, sobretudo quando o animal atinge o objectivo (e não se poderá dizer que seja muito bem educado!...).

```
; LABIRINTO
; Baseado no código caminho2.scm do professor Fernando Nunes Ferreira
; fnf@fe.up.pt
; Modificado por Luis Santos
; ei00008@fe.up.pt
;
; correr o programa -> (caminho)
;
; Opções do Menu
; 1-> Definir obstáculos um a um
; 2-> Definir uma nova posição de partida
> (caminho)
Lado do tabuleiro em celulas : 10
Lado da célula em pixels: 25

1-Definir obstáculo manualmente
2-Obstáculos aleatórios
3-Definir posição partida
4-Definir posição final
5-
6-
7-Andar (metodo Luis Santos)
8-Andar (metodo Prof. FNF)
9-Limpar trajectória

10- fim
=>
numero de obstáculos? 35
```



Teste o programa labirinto.  
Tente as várias opções do menu.



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Programa 4 - A torre de Hanoi com solução textual e gráfica

Os anéis colocados inicialmente na torre da esquerda (e) devem, no final, estar todos na torre da direita (d), usando-se a torre central (c) como auxiliar. Só se pode movimentar um anel de cada vez e nunca um anel de menor diâmetro poderá estar debaixo de um anel de maior diâmetro.

The screenshot shows the DrScheme environment. At the top, there is a Scheme source code window containing:  
; (torre-de-hanoi-1 4)  
; solução sob a forma de uma lista de movimentos, para 4 anéis  
;  
; (hanoi-anim n-anéis largura-janela altura-janela)  
; (hanoi-anim 4 400 400)  
; solução sob a forma gráfica, para 4 anéis  
;  
Welcome to DrScheme, version 299.400p1.  
Language: Graphical (MrEd, includes MzScheme).  
> (torre-de-hanoi-1 3)  
(#(e d) #(e c) #(d c) #(e d) #(c e) #(c d) #(e d))  
> (hanoi-anim 3 200 200)|  
qualquer tecla para avançar...a  
ok  
>  
Below this is a stack of three windows titled "hanoi-anim". Each window displays a state of the Hanoi Tower with three vertical rods. The first window shows four rings on the leftmost rod. The second window shows three rings on the middle rod. The third window shows all four rings on the rightmost rod. The windows are arranged vertically, representing the progression of the animation.



Teste os programas `torre-de-hanoi-1` e `hanoi-anim`.

Para testar a solução textual com 3 anéis:

```
> (torre-de-hanoi-1 3)
```

Para testar a solução gráfica com 4 anéis, numa janela gráfica de 400 x 400:

```
> (hanoi-anim 4 400 400)
```

Tente com um número variado de anéis, tanto na solução textual como na gráfica.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Programa 5 - Jogo hangman - cabe ao utilizador adivinhar uma palavra

O jogador é desafiado a adivinhar uma palavra indicando letras, uma a uma. Nesta tarefa pode contar com algumas ajudas do computador, nomeadamente o comprimento da palavra e, das letras já tentadas, o computador fornece a lista das que estão erradas e a posição na palavra das letras indicadas correctamente.

```
> (hangman)
palavra a adivinhar: (* * * * * * *)
letras erradas: ()
Proxima letra: o
errou...
palavra a adivinhar: (* * * * * * *)
letras erradas: (o)
Proxima letra: a
palavra a adivinhar: (* a * a * * a)
letras erradas: (o)
Proxima letra: v
palavra a adivinhar: (* a * a v * a)
letras erradas: (o)
Proxima letra: p
palavra a adivinhar: (p a * a v * a)
letras erradas: (o)
Proxima letra: l
palavra a adivinhar: (p a l a v * a)
letras erradas: (o)
Proxima letra: r
palavra a adivinhar: (p a l a v r a)
letras erradas: (o)
Acertou...
```



Teste o programa hangman.



Na listagem do programa, veja as palavras que podem aparecer. Tente juntar novas palavras, tendo o cuidado de separar as letras por um espaço. Actue o botão run e recomece o jogo...



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

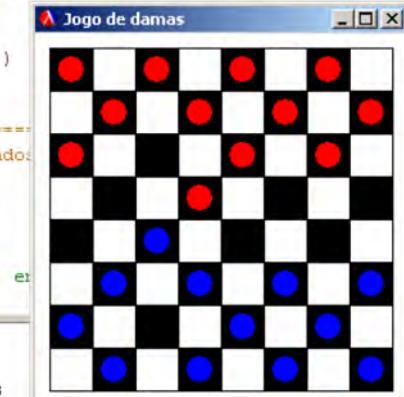
### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Programa 6 - Jogo das damas num tabuleiro visualizado no ecrã

Este jogo foi concebido e desenvolvido pelo estudante Pedro Sousa, mas apresenta ainda algumas limitações, nomeadamente as pedras que atingem a última linha não permitem mais movimentos (dama mestre) e também não são admitidos movimentos para "comer" mais do que uma peça.

```
; Jogo feito por:  
; Pedro Daniel Sousa  
; e-mail: ei05060@fe.up.pt  
;  
; Ano lectivo: 2005/2006  
;  
; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;  
; Chamada de bibliotecas necessárias ao jogo  
  
(require (lib "swgr.scm" "user-feup"))  
(require (lib "tabuleiro.scm" "user-feup"))  
  
=====  
; Cria uma janela com um tabuleiro, ajustado:  
; nao necessita argumentos  
;  
(define jogo-damas  
  (lambda ()  
    (display "Lado do tabuleiro, de damas, em celulas:  
    (let ((dim (read)))  
      (display dim))  
    (newline)))  
  
Welcome to DrScheme, version 299.400p1.  
Language: Graphical (MrEd, includes MzScheme).  
Lado do tabuleiro, de damas, em celulas: 8  
Lado da celula em pixels: 30  
-----  
  

```



Teste o programa `jogo-damas`.



Quando aprender um pouco mais de Scheme e programação, não quererá completar este programa?  
Fica o desafio...

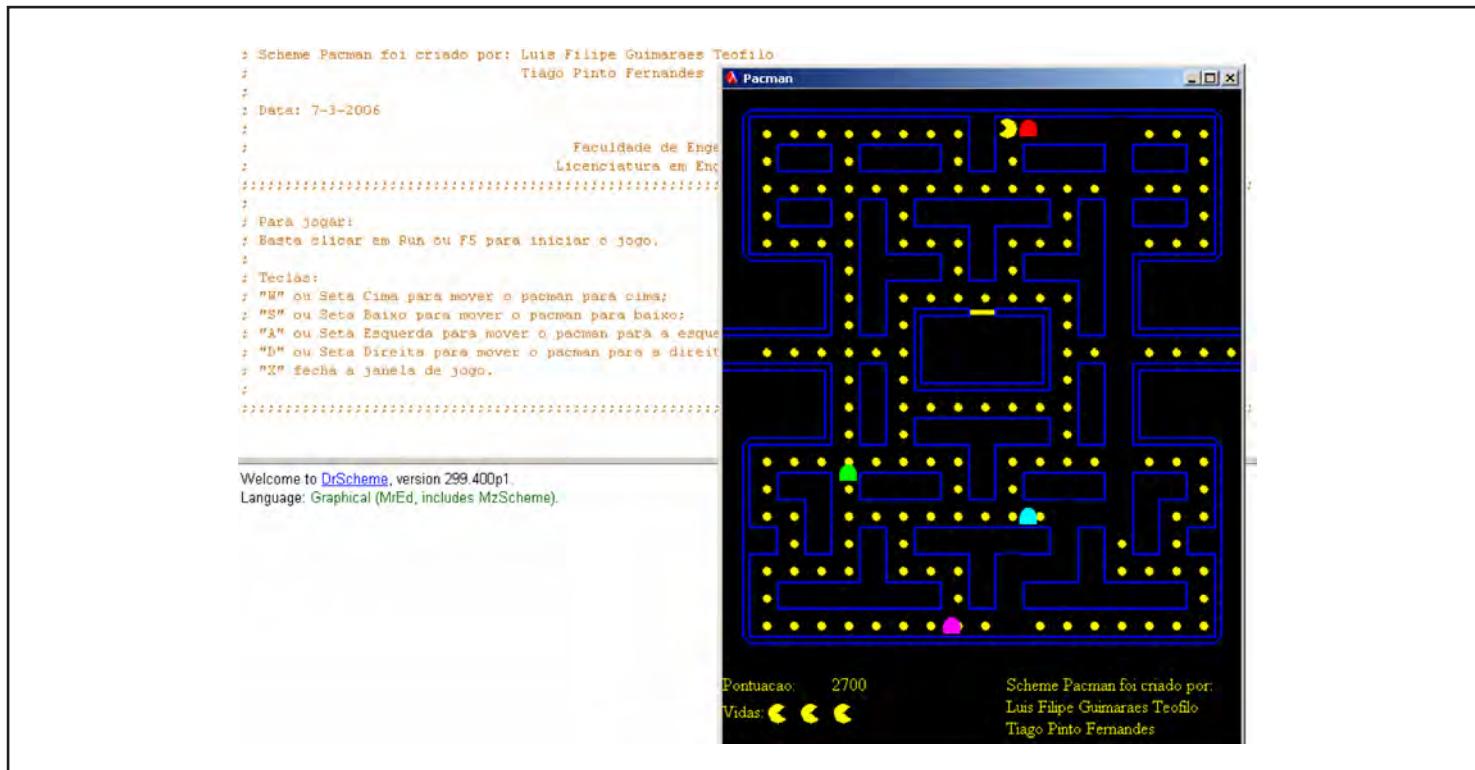




Teste o programa pacman.

## Programa 7 - Jogo *pacman*

Este programa foi desenvolvido pelos estudantes Tiago Fernandes e Luís Teófilo e oferece uma certa interactividade através das teclas w (cima), s (baixo), a (esquerda), d (direita) e x (fim). Trata-se de um jogo que exige alguma rapidez e habilidade, mas o melhor é experimentar.



# **S C H E M E**

## **na descoberta da programação**

### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

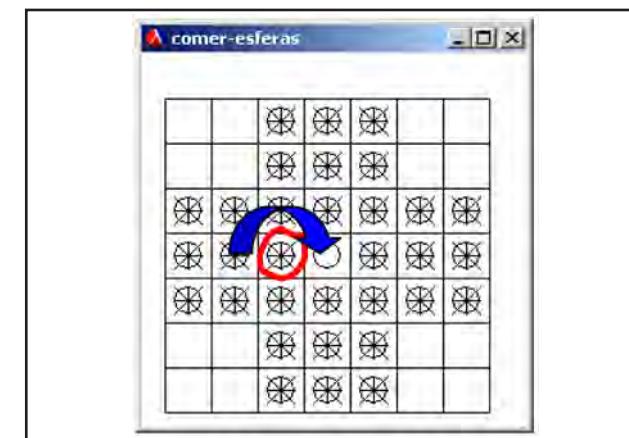
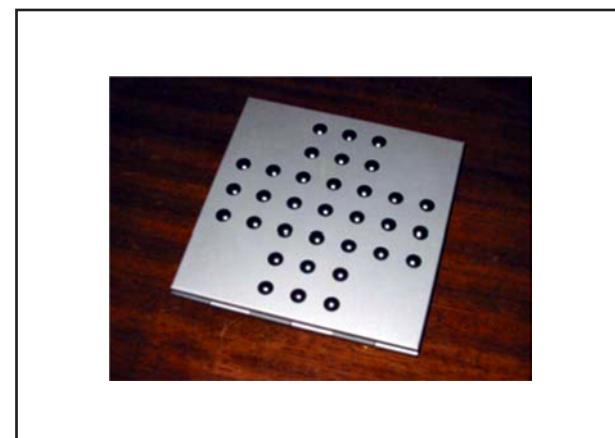
### **ANEXOS**

Quanto a projectos propostos, distinguem-se apenas os mais significativos.

Na Parte 3, surgem três propostas de projecto: um primeiro, com elevado nível de dificuldade, designado por Colorir Mapas, relacionado com o problema de colorir um mapa de países com um número limitado de cores, evitando que a mesma cor seja utilizada para colorir países com fronteiras adjacentes; um segundo projecto, o Jogo das minas, simula um terreno de minas, em variadíssimas situações, desde as minas muito ricas às muito pobres. De acordo com uma forma original de se deslocar no terreno, um mineiro vai visitando as minas, recolhendo (nas ricas) ou perdendo (nas pobres) riqueza; o terceiro projecto, Lançamento-de-projetéis, reveste a forma de um jogo electrónico simplificado, com interface gráfica, onde se procura alcançar, com um projétil, um objecto colocado aleatoriamente no espaço para testar a pontaria de quem joga. Trata-se de um projecto com visualização gráfica animada.

O CODEC é o projecto introduzido na Parte 4 que trata, de uma forma muito simplificada, o problema da codificação e descodificação de mensagens.

Na Parte 5 distinguem-se dois projectos: o primeiro, Helicóptero-digital, em que se controla, através do teclado, um helicóptero imaginário; no segundo, Comer-esferas, imagine que se retira uma das esferas da base indicada na fotografia, criando assim um buraco. Em seguida, essa esfera "come" outra se passar por cima dela para ocupar um buraco. Na figura, a esfera na base da seta, na sua passagem para o buraco, "come" a esfera assinalada.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

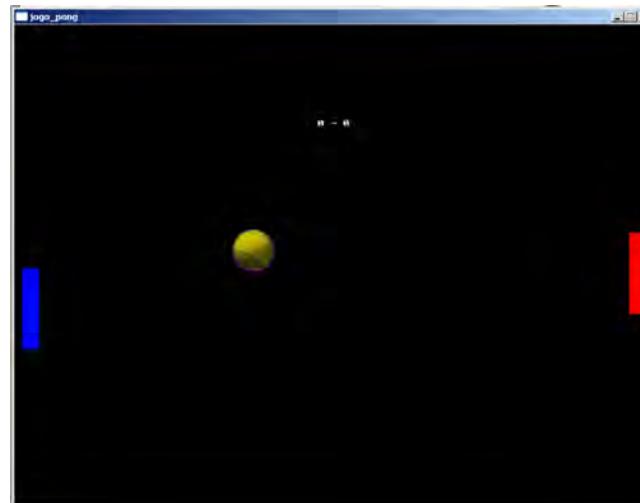
### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

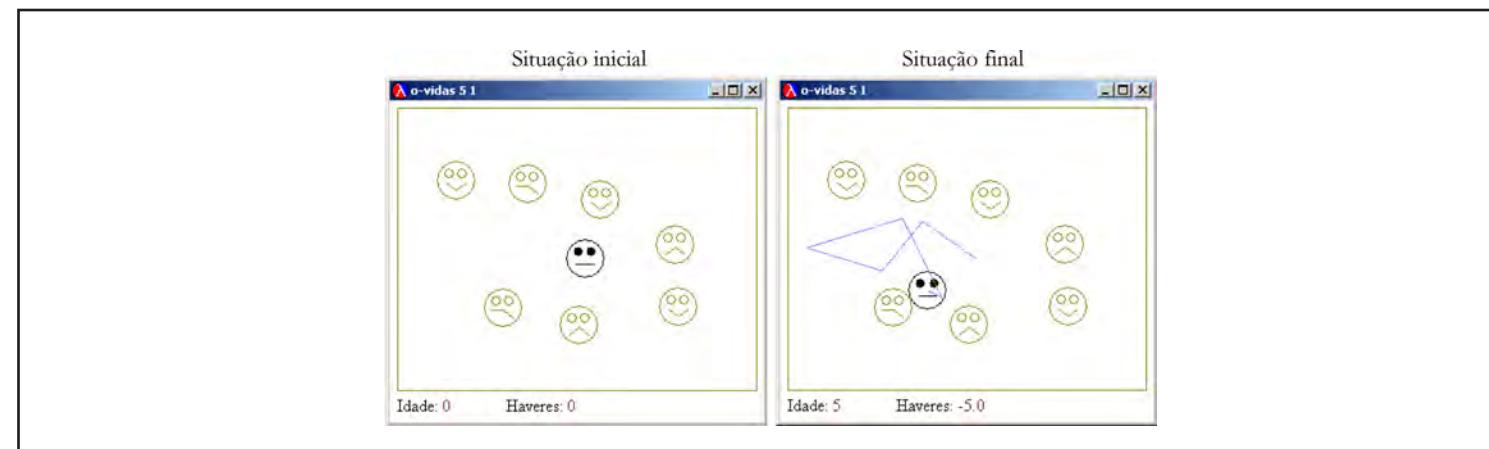
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Na Parte 6 destacam-se dois projectos. O primeiro projecto, o jogo Pong, tem por base uma plataforma para desenvolvimento de programas com interface multimédia. Esta plataforma, designada por Allegro, encontra-se documentada em [Allegro: Multimedia library and game utilities](#).



O jogo de o-vidas é um projecto com um grau de dificuldade já bastante elevado e aparece com uma implementação orientada por objectos. Simula o percurso de uma criatura num tabuleiro, designado por o-vidas, que muda de trajectória quando embate nas paredes ou em criaturas distribuídas aleatoriamente no tabuleiro...



# **S C H E M E**

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

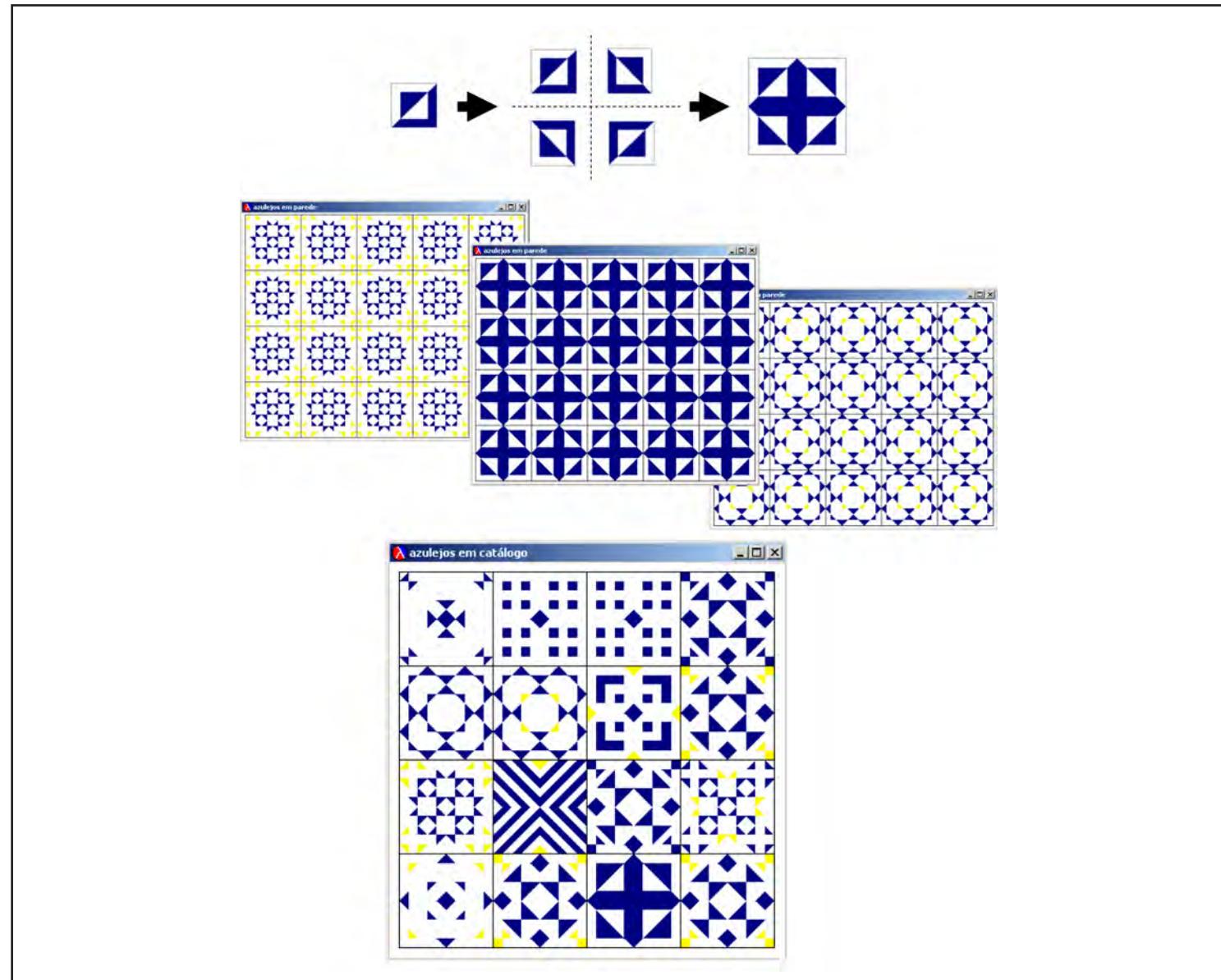
### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

São vários os projectos incluídos na Parte 7 e apenas se distinguem dois: Caminho-mais-curto, uma espécie de labirinto em que os percursos são caracterizados pelo respectivo comprimento e procura-se o caminho mais curto entre um ponto à escolha e uma saída; e Pintura de azulejos, um programa de apoio à concepção de azulejos, que começa por elementos, passa por azulejos isolados e termina em catálogos de azulejos.



# **S C H E M E**

## **na descoberta da programação**

### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

## **Bibliografia e outros recursos de apoio**

Esta obra surge da experiência de lecionação com o Scheme, na primeira unidade curricular de programação do curso de informática da Faculdade de Engenharia da Universidade do Porto (FEUP), iniciada no ano lectivo 1994/95. Neste já longo percurso, estudantes e docentes foram acompanhados, entre outros, por dois livros excepcionais e de inquestionável valor.

Harold Abelson, Gerald Jay Sussman, Julie Sussman

Structure and Interpretation of Computer Programs, 2nd edition

McGraw-Hill Book Company, 1985, second edition, 1996

George Springer and Daniel P. Friedman

Scheme and the Art of Programming

McGraw-Hill, Sixth printing, 1993

Outros livros merecem também uma referência, nomeadamente

James Little, Vincent Manis

The schematics of computation

Prentice-Hall, ISBN:0-13-834284-9, 1995

e um outro por ser o primeiro publicado em português.

João Pavão Martins e Maria dos Remédios Cravo

Programação em Scheme: introdução à programação utilizando múltiplos paradigmas

IST Press, Instituto Superior Técnico, ISBN: 972-8469-32-2, 2004.

Para uma descrição mais aprofundada do Scheme, recomenda-se a consulta de :

**Revised (5) Report on the Algorithmic Language Scheme**



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Nesta altura, já terá visitado **PLT Scheme**, onde encontrou o DrScheme para instalar no seu computador e muitos outros recursos para utilizar neste ambiente de programação. Entre a grande diversidade de documentação lá existente, distingue-se a publicação:

Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi

How to Design Programs e

An Introduction to Computing and Programming

The MIT Press, Cambridge, Massachusetts, 2003.

Os autores teriam uma longa lista de agradecimentos a expressar, que incluiria vários colegas e estudantes e muitos amigos. Não o fazem nem por esquecimento nem por ingratidão, mas pelo risco imenso de algum omitir, no que seria, muito provavelmente, o módulo com maior número de páginas desta publicação!...

No entanto, ao Ademar Aguiar é devida uma palavra muito especial, pelas discussões sobre o estilo mais adequado para este tipo de publicação e por se reconhecer nele o inspirador da forma que ela veio a tomar.

F. Nunes Ferreira

António Fernando Coelho

Setembro - 2009



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Resumo dos módulos 1.1 a 1.7

Resumo dos assuntos abordados nos módulos

Módulo 1.1 - **Dialogar com o Scheme através de expressões**

Módulo 1.2 - **O Scheme não pode saber tudo!**

Módulo 1.3 - **Projecto - Vamos acertar no totoloto!**

Módulo 1.4 - **O Scheme sabe trabalhar com ângulos**

Módulo 1.5 - **Tomar decisões com o Scheme**

Módulo 1.6 - **Saber lidar com os erros!**

Módulo 1.7 - **Introdução à programação através de abstracções**



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Resumo dos assuntos abordados nos módulos 1.1 a 1.7

Neste conjunto de módulos, faz-se uma introdução à linguagem Scheme, o que proporcionará o conhecimento da linguagem necessário ao desenvolvimento de pequenos programas.

### Módulo 1.1

As expressões em Scheme podem ser simples, reduzindo-se a valores constantes ou literais (um número, um booleano, um carácter ou uma cadeia de caracteres), ou compostas. As compostas utilizam uma notação pré-fixa, na qual o operador antecede os operandos. Assim, por exemplo, para somar os números 23, 4 e 33, escreve-se (+ 23 4 33).

Para além das expressões, o Scheme admite também, através da forma especial `define`, a criação de identificadores. Por exemplo, (`define identificador expressão`), associa o valor de expressão a identificador.

A regra de cálculo das expressões não se aplica à forma especial `define` nem aos identificadores criados com esta forma. Por se tratar de uma forma especial, exige também uma regra especial de cálculo. A simplicidade do Scheme está muito associada ao pequeno número de formas especiais desta linguagem.

### Módulo 1.2

A definição de procedimentos compostos é apontada como a solução para resolver situações não previstas nos procedimentos primitivos do Scheme.

Um procedimento, por si só, é apenas um pedaço de texto. Da sua chamada resulta um processo no interior do computador que vai requerer recursos computacionais para a execução da tarefa especificada no procedimento.

Na chamada de um procedimento, seja primitivo ou composto, é exigido o cálculo prévio dos operandos ou argumentos. Por exemplo, em (`remainder (* num 5) 8`), (`* num 5`) é calculado necessariamente antes de `remainder` e, por seu lado, o identificador `num` e o 5 são avaliados antes da multiplicação.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

No âmbito deste módulo, é incentivada a experiência com os procedimentos primitivos do Scheme, vocacionados para o processamento numérico.

O procedimento `random` não faz parte da especificação do Scheme. Na implementação do Scheme que utilizamos, o DrScheme, este procedimento é definido com um parâmetro `n` e a chamada (`random n`) gera um número aleatório entre 0 e `n-1`.

Um procedimento quando é chamado deve ser visto como um bloco ou caixa-preta, pois não é necessário saber como foi feito, mas apenas saber o que faz. É suficiente, portanto, conhecer a sua *interface* (nome, parâmetros e valor que devolve).

A relação dos procedimentos entre si é várias vezes posta em evidência através dos chamados diagramas de fluxos de dados.

### Módulo 1.3

Neste módulo é desenvolvido um pequeno projecto relacionado com o preenchimento de boletins de totoloto.

No desenvolvimento de um projecto é necessário: 1- perceber muito bem o problema que se pretende resolver, 2- procurar uma ideia para o resolver e traduzi-la através de uma sequência de passos, ou seja, definir o algoritmo, 3- escrever um programa que siga a sequência de passos do algoritmo e 4- testar o programa.

Deste programa apenas se espera um conjunto de mensagens no ecrã, conseguidas através de chamadas ao procedimento primitivo `display`. Este procedimento não se espera um valor de retorno, contrariamente ao que acontece com a grande maioria dos procedimentos, mas apenas um efeito colateral de visualização de entidades no ecrã. Por exemplo, (`display "Uma mensagem"`) visualiza o texto Uma mensagem e (`display (* 2 3)`) visualiza o valor numérico 6. Chamando o procedimento `newline`, a visualização passa para o início da linha seguinte.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Módulo 1.4

O processamento trigonométrico em Scheme é o objectivo deste módulo. Dos procedimentos envolvendo cálculos trigonométricos distinguem-se: `sin`, `asin`, `cos`, `acos`, `tan`, `atan`, todos envolvendo ângulos em radianos. Estes procedimentos calculam, respectivamente, seno, arco-seno, cosseno, arco-cosseno, tangente e arco-tangente.

A conversão de radianos em graus e vice-versa é disponibilizada em certas implementações do Scheme, através dos procedimentos `radians->degrees` e `degrees->radians`. Nos exemplos e exercícios desenvolvidos com o DrScheme optou-se por definir aqueles procedimentos. Optou-se igualmente por definir a constante `pi` como sendo igual a 3.141592653589793. Também há quem defina esta constante como sendo (`acos -1`).

A definição de variáveis locais, para aumentar a legibilidade e o desempenho dos procedimentos, é possível com `let`. A entrada de dados através do teclado é concretizada com o procedimento primitivo `read`.

## Módulo 1.5

As estruturas de selecção `if` e `cond` permitem a implementação de decisões sob condições. As condições simples podem ser especificadas com operadores de relação (`>`, `>=`, `<`, `<=`, `=`), e as condições compostas com os operadores lógicos (`and`, `or`, `not`). A condição composta (`(and (< num 10) (>= num 0))`) é verdadeira, se as duas condições simples forem verdadeiras, ou seja, se `num` for menor que 10 e também maior que ou igual a 0. As condições apresentam um valor booleano, que se assumirá verdadeiro, `#t`, ou falso, `#f`.

Uma expressão que devolva um booleano é designada por predicado. Em Scheme, a um procedimento que seja predicado é habitual dar-se-lhe um nome que termine com um ponto de interrogação. O próprio Scheme utiliza este tipo de designação em relação aos predicados que disponibiliza: `negative?`, `positive?`, `zero?`, `even?`, `odd?`, `boolean?`, `symbol?`, `procedure?`, `number?`, `integer?`, `real?`.

A construção `begin` permite agrupar várias expressões e esse agrupamento poderá ocupar o lugar normalmente tomado por uma única expressão. Para além disto, as várias expressões agrupadas são calculadas em sequência da primeira até à última.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

No código que se segue, a construção de selecção `if` apresenta uma expressão-consequente com várias expressões agrupadas com um `begin` e a expressão-alternativa com apenas uma expressão.

...

```
(if (>= num 0)
    (begin ; expressão-consequente
        (display "Raiz quadrada de ")
        (display num)
        (display ": ")
        (display (sqrt num)))
        (display "Valor negativo")) ; expressão-alternativa
    ...
)
```

### Módulo 1.6

Não é fácil em programação escapar aos erros, que são normalmente classificados como erros de sintaxe, erros de execução e erros lógicos.

É muito importante desenvolver manualmente um conjunto bem escolhido de situações do problema a resolver. Para além de um melhor entendimento desse problema, também resulta um conjunto de testes que ajudará, mais tarde, a encontrar os erros lógicos, já que os erros de sintaxe e os erros de execução são, em geral, detectados automaticamente pelo sistema.

### Módulo 1.7

A programação é uma actividade criativa, que se alimenta de ideias, onde, com um conhecimento mínimo de uma linguagem (neste caso, o Scheme) e através de algumas abstracções orientadas para os problemas, é possível programar situações já com alguma complexidade, nomeadamente atravessar labirintos, muitas vezes com o acompanhamento de sons a ilustrar o que vai ocorrendo em cada momento. Ao longo deste módulo encontram-se as abstrações áudio, tabuleiro, naves e janela gráfica e, pela primeira vez, são abordados procedimentos recursivos.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Exercícios e Exemplos

São apresentados alguns exercícios e exemplos para consolidação da matéria abordada ao longo dos Módulos 1.1 a 1.7.

É importante que estude os exemplos e resolva alguns dos exercícios em frente do computador, atitude que é fundamental tomar desde o início do processo de aprendizagem da programação. Enfrente os erros que surjam com naturalidade e, para cada exercício, desenvolva manualmente um conjunto de situações que ajudem a esclarecer os problemas e que sirvam, posteriormente, de teste para os programas desenvolvidos. Estudar os exemplos, introduzir-lhes alterações, inventar e resolver novos exercícios são tarefas que deverão fazer parte dos planos de quem pretende dominar a arte de bem programar.

Apesar de ainda muito limitados em termos do conhecimento das potencialidades do Scheme, é já possível resolver alguns problemas interessantes.

Para estes exercícios e exemplos recomenda-se o estudo, mesmo que ligeiro, do [\*\*Anexo A\*\*](#) (resumo dos principais procedimentos do Scheme) no que se refere às secções Processamento booleano, Processamento numérico, Processamento trigonométrico, Estruturas de selecção e Entrada/Saída.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

### Exemplo 1

Os números decimais ímpares terminam em 1, 3, 5, 7 e 9. Vamos agora inventar uma nova classe de números, os ímpares-curvos (!!!), que serão os números ímpares que terminam em 3, 5 e 9 (dígitos que, no respectivo desenho, usam segmentos curvos!).

Considere o procedimento `impar-curvo?` que determina se um número é ou não ímpar-curvo.

```
> (impar-curvo? 2345)
#t
> (impar-curvo? 2346)
#f
> (impar-curvo? 2347)
#f
```

Escreva em Scheme o procedimento `impar-curvo?` que, pelo facto de devolver um booleano, é um predicado.

A análise dos 3 exemplos de utilização do procedimento `impar-curvo?`, anteriormente indicados, ajuda a entender o que se pretende. De facto, o que é importante é o dígito menos significativo do valor numérico que o procedimento recebe.

Vamos adoptar esses 3 exemplos como o conjunto de testes a que se submeterá a solução a desenvolver.



Este conjunto de teste será o mais adequado? Justifique.

A solução que se apresenta começa por definir localmente a variável `digito-menos-significativo`, que se obtém calculando o resto da divisão inteira do número dado por 10. Seguidamente, deduz-se se o número em questão é ou não ímpar-curvo testando se `digito-menos-significativo` é 3, 5 ou 9.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(define impar-curvo?  
  (lambda (num)  
    (let ((digito-menos-signif (remainder num 10)))  
      (cond ((= digito-menos-signif 3) #t)  
            ((= digito-menos-signif 5) #t)  
            ((= digito-menos-signif 9) #t)  
            (else #f)))))
```

Uma outra solução é conseguida através de uma condição composta, que utiliza o operador lógico `or`.

```
(define impar-curvo-versao-2?  
  (lambda (num)  
    (let ((digito-menos-signif (remainder num 10)))  
      (or  
        (= digito-menos-signif 3)  
        (= digito-menos-signif 5)  
        (= digito-menos-signif 9))))))
```



Identifique a razão para se ter utilizado, nestas duas resoluções, uma variável local.



Experimente as duas versões do procedimento, submetendo-as ao conjunto de 3 testes anteriormente definidos.  
Escreva o procedimento `impar-curvo-minha-versao?` e submeta-o ao conjunto de 3 testes.  
Esta versão segue uma pista diferente das outras 2 versões, pois irá testar os dígitos 1 e 7, em vez dos dígitos 3, 5 e 9...



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

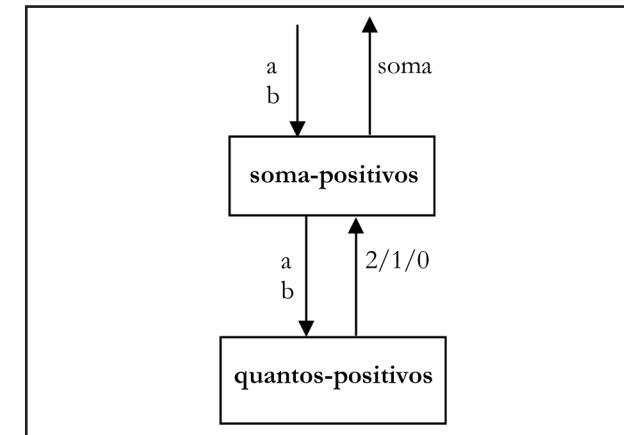
### Exemplo 2

Suponha que já dispõe do procedimento quantos-positivos, com dois parâmetros. Este procedimento analisa os argumentos que recebe e fornece como resultado 2, 1 ou 0, conforme o número de argumentos positivos.

```
> (quantos-positivos 1 2)
2
> (quantos-positivos 1 -2)
1
> (quantos-positivos -1 -1)
0
```

Por seu turno, o procedimento soma-positivos também analisa os dois argumentos que recebe:

- Se ambos forem positivos, devolve a soma dos dois;
- Se apenas um for positivo, devolve o valor desse positivo;
- Se ambos forem negativos, devolve o valor 0.



Escreva em Scheme o procedimento soma-positivos, utilizando o procedimento quantos-positivos, que se supõe já existente.

Comece por desenvolver, manualmente, quatro situações de funcionamento do procedimento soma-positivos.

- Com os argumentos 3 e 2, como ambos são positivos, a resposta será 5.
- Com 3 e -2, só temos um positivo, o número 3, e a resposta é 3.
- Com -3 e 2, só temos um positivo, o número 2, e a resposta é 2.
- Com -3 e -2, temos zero positivos, logo a resposta é zero.

Optou-se por criar localmente a variável numero-de-positivos, que vai conter o número de positivos, através de uma chamada ao procedimento quantos-positivos.

- Com dois positivos, a resposta é a soma dos dois números.
- Com zero positivos, a resposta é zero.
- Com um positivo, verifica-se qual deles é o positivo, pois a resposta será esse número

```
(define soma-positivos
  (lambda (a b)
    (let ((numero-de-positivos (quantos-positivos a b)))
      (cond ((= numero-de-positivos 2) (+ a b))
            ...)))
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
((zero? numero-de-positivos) 0)
((positive? a) a)
(else b))))
```



Sem a ajuda do Scheme, indique quais seriam as respostas do procedimento soma-positivos às 4 situações de teste preparadas manualmente se, neste procedimento, em (= numero-de-positivos 2), em vez de 2, por engano, fosse escrito 3.

É agora o momento de tratar do procedimento quantos-positivos que se baseia nos seguintes testes: x e y são ambos positivos; x e y são ambos negativos; se não for nenhum destes casos, então um dos argumentos será positivo.

```
(define quantos-positivos
  (lambda (x y)
    (cond ((and
              (positive? x)
              (positive? y)) 2)
          ((and
              (negative? x)
              (negative? y)) 0)
          (else 1))))
```



Em quantos-positivos, na última linha, imagine que, por engano, em vez de 1 foi escrito 3. Sem a ajuda do Scheme, indique quais seriam as respostas do procedimento soma-positivos às 4 situações de teste preparadas manualmente.



Experimente o procedimento soma-positivos e o procedimento auxiliar quantos-positivos, submetendo-os ao conjunto dos 4 testes já preparados.

O procedimento quantos-positivos deverá ser testado em primeiro lugar. Porquê?



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

### Exemplo 3

Para este exemplo, deve supor que a Terra é uma esfera perfeita, com um raio de 6378 Km. Assim, o perímetro da Terra, ao percorrer o Equador (latitude 0°), será  $2 * \pi * 6378000$  metros. O objectivo é conhecer o perímetro da Terra, quando se percorre um paralelo ao Equador, identificado pela respectiva latitude (entre 0 e 90°, no Hemisfério Norte).

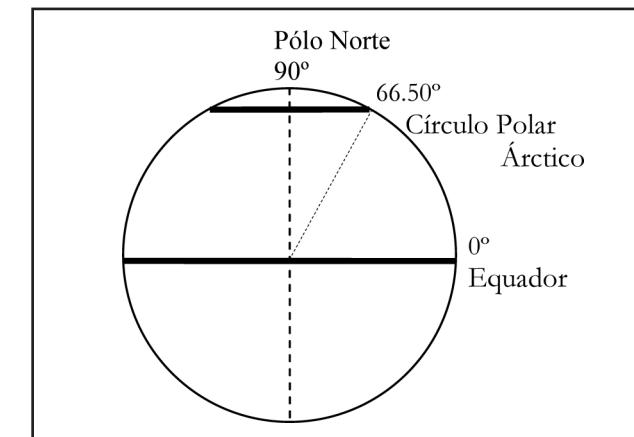
Se, para a latitude 0°, o perímetro coincide com o comprimento do Equador, para a latitude 90°, que coincide com o pólo Norte, o perímetro será, teoricamente, 0 metros.

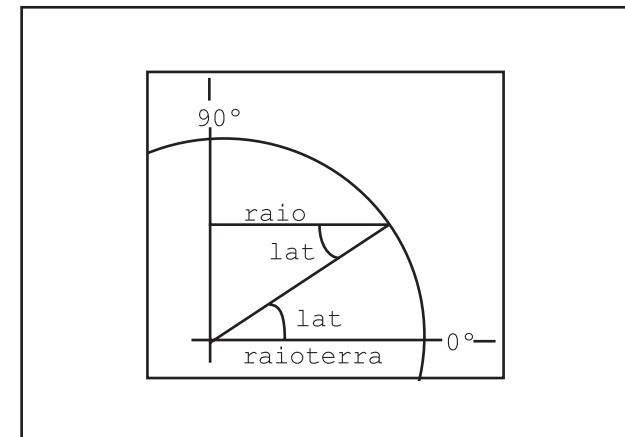
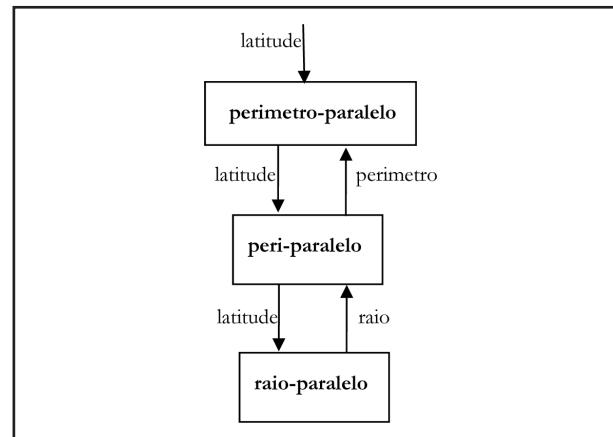
Mas qual será o perímetro do trópico de Câncer, sabendo que se encontra à latitude 23.5° N? E o perímetro do Círculo Polar Ártico, a 66.5° N? E o perímetro de outro paralelo qualquer?

Pretende-se definir o procedimento `perimetro-paralelo` que recebe, como argumento, a latitude de um paralelo do Hemisfério Norte e devolve o perímetro desse paralelo. O tipo de resposta do procedimento é como se indica.

```
> (perimetro-paralelo 0)  
40074155.889191404 metros  
> (perimetro-paralelo 66.5)  
15979532.348780245 metros  
> (perimetro-paralelo 90)  
2.453753296298615e-009 metros  
> (perimetro-paralelo 89.5)  
349708.5439343981 metros
```

Na resolução deste problema, como pode ver pelo diagrama de fluxo de dados, principiámos pelo mais geral, com o procedimento `perimetro-paralelo` que trata apenas da mensagem da resposta.





Este procedimento recorre ao procedimento `peri-paralelo` que calcula o perímetro de um paralelo para uma certa latitude.

Observando um pouco mais em profundidade, concluiremos que, por sua vez, peri-paralelo recorre ao procedimento raio-paralelo que devolve o raio do paralelo para uma latitude.

A figura mostra como, a partir do raio da Terra e da latitude, se pode calcular o raio do paralelo. Tendo em conta o triângulo rectângulo desenhado na figura, sabe-se que um cateto,  $raio$ , é igual à hipotenusa, neste caso coincidindo com o raio da terra, multiplicada pelo cosseno do ângulo adjacente,  $\text{lat}$ .

Podemos começar por desenvolver um conjunto de situações de funcionamento do procedimento *raio-paralelo*.

Para uma certa latitude em radianos, uma vez que as funções trigonométricas no Scheme funcionam em radianos, `raio = raio-da-terra*coseno(latitude)`. Ou seja, abaixo do procedimento `raio-paralelo`, ainda aparece um procedimento que converte graus em radianos, designado por `degrees->radians`.

Sabendo que a  $\pi$  radianos corresponde 180 graus,  $\text{ang-radianos} = \text{ang-graus} * \pi / 180$ .

Assim, considerando  $\pi=3.141592653589793$ ,

- para  $\text{ang-graus} = 66.5$ ,  $\text{ang-radianos} = 66.5 \cdot 3.141592653589793/180 = 1.160644$ .
  - para  $\text{ang-graus} = 89.5$ ,  $\text{ang-radianos} = 89.5 \cdot 3.141592653589793/180 = 1.56207$ .
  - para  $\text{ang-graus} = 90.0$ ,  $\text{ang-radianos} = 90.0 \cdot 3.141592653589793/180 = 1.570796$ .

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Passando, de seguida, à definição e teste do procedimento degrees->radians.

```

;
; Procedimento que converte graus em radianos
;
(define degrees->radians
  (lambda (ang)
    (/ (* pi ang)
       180)))
;
(define pi 3.141592653589793) ; valor de pi

> (degrees->radians 66.5)
1.160643952576229
> (degrees->radians 89.5)
1.562069680534925
> (degrees->radians 90.0)
1.5707963267948966
>
```

A menos de alguns arredondamentos, os resultados são muito semelhantes aos obtidos manualmente.

Agora estamos em condições para passar ao procedimento raio-paralelo, para o qual vamos desenvolver algumas situações de teste.

Retomando o raciocínio anterior, para uma certa latitude,  $raio = raio\text{-da-terra} \cdot \cos(\text{latitude})$ .

- Para latitude= 0°= 0 radianos,

$$raio = 6378000 \cdot \cos(0) = 6378000 \cdot 1 = 6378000$$

- Para latitude= 66.5°= 1.160643952576229 radianos

$$raio = 6378000 \cdot \cos(1.160643952576229) = 6378000 \cdot 0.3987490689 = 2543221.5614442$$

- Para latitude= 89.5°= 1.562069680534925 radianos

$$raio = 6378000 \cdot \cos(1.562069680534925) = 6378000 \cdot 0.0087265355 = 55657.843419$$

- Para latitude= 90°

$$raio = 0 \text{ (ou muito próximo de zero, tendo em conta os arredondamentos)}$$



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Passando, de seguida, à definição e teste do procedimento `raio-paralelo`.

```
; calcula o raio do paralelo cuja latitude, em graus, é o argumento
;
(define raio-paralelo
  (lambda (lat)
    (* raio-terra
       (cos (degrees->radians lat)))))

(define raio-terra 6378000) ; raio da Terra em metros
;

> (raio-paralelo 0)
6378000
> (raio-paralelo 66.5)
2543221.5616052207
> (raio-paralelo 89.5)
55657.84340862871
> (raio-paralelo 90.0)
3.905269662339561e-010
>
```

Também neste caso, se não atendermos a alguns arredondamentos, os resultados são muito semelhantes aos obtidos manualmente.

Faltaria agora desenvolver os procedimentos `peri-paralelo` e `perímetro-paralelo`.

O primeiro recebe a latitude de um paralelo e determina o seu perímetro. Recorre a `raio-paralelo` para obter o raio do paralelo, que multiplica por  $2\pi$ , calculando assim o perímetro. O segundo, como já foi dito, limita-se a visualizar no ecrã uma mensagem adequada, tomando por base o valor que lhe é fornecido pelo procedimento `peri-paralelo`.

Estes dois procedimentos poderiam ser testados em conjunto, com base nas situações apresentadas inicialmente no enunciado, supondo que a correcção das mesmas foi previamente confirmada.

```
; visualiza o resultado no formato indicado
;
(define perimetro-paralelo
  (lambda (latitude)
    (display (peri-paralelo latitude))
    (display " metros")
    (newline)))
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
;  
; calcula o perímetro do paralelo cuja latitude é o argumento  
;  
(define peri-paralelo  
  (lambda (latitude)  
    (* 2  
      pi  
      (raio-paralelo latitude))))
```



Experimente o procedimento `perímetro-paralelo` e os procedimentos auxiliares `peri-paralelo`, `raio-paralelo` e `degrees->radians`.

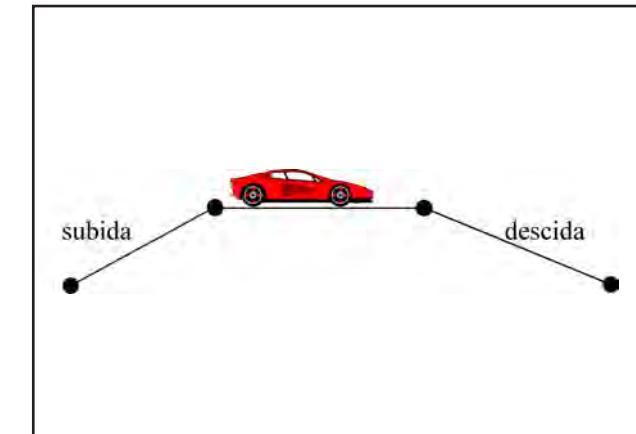
### Exercício 1

Um percurso rodoviário é composto por uma parte em piso horizontal, mas também apresenta uma subida e uma descida. De uma viatura é conhecido o consumo médio aos 100 Km, quando se desloca em percurso horizontal.

Em relação ao consumo, a mesma viatura, em subida, gasta mais 30% e, em descida, menos 10%.

Escreva em Scheme o procedimento `consumo-total` que tem como parâmetros `consumo`, `horiz`, `sub` e `desc`, que representam, respectivamente, o consumo médio da viatura em percurso horizontal, o número de Km de percurso horizontal, o número de Km em subida e, finalmente, o número de Km em descida. Este procedimento devolve a quantidade em litros gasta pela viatura no percurso definido pelos parâmetros respectivos.

Esboce um diagrama de fluxo de dados que coloque em evidência a ligação entre os procedimentos utilizados e desenvolva, manualmente, situações de teste para todos eles.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Desenvolva e teste o procedimento consumo-total.

### Exercício 2

Tomando como base o exercício anterior, escreva um programa em Scheme que tem como parâmetros gas-no-tanque (que representa a quantidade de combustível que a viatura tem no tanque), e ainda com os mesmos parâmetros do procedimento consumo-total. O programa pretendido responde, conforme o caso, com uma das seguintes mensagens:

- Nao é suficiente. Faltam x litros.
- É suficiente. Sobram x litros.
- Gasolina 'a justa.

Esboce um diagrama de fluxo de dados que coloque em evidência a ligação entre os procedimentos utilizados e desenvolva, manualmente, situações de teste para todos eles.



Desenvolva e teste o procedimento pedido.

### Exercício 3

No contexto do Exemplo 3 (cujo tema é o cálculo do perímetro de um paralelo ao Equador conhecida a respectiva latitude), suponha que se colocava à volta da Terra, sobre o Equador, um anel de arame com 40074155.889191404 metros. O anel ficava justo à Terra, portanto à distância zero, em todos os seus pontos. Imagine agora que cortava o anel e que lhe acrescentava um metro. O anel era novamente colocado à volta da Terra, no Equador, mas já não ficava completamente justo.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Se o anel fosse colocado concêntrico com o Equador, a que distância ficaria o anel da Terra? Ou de outra forma, qual a diferença entre o raio do anel, *raio-ext*, e o raio do Equador, *raio-int*?

E se uma experiência idêntica fosse feita num paralelo, muito junto ao Pólo Norte, com um arame igual ao seu perímetro, ao qual se acrescentaria também um metro. Neste caso, qual seria a diferença entre os raios do anel e do paralelo?

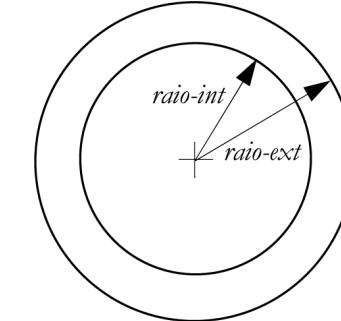
O procedimento *diferenca-raios* com dois parâmetros, um associado à latitude e outro ao comprimento de arame a acrescentar ao anel, calcula e devolve a diferença entre o raio do anel de arame, *raio-ext*, e o raio do paralelo identificado pela latitude, *raio-int*, quando se lhe acrescenta aquele comprimento de arame.

Sugere-se a seguinte pista:

- Para a latitude dada, determine o raio do respectivo paralelo, *raio-int*
- Para a latitude dada, determine o perímetro do respectivo paralelo, uma vez que conhece *raio-int*. Adicione a este perímetro o comprimento a acrescentar
- Determine o raio correspondente ao perímetro aumentado, *raio-ext*
- Ache a diferença entre *raio-ext* e *raio-int*

Escreva em Scheme o procedimento *diferenca-raios* e experimente, entre outras, as seguintes situações:

```
> (diferenca-raios 0 1)      ; acrescentar 1 metro no anel do Equador
???
> (diferenca-raios 23.5 1)   ; acrescentar 1 metro no anel
                               ; do Trópico de Câncer
???
> (diferenca-raios 66.5 1)   ; acrescentar 1 metro no anel
                               ; do Círculo Polar Ártico
???
> (diferenca-raios 90 1)     ; acrescentar 1 metro no anel do Pólo Norte
???
```



Esboce o diagrama de fluxo de dados que coloque em evidência a ligação entre os procedimentos utilizados e desenvolva, manualmente, situações de teste para todos eles.



Desenvolva e teste o procedimento diferença-raios.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



A análise dos resultados obtidos vai certamente conduzir a uma conclusão, que poderá ser considerada muito estranha... Indique essa conclusão.

### Exercício 4

Para este exercício vai ser necessário relembrar, embora de uma forma um pouco simplificada, algumas fórmulas do movimento de projécteis no espaço. Tome em atenção as figuras que se seguem.

Um projéctil é lançado de um ponto  $(x_0, y_0)$ , com a velocidade inicial  $v_0$  e o ângulo de lançamento  $\text{ang}$ . A velocidade inicial decomponde-se em  $v_{0x}$  e  $v_{0y}$ .

A velocidade  $v$  do projéctil, em cada momento, decomponde-se em  $v_x$  e  $v_y$ . Despreza-se o atrito do ar quando o projéctil se move, supondo-se por isso que a velocidade  $v_x$  é constante, enquanto  $v_y$  sofre apenas a influência permanente da gravidade ( $g = 9.75 \text{ m/s}^2$ ). Assim,  $v_y$  aponta para cima no início, vai diminuindo sucessivamente até atingir o valor zero. Isto acontece no ponto em que o projéctil atinge a altura máxima. A partir deste ponto,  $v_y$  começa a apontar para baixo e a aumentar por influência da gravidade.

As equações  $x$  e  $y$  definem a posição do projéctil, em função do ponto inicial, da velocidade inicial, do tempo e da gravidade.

$$v_{0x} = v_0 \cdot \cos(\text{ang})$$

$$v_{0y} = v_0 \cdot \sin(\text{ang})$$



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

$$v_x = v_{0x}$$

$$v_y = v_{0y} - g \cdot t$$

$$x = x_0 + v_{0x} \cdot t$$

$$y = y_0 + v_{0y} \cdot t - 0.5 \cdot g \cdot t^2$$

$$g = 9.75 \text{ m/s}^2$$

No conjunto de exercícios que se segue, considere que o ponto de lançamento está na origem do sistema de eixos, ou seja,  $x_0 = y_0 = 0$ .

1- Escreva em Scheme o procedimento `distancia-max`, com os parâmetros `vel-inicial` e `ang-inicial`, correspondentes, respectivamente, à velocidade e ao ângulo iniciais do lançamento do projétil. O procedimento devolve a distância máxima alcançada pelo projétil.

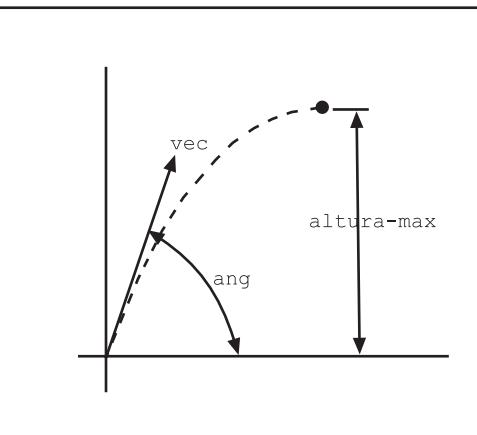
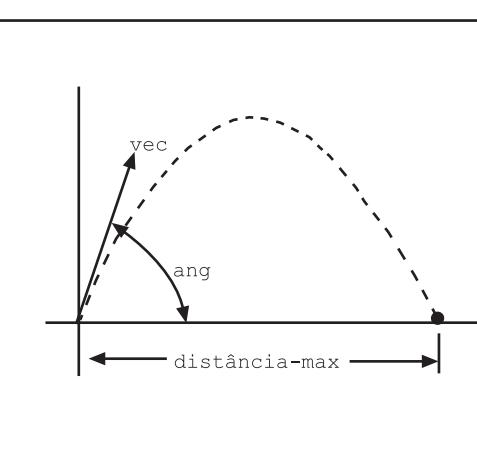
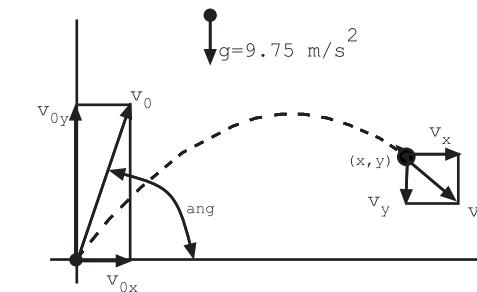
Pista sugerida:

- Com a expressão  $y$  determine  $t$  em que  $y = 0$
- Determine  $x$ , com a respectiva expressão, para o  $t$  determinado

2- Escreva em Scheme o procedimento `altura-max`, com os parâmetros, `vel-inicial` e `ang-inicial`, correspondentes, respectivamente, à velocidade e ao ângulo iniciais do lançamento do projétil. O procedimento devolve a altura máxima alcançada pelo projétil.

Pista sugerida:

- Com  $v_y$  determine  $t$  em que  $v_y = 0$
- Determine  $y$ , com a respectiva expressão, para o  $t$  determinado



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



3- Escreva em Scheme o procedimento `posicao`, com três parâmetros, `vel-inicial`, `ang-inicial` e `tempo`, correspondentes, respectivamente, à velocidade e ao ângulo iniciais do lançamento de um projétil e ao intervalo de tempo contado a partir do lançamento.

O procedimento, através de uma mensagem apropriada, visualiza as coordenadas x e y da posição onde se encontra o projétil no final daquele intervalo de tempo.

Pista sugerida:

- Calcule  $v_{0x}$  e  $v_{0y}$ , através de  $v_0$  e  $\text{ang}$
- Determine x e y para o t fornecido

4- Com base nos procedimentos desenvolvidos, realize algumas experiências:

- Com várias chamadas de `distancia-max` e para uma certa velocidade inicial, descubra qual o ângulo que permite o lançamento mais longo.
- Com várias chamadas de `altura-max` e para uma certa velocidade inicial, descubra qual o ângulo que permite o lançamento mais alto.
- Com várias chamadas de `posicao`, mantendo fixos a velocidade e o ângulo iniciais e fazendo variar o tempo, obtenha os dados necessários para desenhar, em papel quadriculado, a trajectória do projétil.



Desenvolva e teste o trabalho proposto.

## INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Módulo 1.1 - Dialogar com o Scheme através de expressões

O Scheme apresenta-se num ciclo permanente de leitura-cálculo-escrita de expressões, ou seja, está sempre disponível para ler expressões, calculá-las e visualizar o respectivo resultado.

Neste módulo, pretende-se, essencialmente, mostrar como se escrevem as expressões em Scheme, através de operadores aritméticos e números, e apresentar a regra de cálculo respectiva.

Também se mostrará como se cria um nível de abstracção que se situa acima dos números, possibilitando a substituição destes por entidades com nomes legíveis, designadas por identificadores. A propósito da regra de constituição dos nomes dos identificadores, é apresentada a notação BNF (Backus-Naur Form).

As formas do Scheme, que estão na base da definição da linguagem, são as expressões e a definição de identificadores.

As formas do Scheme, vistas de outro ângulo, aparecem como formas normais, que seguem uma regra semelhante para todas, e formas especiais, que requerem regras específicas para cada uma delas.

É incentivado o uso do Scheme na exploração de situações propostas ou outras criadas pelo próprio leitor, para que, no final do módulo, esta matéria esteja dominada de forma consistente.

### Palavras-Chave

Ciclo de leitura-cálculo-escrita, constante ou literal, número, booleano, carácter, cadeia de caracteres, expressão simples, expressão composta, interpretador de scheme, *prompt*, procedimento primitivo, operador aritmético, notação pré-fixa, notação in-fixa, regra de cálculo de expressões compostas, forma especial *define*, identificador, regra de cálculo de *define*, regra de cálculo de um identificador, notação BNF (Backus-Naur Form).



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Expressões simples

Os números são expressões simples, na linguagem Scheme, que representam o seu próprio valor na base de numeração 10. Por exemplo, o número 15, quando avaliado, vale 15, como seria de esperar. O número 35E2 é a representação exponencial do valor 3500, pois equivale a  $35 \times 10^2$  (o expoente 2 é o número à direita de E), enquanto que o número 35E-3 representa o valor 0.035, por ser equivalente a  $35 \times 10^{-3}$ .

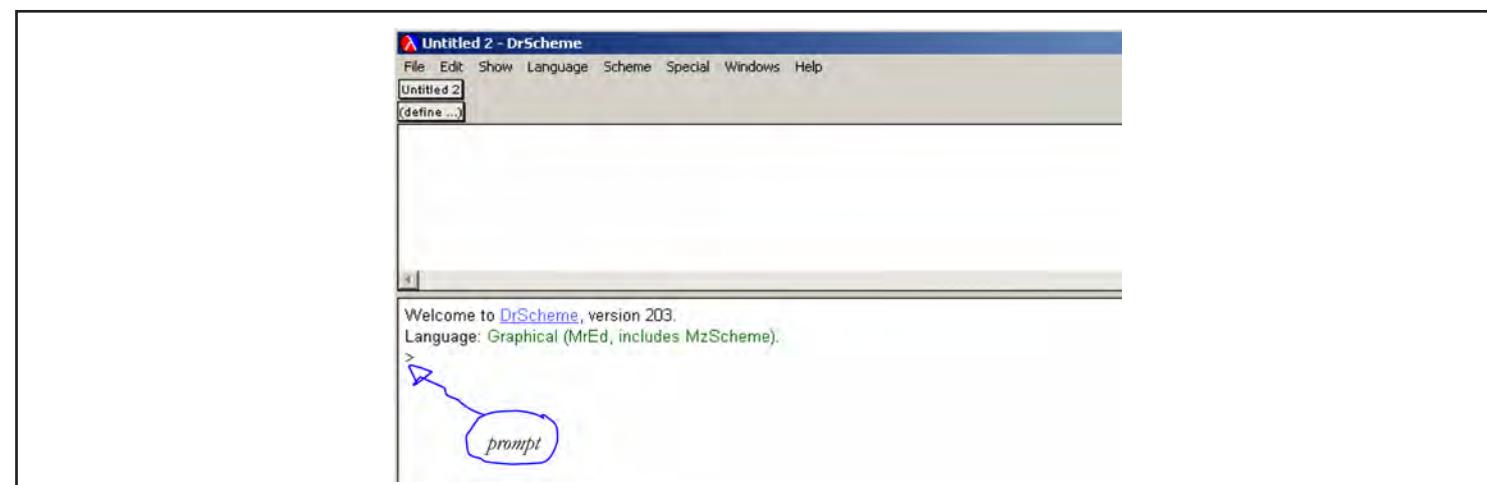
A partir deste momento, mas de uma forma muitíssimo limitada, já pode dialogar com o computador, através da linguagem Scheme. Para isso, torna-se necessário pôr em funcionamento um programa que entenda a sintaxe desta linguagem e que poderá ser um interpretador de Scheme.



**Se está disposto a enfrentar este desafio, vai ter que começar por instalar uma implementação do Scheme no seu computador. Combinado?**

**Se ainda não instalou o Scheme no seu computador, poderá fazê-lo com alguma ajuda indicada na [Introdução](#).**

O interpretador de Scheme, normalmente num ciclo permanente de leitura-cálculo-escrita, está preparado para ler uma expressão, calculá-la e visualizar o respectivo resultado. O Scheme identifica esta situação através de um carácter especial (vulgarmente denominado *prompt*), visualizado no ecrã.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Os dois exemplos que se seguem são expressões simples, que envolvem apenas um número.

```
> 5
5
> 5.6
5.6
>
```

Em Scheme, os números na base 10 (e também na base 2, 8 e 16) são valores constantes ou literais, como também o são os booleanos (#t para o valor verdadeiro e #f para o valor falso), os caracteres (#\a para o carácter a, ... #\z para o carácter z, #\space para o carácter espaço, #\newline para o carácter nova-linha,...) e as cadeias de caracteres ("programar" para a cadeia de caracteres programar).

Seguem-se mais alguns exemplos de expressões simples que envolvem valores literais não numéricos.

```
> #t
#t
> #\b
#\b
> #\space
#\space
> "programar"
"programar"
>
```

### Expressões compostas

Antes de experimentar situações de expressões criadas por si, deve analisar mais alguns exemplos, agora de expressões compostas, que utilizem os 4 operadores aritméticos: adição, subtração, multiplicação e divisão. Estes operadores do Scheme são alguns dos procedimentos primitivos da linguagem, operadores que o Scheme entende sem qualquer ajuda adicional.

```
> (+ 4 5.5)
9.5
> (- 4 5.5)
-1.5
> (* 4 5.5)
22.0
> (/ 4 5.5)
0.7272727272727273
>
```





Experimente expressões em Scheme compostas apenas por um operador e dois operandos numéricos.

## INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Nesta altura, já terá identificado e enfrentado uma característica das expressões compostas, que não lhe será muito habitual.



**É capaz de adiantar a característica de que estamos a falar?**

Apenas se adianta que essa característica está associada à chamada notação pré-fixa utilizada pelo Scheme. A outra notação, que certamente já conhece, é designada por in-fixa.

Para dominar adequadamente as expressões, é importante que experimente ainda exemplos que envolvam:

1. mais do que dois operandos, equivalente na notação in-fixa a  $5 + 6 + 6.7 + 0.4$

```
> (+ 5 6 6.7 0.4)  
18.099999999999998
```

2. mais do que um operador aritmético, equivalente na notação in-fixa a  $5 + 45 * 2$

```
> (+ 5 (* 45 2))  
95
```

3. operandos que são números em notação exponencial

```
> 35E2  
3500.0  
> 35E-2  
0.35
```



Experimente expressões em Scheme que envolvam um ou mais operadores.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



### Exercício 1

Analise a situação que se segue:

```
> (+ 12 (- 45 38) 2.5)  
21.5
```

e, sobre ela:

- identifique os operandos da adição;
- tenha em consideração que o segundo operando é, ele próprio, uma operação de subtração que exige a aplicação prévia da regra de cálculo;
- aplique manualmente a regra de cálculo à expressão e compare com o resultado indicado.

### Exercício 2

Calcule manualmente as expressões que se indicam e, seguidamente, verifique com o Scheme a correcção das suas respostas.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (* (+ 34 30) 6)
?
> (/ 4 3 2.0)
?
> (+ 23 (* 3 5 3) (- 22 2 34))
?
> (* (/ 34 (- 4 5)) (+ 12 (- 15 3)))
?
```



Experimente as expressões do Exercício 2.

Enfrentou certamente algumas dificuldades na leitura da última expressão do Exercício 2. Verifique que o editor do Scheme admite a formatação das expressões de outra maneira, permitindo o alinhamento vertical dos operandos de cada um dos operadores da expressão.

Por exemplo, aquela expressão poderá então apresentar uma forma muito mais legível.

```
> (* (/ 34
         (- 4 5))
      (+ 12
         (- 15 3)))
```

Os dois operandos da multiplicação foram postos em destaque.



Tente também destacar os operandos da divisão e da adição...



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Identificadores

Alguma coisa de novo, relativamente às expressões anteriormente apresentadas, surge no exemplo que se segue.

```
> (define lado-da-casa 16)
> lado-da-casa
16
> (* 5 lado-da-casa)
80
```

Já descobriu? Certamente que sim. O identificador `lado-da-casa` foi definido, tendo-lhe sido associado o valor 16.

Na expressão que se segue, o resultado é o mesmo que na expressão anterior, mas, em termos de legibilidade da expressão, apenas ficamos a saber que se multiplicam os números 5 e 16.

```
> (* 5 16)
80
```

Mais um exemplo que deve analisar.

```
> (* 5 lado-casa)
reference to undefined identifier: lado-casa
```



Que conclusão retirou dessa análise?



Experimente expressões com identificadores.

**Pista:** defina identificadores com nomes adequados para os lados de um rectângulo e, com base neles, defina um identificador correspondente à área desse rectângulo.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Acabámos de utilizar `define`, uma forma especial do Scheme, que se representa agora genericamente por:

```
(define identificador expressão)
```

Cada forma especial, como o próprio nome indica, tem no Scheme uma regra de cálculo especial, que no caso de `define` será:

1. calcula a expressão;
2. associa o valor de expressão a identificador.

A regra de cálculo de um identificador resume-se a:

1. Se um identificador está associado a um valor, é devolvido esse valor;
2. Caso contrário, é assinalada uma mensagem de erro.

A regra de constituição dos nomes dos identificadores está ainda em falta, mas antes dela, um pormenor de grande importância. O nome do identificador deve ser muito bem escolhido, de tal forma que, numa simples leitura, se alcance uma ideia clara do que ele representa. Por exemplo, o nome `lado-da-casa`, num certo contexto, deverá ser suficientemente significativo para quem o lê.

A regra de constituição dos nomes dos identificadores é apresentada em [notação BNF](#) (Backus-Naur Form).

```
<identificador> -> <inicial> <subsequente>*
<inicial> -> <letra> | <inicial especial> | <identificador peculiar>
<letra> -> a | b | c | ... | z
<inicial especial> -> ! | $ | % | & | * | / | : | < | = | > | ? | ~ | _ | ^
<subsequente> -> <inicial> | <dígito> | <subsequente especial>
<dígito> -> 0 | 1 | 2 | ... | 9
<subsequente especial> -> . | + | - | @
<identificador peculiar> -> + | - | ...
```

Na notação BNF:

- entre os caracteres `<` e `>` colocam-se os elementos não terminais, isto é, os elementos que não estão completamente definidos.  
Por exemplo, os dígitos decimais: `<dígito> -> 0 | 1 | 2 | 3 | ... | 9`
- o carácter `|` representa alternativas;
- o carácter `*` após um elemento não terminal significa que esse elemento pode aparecer zero ou mais vezes;

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

- o carácter + após um elemento não terminal (que não é utilizado nesta definição) significa que esse elemento pode ser repetido uma ou mais vezes.

### Exercício 3

Tendo em conta a regra de constituição dos nomes dos identificadores apresentada, indique os identificadores correctos e os incorrectos na lista que se segue, tendo o cuidado de justificar a sua opção.

x % a67 a6%7d a6#7 %a67 -abc lado-inf 5rty



Algumas implementações do Scheme são muito permissivas e admitem como correctos identificadores que não estão de acordo com a regra apresentada. Não se surpreenda se assim acontecer com o DrScheme...

A propósito, duas notas:

- 1- Os identificadores cujos nomes sigam esta regra devem ser aceites como correctos em qualquer implementação do Scheme.
- 2- A regra que se apresentou traz consigo um outro benefício: faz uma breve introdução à notação BNF, que irá encontrar, muitas vezes, no estudo e consulta de documentação no domínio da informática.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Módulo 1.2 - O Scheme não pode saber tudo!

A linguagem Scheme está preparada para interpretar e executar um certo conjunto de funcionalidades básicas, designadas por procedimentos primitivos. Normalmente, somos confrontados com a necessidade de preparar o Scheme para entender outras funcionalidades para além das citadas, o que se consegue através da escrita de procedimentos compostos.

Neste módulo, principia-se por mostrar o que o Scheme sabe fazer através dos seus procedimentos primitivos, sobretudo no que se refere ao processamento numérico. Depois, mostra-se como o Scheme resolve os nossos problemas através de procedimentos compostos.

Outro aspecto importante apresentado é a noção de processo. Um procedimento é um pedaço de texto, enquanto que processo é o que resulta da chamada do procedimento. O processo é criado no interior do computador para realizar uma determinada tarefa, obviamente gastando recursos computacionais (espaço de memória e tempo de processador).

Um procedimento é normalmente composto recorrendo a outros procedimentos. O diagrama de fluxo de dados mostra a relação estabelecida entre os procedimentos utilizados e põe em evidência os dados trocados entre eles.

O módulo encerra com a noção de procedimento visto como bloco ou caixa-preta.

Incentiva-se a utilização do Scheme para exploração das situações propostas ou outras criadas pelo próprio leitor, para que, no final do módulo, a matéria fique plenamente dominada.

### Palavras-Chave

Procedimento primitivo, notação pré-fixa, random, definição de procedimento composto, forma especial lambda, chamada de um procedimento composto, processo, parâmetro, argumento, diagrama de fluxo de dados, procedimento como caixa-preta, *interface* de um procedimento.



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## O que o Scheme sabe - procedimentos primitivos

A linguagem Scheme foi criada para interpretar e executar um conjunto de funcionalidades básicas, os procedimentos primitivos, resumidamente descritos no [Anexo A](#). Destes procedimentos primitivos destacam-se agora, fundamentalmente, os que se destinam ao processamento numérico.

```
> (+ 34 6 78)  
118  
> (sqrt 16)  
4  
>
```

O Scheme utiliza expressões em notação pré-fixa, colocando entre parêntesis, em primeiro lugar, o nome do operador, seguido dos operandos.



Experimente expressões que envolvam procedimentos primitivos do Scheme, essencialmente os que se destinam ao processamento numérico ([Anexo A](#)).

### Exercício 1

Com a ajuda do [Anexo A](#), indique as respostas às seguintes expressões e confirme depois, com a ajuda do próprio Scheme, a correcção das suas respostas.

> (add1 34)	> (sub1 16)	> (sub1 16 2)
??	??	??
> (expt 2 4)	> (expt 2 5)	> (abs 5.3)
??	??	??
> (abs -5.3)	> (round 16.3)	> (round 16.5)
??	??	??
> (round 17.5)	> (round -17.5)	> (truncate 16.3)
??	??	??
> (truncate 16.9)	> (ceiling 16.3)	> (ceiling 16.9)
??	??	??

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (floor 16.3)          > (floor 16.9)          > (max 6 2 -15.2 9 1.7)  
??                      ??                      ??  
> (min 6 2 -15.2 9 1.7) > (quotient 16 3)    > (remainder 16 3)  
??                      ??                      ??  
> (gcd 16 4 12)        > (lcm 16 4 12)     > (random 5)  
??                      ??                      ??  
> (even? 2)            > (even? 3)          > (odd? 2)  
??                      ??                      ??  
> (number? 3.5)        > (integer? 3.5)    > (real? 3.5)  
??                      ??                      ??  
> (real? 3.0)          > (real? 3)          > (number? "2")  
??                      ??                      ??  
> (real? (* 2 3.5))   > (integer? (* 2 3.5)) > (integer? (* 3 3.5))  
??                      ??                      ??
```



Confirme as respostas relativas às expressões do Exercício 1.



No contexto do Scheme, os procedimentos cujo nome termina em "?" são normalmente predicados.  
Que tipo de valores são devolvidos pelos predicados?

Se as experiências que fez tiverem avançado até ao procedimento `random`, deve verificar no [Anexo A](#) uma anotação muito pouco agradável, ou seja, nem todas as implementações do Scheme disponibilizam este procedimento primitivo com a mesma sintaxe!



Poderá antever que tipo de problemas ocorrerão pelo facto das implementações de uma linguagem não seguirem uma norma estabelecida para todos os procedimentos primitivos?



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

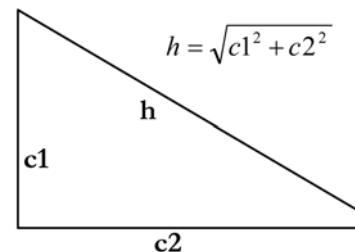
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Programar o Scheme para resolver os nossos problemas

Quando não encontrar procedimentos primitivos que resolvam directamente o problema que tem pela frente, a saída é criar novos procedimentos, compostos por procedimentos primitivos ou até por outros procedimentos entretanto já criados. São os designados procedimentos compostos.

Suponha que o problema que pretende resolver é o cálculo da hipotenusa de um triângulo rectângulo, segundo o teorema de Pitágoras (o quadrado da hipotenusa é igual à soma dos quadrados dos catetos). Em primeiro lugar, confirme no **Anexo A** que não encontra nenhum procedimento primitivo para resolver este problema.



Já agora, pode adiantar uma ideia que justifique a não existência de um procedimento primitivo para calcular a hipotenusa de um triângulo rectângulo?

Não existindo procedimentos primitivos para calcular a hipotenusa, só lhe resta criar um procedimento com esse objectivo e com um nome sugestivo, que poderá ser exactamente hipotenusa.

Antes de tentar a escrita do procedimento hipotenusa, convém lembrar que o Scheme, através da forma especial define, associa o valor de uma expressão a um nome, criando assim um identificador. É isso o que acontece com os nomes altura e base nos exemplos que se seguem.

```
> (define altura 3)
> (define base (- (* 2 altura) 1))
> altura
3
> base
5
>
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Mas voltemos ao procedimento hipotenusa, que é o que nos interessa, neste momento. Este toma dois valores numéricos, que representam os comprimentos dos dois catetos de um triângulo rectângulo, e devolve o valor do comprimento da sua hipotenusa. Com isto em mente, analise cuidadosamente a figura que se segue.

The screenshot shows the DrScheme interface. In the top-left corner, there's a red icon with a white letter 'A'. The window title is 'Untitled - DrScheme'. The menu bar includes 'File', 'Edit', 'Show', 'Language', 'Scheme', 'Special', 'Windows', and 'Help'. A toolbar below the menu has buttons for 'Untitled' (highlighted), 'Save', '(define ...)', '(lambda ...)', '(lambda ...)', '(lambda ...)', and '(lambda ...)'. The main code area contains:

```
(define hipotenusa ; nesta linha, é definido o nome do procedimento
  (lambda (c1 c2) ; aqui estão c1 e c2, os parâmetros do procedimento
    (sqrt (+ (* c1 c1) ; aqui começa o corpo do procedimento, onde se
              (* c2 c2))))) ; define o cálculo que se associa ao procedimento
```

A blue callout bubble points to the word 'lambda' with the text 'mais uma forma especial'.

Below the code, the DrScheme welcome message and language information are displayed:

```
Welcome to DrScheme, version 203.
Language: Graphical (MrEd, includes MzScheme).
> (define altura 3)
> (define base (- (* 2 altura) 1))
> altura
3
>
```

Na definição de novos procedimentos recorremos a mais uma forma especial do Scheme, lambda, que não devolve um valor numérico, como acontece com os operadores aritméticos, mas devolve, isso sim, um procedimento associado a um cálculo bem definido. Foi assim que o identificador hipotenusa, criado por define, foi associado a um procedimento, devolvido por lambda, que calcula e devolve o valor da hipotenusa de um triângulo dados os comprimentos dos seus catetos.

```
> (define altura 3)
> (define base (- (* 2 altura) 1))
> altura
3
> base
5
> (hipotenusa altura base)
5.830951894845301
> (hipotenusa 3 5)
5.830951894845301
>
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Experimente hipotenusa com vários valores para os catetos.

Experimente também alterar o nome dos parâmetros, por exemplo, para `cateto1` e `cateto2`, e altere o corpo do procedimento para que continue a funcionar correctamente.



**Em vez de `c1` e `c2`, imagine que escolhia `x` e `y` para nomes dos parâmetros.**

**Acha que seria uma boa opção? Justifique.**

**E se em vez de `c1` e `c2` escolhesse `cateto1` e `cateto2`? Justifique.**

## Chamada de um procedimento composto

Depois de um procedimento estar completamente definido, o cálculo que lhe está associado será executado quando o procedimento for chamado. A chamada do procedimento `hipotenusa` faz-se colocando entre parêntesis o nome do procedimento, seguido dos valores correspondentes aos comprimentos dos quatro catetos. Esses valores, associados aos parâmetros, são geralmente designados por argumentos.

```
> (hipotenusa 3 5)  
5.830951894845301  
>
```

Nesta chamada, associa-se 3 a `c1` e 5 a `c2` e o cálculo da hipotenusa é realizado com estes valores.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Note que na definição do procedimento hipotenusa, designámos c1 e c2 por parâmetros, enquanto na chamada do mesmo procedimento designámos os valores associados a c1 e c2 por argumentos.

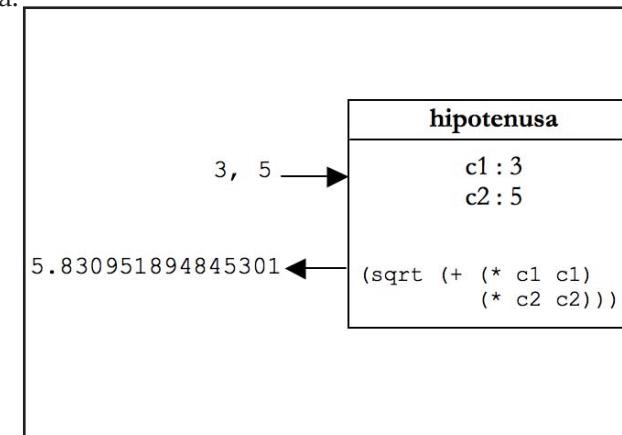


**Na sua opinião, qual é a diferença fundamental entre parâmetro e argumento?**

### Um procedimento não faz nada!

Um procedimento, por si só, não faz nada! Trata-se apenas de uma porção de texto, mais ou menos legível. Só quando o procedimento é chamado, é que acontece qualquer coisa.

Por cada chamada é gerado um processo no interior do computador, o qual vai requerer os recursos computacionais necessários para a execução do cálculo que lhe está associado. O processo toma, como recursos, espaço de memória para os dados (no último exemplo, espaço de memória fundamentalmente para c1 e c2) e também tempo do processador para os cálculos necessários. Podemos imaginar que a figura representa o processo gerado pela chamada (hipotenusa 3 5) e os respectivos recursos computacionais necessários.



No final do processo, este devolve o resultado calculado, 5.830951894845301, e "morre", libertando os recursos computacionais de espaço que tinha tomado. A este assunto voltaremos, mais à frente, onde se procurará avaliar os recursos que cada processo requer de uma forma um pouco mais rigorosa.



**No final, o processo "morre" e liberta apenas os recursos computacionais de espaço.  
Que outros recursos usou o processo que não poderá no final libertar?**

# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Sintaxe do Scheme relativa à definição de um procedimento

Chegado a este ponto, é desafiado a verificar a adequação da forma genérica que se apresenta para definir um procedimento:

```
(define nome-do-procedimento
  (lambda (parâmetro-1 parâmetro-2 ...)
  [-----]
  [ corpo-do-procedimento ] )
  [-----]
```

O procedimento primitivo `lambda` é uma forma especial do Scheme e a sua regra de cálculo é:

1. os identificadores que se encontram a seguir a `lambda`, entre parêntesis, representam valores genéricos e são designados por parâmetros do procedimento;
2. `lambda` devolve o procedimento com os seus parâmetros, associando-lhe a expressão especificada em `corpo-do-procedimento`.



No caso do procedimento `hipotenusa`, indique:

- 1- Quais os seus parâmetros
- 2- Qual a expressão especificada no `corpo-do-procedimento`

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Sintaxe do Scheme relativa à chamada de um procedimento

A chamada de um procedimento apresenta-se segundo a notação pré-fixa:

(nome-do-procedimento argumento-1 argumento-2 ...)

e a regra a aplicar é:

1. são calculados todos os argumentos;
2. no corpo do procedimento, os valores dos argumentos substituem os parâmetros respectivos (argumento-1 substitui parâmetro-1, argumento-2 substitui parâmetro-2, ...);
3. são calculadas as expressões que integram o corpo-do-procedimento e devolvido o valor da última expressão calculada.



Quantos argumentos deve ter uma chamada de um procedimento? Justifique.

### Exercício 1

Escreva em Scheme o procedimento soma-os-dois-e-tira-5 com dois parâmetros. De acordo com o próprio nome, este procedimento recebe dois valores como argumentos, calcula a soma dos dois e subtrai-lhe 5.

```
> (soma-os-dois-e-tira-5 10 3)  
8  
> (soma-os-dois-e-tira-5 10 -3)  
2
```



Desenvolva o procedimento soma-os-dois-e-tira-5 e experimente-o em várias chamadas.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Diagrama de fluxo de dados para evidenciar a relação entre procedimentos

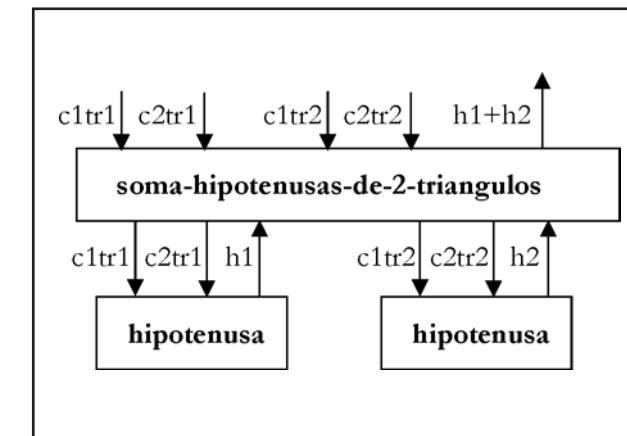
Escreva em Scheme o procedimento soma-hipotenusa-de-2-triangulos que tem quatro parâmetros, c1tr1 e c2tr1, os catetos de um primeiro triângulo, e c1tr2 e c2tr2, os catetos de um segundo triângulo, e que devolve a soma das hipotenusas desses dois triângulos.

O procedimento pretendido é muito simples e o enunciado é suficientemente esclarecedor. É recomendável, no entanto, desenvolver manualmente algumas situações do problema, que servirão mais tarde para testar o funcionamento da solução encontrada. Assim, se os 4 argumentos forem 3, 4, 6 e 8, é de esperar que o resultado seja  $(\text{hipotenusa } 3 \ 4) + (\text{hipotenusa } 6 \ 8) = 5 + 10 = 15$ .

O procedimento soma-hipotenusa-de-2-triangulos, com quatro parâmetros, pode ser visto como recorrendo duas vezes ao procedimento hipotenusa, previamente definido, e é isto o que se pretende esboçar na figura.

Esta figura representa o diagrama de fluxo de dados do procedimento soma-hipotenusa-de-2-triangulos. O diagrama mostra a relação estabelecida entre os procedimentos utilizados e põe em evidência os dados trocados entre eles.

```
> (hipotenusa 3 4)
5
> (hipotenusa 6 8)
10
> (soma-hipotenusa-de-2-triangulos 3 4 6 8)
15
> (soma-hipotenusa-de-2-triangulos 4 3 8 6)
15
>
```



Desenvolva o procedimento soma-hipotenusa-de-2-triangulos e experimente-o em várias situações.

### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

R E 1 2 3 4 5 6 7

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

## **Os Procedimentos como caixas-pretas**

Na definição de `soma-hipotenusa-de-2-triangulos`, foi utilizado o procedimento `hipotenusa`, mas para isso não seria necessário conhecer `hipotenusa` por dentro. Este procedimento pode ser visto como um bloco ou uma caixa-preta em que entram os comprimentos de dois catetos de um triângulo rectângulo e sai o comprimento da hipotenusa. Quando utilizamos `hipotenusa`, apenas precisamos de saber o que faz e não como faz. Ou seja, bastar-nos-ia conhecer a sua interface (nome, parâmetros e valor que devolve), não nos importando com a sua implementação. Isto é um exemplo, embora muito simples, de uma abstracção procedimental, pois estamos a trabalhar com um procedimento que calcula a hipotenusa, sem a preocupação de saber como esse cálculo é realizado.

Outro aspecto interessante da caixa-preta tem a ver com o facto de o procedimento `hipotenusa` utilizar `c1` e `c2` como nomes dos seus parâmetros, sem que isso impeça que qualquer outro procedimento utilize parâmetros com estes mesmos nomes. Isto não causará qualquer confusão ao Scheme, pois os nomes dos parâmetros têm um significado muito bem localizado e apenas reconhecido no corpo do respectivo procedimento. Aliás, como vimos, sempre que surge um processo criado pela chamada de um procedimento, é reservado espaço próprio em memória para as entidades que irá processar, não se confundindo portanto, com o espaço reservado por qualquer outro processo.

### **Exercício 2**

O objectivo deste exercício é provar que os procedimentos podem ser utilizados como se fossem caixas pretas. Assim, propõe-se que tome como ponto de partida a solução apresentada anteriormente para o procedimento `soma-hipotenusa-de-2-triangulos` que utiliza o procedimento `hipotenusa`. Desenvolva uma nova solução para este procedimento `hipotenusa` e teste-a para verificar que funciona correctamente. Depois disto, teste o procedimento `soma-hipotenusa-de-2-triangulos`, sem lhe introduzir qualquer alteração, e verifique que continua a funcionar bem.



## INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



A simplicidade do procedimento `hipotenusa` não dará certamente muitas hipóteses para encontrar uma nova solução, diferente da já conhecida.

E se tentar uma solução que utilize um procedimento, designado por `quadrado`, que devolve o quadrado do valor que recebe como argumento?

> (`quadrado 5`)

25

> (`quadrado 2`)

4



Desenvolva e teste o exercício proposto.



Apresente uma explicação para o facto de `soma-hipotenusas-de-2-triangulos` continuar a funcionar correctamente, mesmo tendo alterado o procedimento `hipotenusa`.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Módulo 1.3 - Projecto - Vamos acertar no totoloto!

O problema que se propõe, sob a forma de um projecto, está relacionado com o desenvolvimento de um programa em Scheme para auxiliar os apostadores no preenchimento de boletins de totoloto.

A tarefa que o programa deve realizar, definida de uma forma genérica, resume-se a gerar aleatoriamente apostas de totoloto. Mas é fundamental entender o problema em pormenor e depois encontrar uma ideia para o resolver. Em seguida, essa ideia é materializada numa sequência de passos, o algoritmo, utilizando para tal uma linguagem informal. Falta converter o algoritmo em código Scheme e testá-lo...

No decorrer do projecto, surgirá a oportunidade para trabalhar com números aleatórios e para programar a visualização no ecrã.

A utilização do Scheme é obrigatória no contexto deste módulo, uma vez que, no final, deve existir um programa para auxiliar os apostadores de totoloto.

### Palavras-Chave

Especificação, algoritmo, codificação, teste.



# **S C H E M E**

## **na descoberta da programação**

### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

R E 1 2 3 4 5 6 7

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

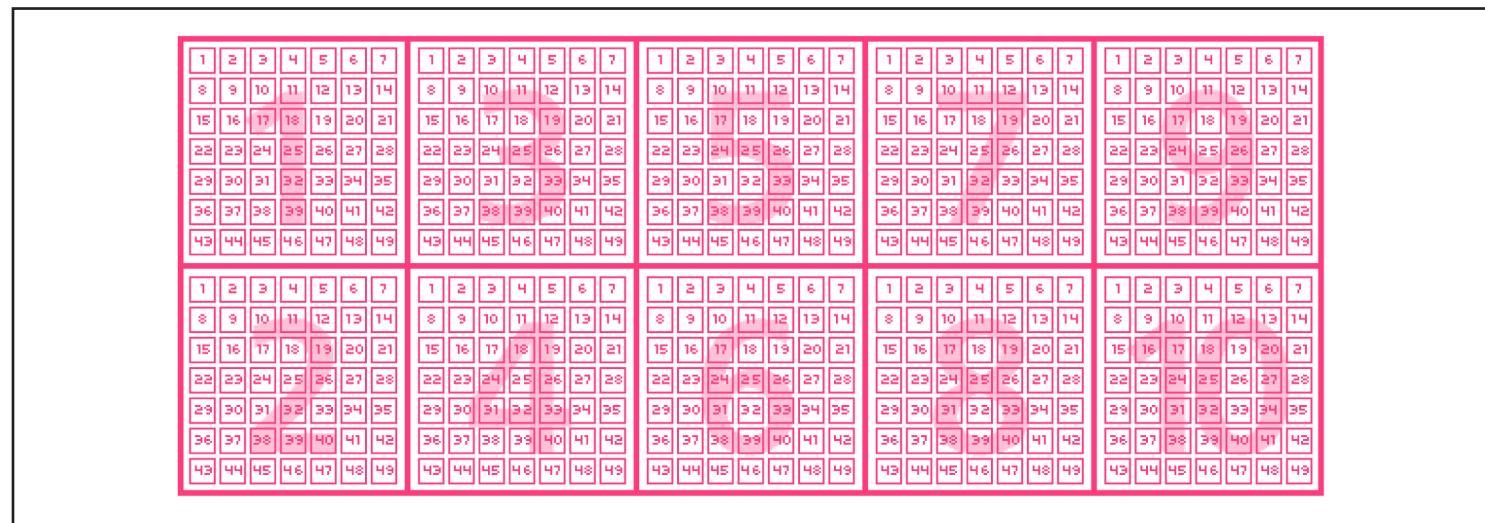
### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

## **O problema a resolver**

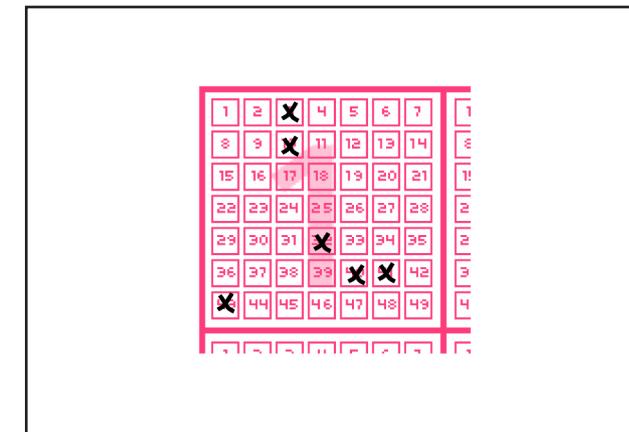
É importante que comece por entender muito bem o problema que lhe colocam, para não correr o risco de resolver, mesmo que bem, outra coisa que não a que interessa. Ou seja, a especificação do problema não deve suscitar dúvidas e, se alguma existir, deverá ser ultrapassada com os esclarecimentos necessários. A especificação de um certo problema vai agora começar.

Um boletim de totoloto contém espaço para 10 apostas.



Cada aposta do boletim resume-se a uma sequência de 6 inteiros não repetidos, situados entre 1 e 49. Na figura vê-se uma aposta preenchida com os números 3, 10, 32, 40, 41 e 43.

O programa a desenvolver deve auxiliar os apostadores a preencher uma aposta de cada vez, ou seja, deve gerar uma sequência de 6 inteiros, não repetidos, situados entre 1 e 49.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Consegue explicar por que razão esta opção vem simplificar a realização do projecto?

Podemos imaginar que, a cada chamada do programa, será visualizado no ecrã algo equivalente a:

Numero 1: 49

Numero 2: 32

Numero 3: 6

Numero 4: 26

Numero 5: 19

Numero 6: 22

1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	X	7	8	9	10	11	12	13	14	15	16	17	18	X	20	21	22	23	24	25	X	27	28	29	30	31	X	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	X	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	7																						

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

- Passo 1 - visualiza linha correspondente ao número 1
- Passo 2 - visualiza linha correspondente ao número 2
- Passo 3 - visualiza linha correspondente ao número 3
- Passo 4 - visualiza linha correspondente ao número 4
- Passo 5 - visualiza linha correspondente ao número 5
- Passo 6 - visualiza linha correspondente ao número 6

### Escrever o programa em Scheme

Atendendo ao que já ficou estabelecido, parece perfeitamente razoável aceitar que o programa pode ser concretizado num procedimento com o nome `aposta-totoloto` e que não precisa de qualquer parâmetro.

```
> (aposta-totoloto)
Numero 1: 30
Numero 2: 19
Numero 3: 9
Numero 4: 39
Numero 5: 9
Numero 6: 46
> (aposta-totoloto)
Numero 1: 48
Numero 2: 47
Numero 3: 33
Numero 4: 18
Numero 5: 9
Numero 6: 37
>
```

Aprofundando um pouco mais o procedimento `aposta-totoloto`, poder-se-ia dizer que o mesmo usa 6 vezes um procedimento auxiliar, designado por `linha-aposta`, relacionado com a visualização no ecrã de uma linha da aposta. Ao analisar o conteúdo de uma linha visualizada, conclui-se que a sequência de passos do procedimento `linha-aposta` deverá ser:

- Passo 1 - visualiza a cadeia de caracteres "Numero "
- Passo 2 - visualiza um inteiro entre 1 e 6, que lhe é fornecido
- Passo 3 - visualiza a cadeia de caracteres ":"
- Passo 4 - visualiza um inteiro entre 1 e 49, gerado aleatoriamente



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

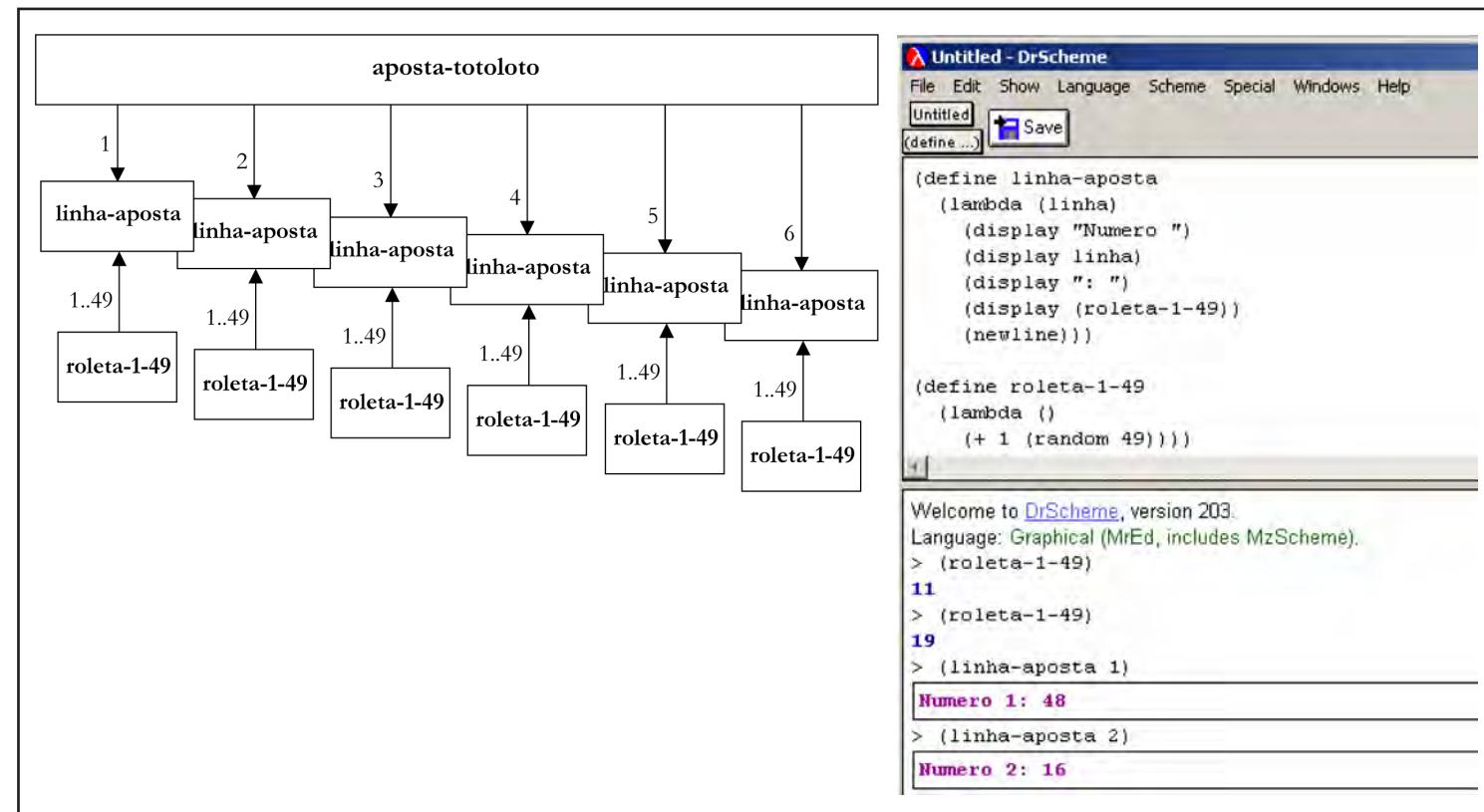
## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

O inteiro referido no passo 2 pode ser fornecido através de um parâmetro de linha-aposta, enquanto o inteiro referido no passo 4 pode ser gerado recorrendo a um procedimento auxiliar, designado por roleta-1-49.



Verifique estas decisões de projecto no diagrama de fluxo de dados de apostatotoloto.





Experimente os procedimentos `linha-aposta` e `roleta-1-49`.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



INTRODUÇÃO
1 - O ESSENCIAL DO SCHEME
R E 1 2 3 4 5 6 7
2 - RECURSIVIDADE
3 - ABSTRACÇÃO DE DADOS
4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE
5 - ABSTRACÇÕES COM DADOS MUTÁVEIS
6 - SCHEME E OUTRAS TECNOLOGIAS
7-EXERCÍCIOS E PROJECTOS
ANEXOS

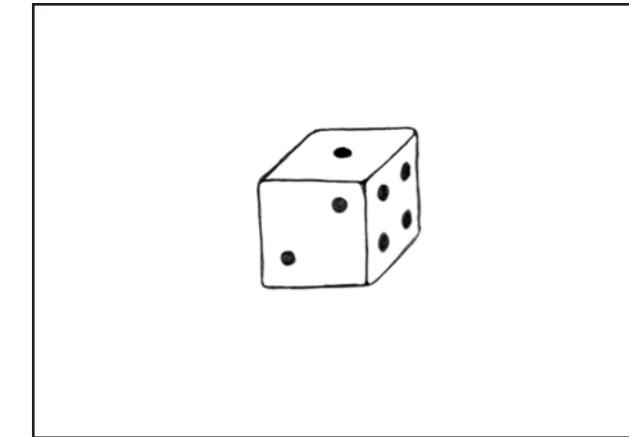


O que acha da solução para o procedimento `aposta-totoloto` que se segue?  
Será igual à solução a que chegou?

```
(define apostatotoloto
  (lambda ()
    (linha-aposta 1)
    (linha-aposta 2)
    (linha-aposta 3)
    (linha-aposta 4)
    (linha-aposta 5)
    (linha-aposta 6))))
```

### Exercício 2

O procedimento `roleta-1-49` não tem parâmetros e por isso só pode ser utilizado para gerar números inteiros entre 1 e 49. Se pretendesse, por exemplo, gerar números inteiros aleatórios entre 1 e 6, para simular o lançamento de um dado, teria que escrever um novo procedimento, provavelmente com o nome `roleta-1-6`. Contudo, uma solução mais flexível será o procedimento `roleta-1-n`, com um parâmetro `n`, para gerar aleatoriamente um inteiro entre 1 e `n`.



```
> (roleta-1-n 6)
5
> (roleta-1-n 6)
3
> (roleta-1-n 49)
13
```

Escreva em Scheme o procedimento `roleta-1-n` e teste-o.

No projecto do preenchimento de apostas de totoloto substitua o procedimento `roleta-1-49` por `roleta-1-n` e introduza as alterações necessárias.



Desenvolva e teste o procedimento roleta-1-n e modifique o programa para preenchimento de apostas de totoloto.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

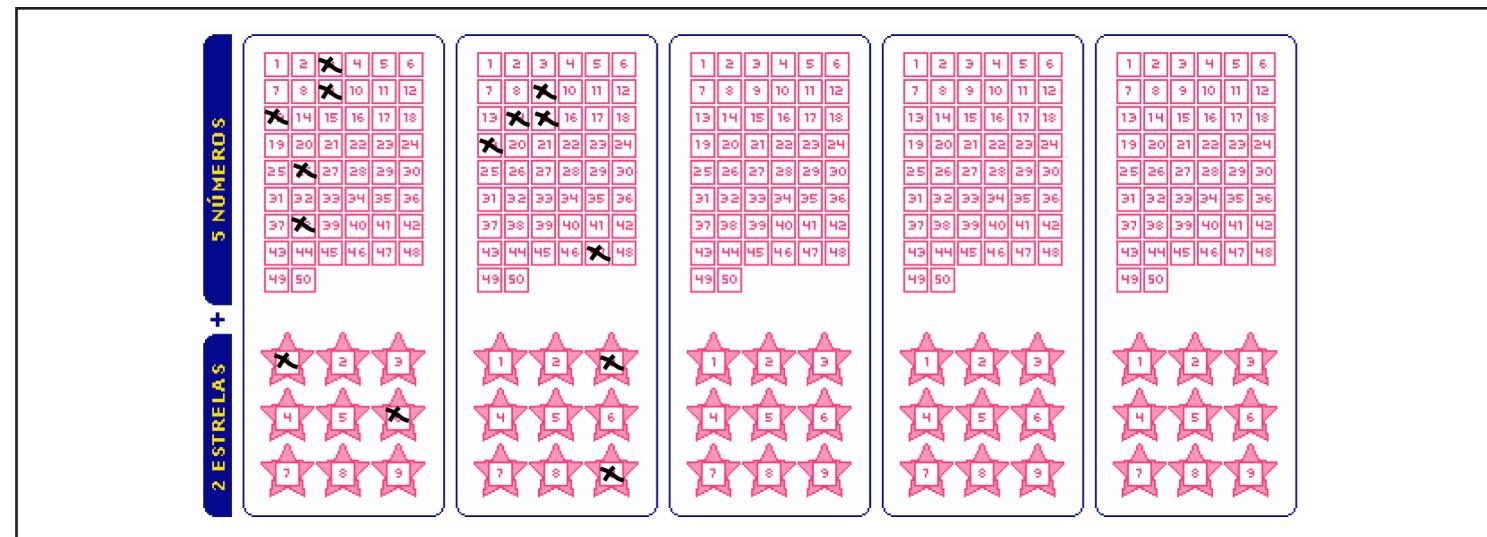
### Exercício 3

Um boletim do euromilhões contém espaço para 5 apostas. Cada aposta do boletim resume-se a uma sequência de 5 inteiros não repetidos, situados entre 1 e 50, e outra sequência de 2 inteiros não repetidos, as estrelas, situados entre 1 e 9.

Na figura pode ver um boletim do euromilhões preenchido com 2 apostas, correspondentes a:

números: 3, 9, 13, 26, 38    estrelas: 1, 6

números: 9, 14, 15, 19, 47    estrelas: 3, 9



O programa a desenvolver deve auxiliar os apostadores a preencher uma aposta de cada vez, gerando aleatoriamente 5 números, seguidos de 2 estrelas. Para simplificar o problema, não é feita a verificação de números ou estrelas repetidos.

Podemos imaginar que, a cada chamada do programa, será visualizado o que se segue:



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (aposta-euromilhoes)
Número 1: 38
Número 2: 26
Número 3: 9
Número 4: 13
Número 5: 3
Estrela 1: 6
Estrela 2: 1
> (aposta-euromilhoes)
Número 1: 15
Número 2: 9
Número 3: 14
Número 4: 19
Número 5: 47
Estrela 1: 9
Estrela 2: 3
```

Escreva em Scheme o procedimento `aposta-euromilhoes`, introduzindo as alterações necessárias na solução do Exercício 1, relativo ao programa para preenchimento de apostas de totoloto.



Desenvolva e teste o procedimento `aposta-euromilhoes`.



Revendo o trajecto que o levou ao procedimento `aposta-euromilhoes`, vai conseguir identificar os vários passos de desenvolvimento daquele programa:

- identificação e especificação do problema
- procura de uma ideia para resolver o problema
- desenvolvimento de um algoritmo, com a sequência de passos adequada
- codificação do procedimento na linguagem Scheme
- teste

Destes 5 passos, qual deles

- exigirá mais criatividade?
- será mais rotineiro?
- exigirá mais interacção com o programador?



## INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 **4** 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Módulo 1.4 - O Scheme sabe trabalhar com ângulos

O Scheme disponibiliza um conjunto de procedimentos primitivos para o processamento trigonométrico, processamento que envolve a manipulação de ângulos.

Neste módulo, pretende-se essencialmente mostrar quais são e como se utilizam os procedimentos primitivos que o Scheme coloca à nossa disposição para o processamento trigonométrico.

No decorrer do módulo, surgirá ainda a oportunidade para mostrar como criar variáveis locais e qual o respectivo interesse, bem como ler dados directamente do teclado.

### Palavras-Chave

Círculo trigonométrico, processamento trigonométrico com procedimentos primitivos, grau, radiano, variável local.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

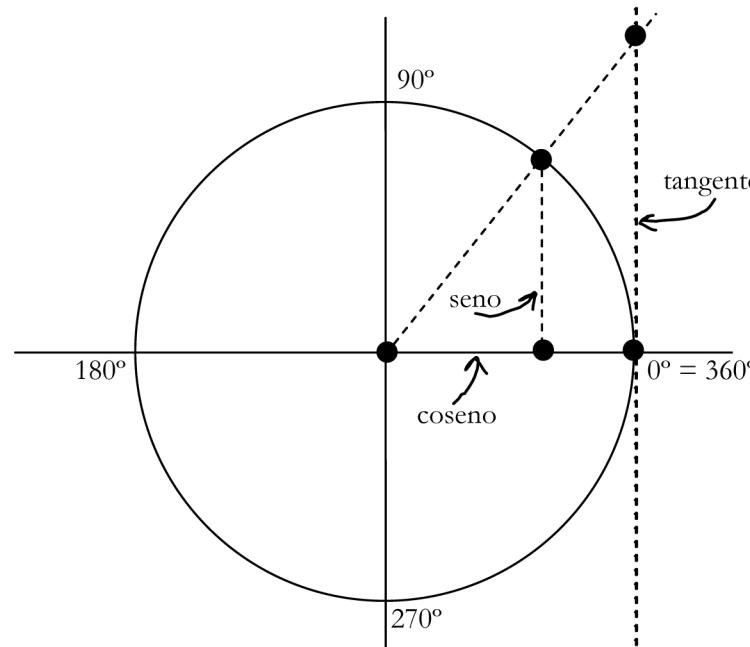
### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## É importante relembrar o círculo trigonométrico

A intenção não é substituir os manuais que tratam do processamento trigonométrico de uma forma aprofundada, pois aqui apenas se adiantam algumas notas julgadas necessárias a uma boa compreensão do tema deste módulo. O principal elemento da figura é o círculo trigonométrico, por convenção, de raio unitário. Neste círculo são representados graficamente o seno, o coseno e a tangente de um ângulo. Por simples observação, é possível concluir que se este ângulo fosse de  $0^\circ$ , o seno seria 0, o coseno 1 e a tangente 0. No entanto, se o ângulo fosse de  $90^\circ$ , o seno passaria a ser 1, o coseno 0 e a tangente infinita.



Continue a pesquisar o círculo trigonométrico de uma forma gráfica, sem fazer cálculos:

- indique o valor da tangente para  $45^\circ$ ,  $135^\circ$ ,  $225^\circ$  e  $315^\circ$
- indique o seno de  $180^\circ$  e  $270^\circ$ .



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



E agora ao contrário, ou seja, dado, por exemplo, o valor de um seno ou de um coseno ou de uma tangente, determinar o ângulo correspondente:

- indique um ângulo cujo coseno seja 0
- indique um ângulo cujo coseno seja -1
- indique um ângulo cujo seno seja -1.

## O processamento trigonométrico em Scheme

A funcionalidade do Scheme relacionada com o processamento trigonométrico resume-se praticamente ao cálculo do seno, coseno e tangente de um ângulo dado e ao cálculo de um ângulo dado o seno, coseno ou tangente, como pode ser visto no [Anexo A](#).

```
> (sin 0)
0
> (asin 0)
0
```

Observe agora com atenção como responde o Scheme quando se pretende saber o seno com o argumento 90 ou o arco cujo seno é 1.

```
> (sin 90)
0.8939966636005579
> (asin 1)
1.5707963267948966
>
```

Observando estas respostas, ficamos com a ideia de que o Scheme não sabe tratar correctamente o processamento trigonométrico. Provavelmente, já percebeu a razão de respostas aparentemente tão surpreendentes. Elas são devidas ao facto do Scheme trabalhar com ângulos especificados em radianos e não em graus. Quando apresenta ao Scheme a expressão `(sin 90)` a resposta devolvida refere ao seno de 90 radianos e não ao seno de 90°. Como estamos muito mais habituados a trabalhar com graus, o melhor é preparar dois procedimentos auxiliares, um que converta graus em radianos e outro que faça a operação inversa.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Para entender a solução que se apresenta, bastará lembrar que a  $180^\circ$  corresponde a  $\pi$  radianos, em que  $\pi = 3.14159...$

```
(define pi 3.141592653589793) ; definição do identificador pi

; toma um ângulo em graus
; devolve o equivalente em radianos
(define degrees->radians
  (lambda (ang-deg)
    (/ (* pi ang-deg)
       180.0)))

; toma um ângulo em radianos
; devolve o equivalente em graus
(define radians->degrees
  (lambda (ang-rad)
    (/ (* ang-rad 180.0)
       pi)))
```

Algumas implementações do Scheme pouparam-nos este trabalho, pois, à partida, já disponibilizam estes dois procedimentos auxiliares.

Analise os dois exemplos que se seguem e verifique que, afinal, o Scheme processa bem as funções trigonométricas.

```
> (sin (degrees->radians 90))
1.0
> (radians->degrees (asin 1))
90.0
```



Experimente os procedimentos primitivos do Scheme destinados ao processamento trigonométrico e os procedimentos auxiliares `degrees->radians` e `radians->degrees`.

**Pista:** Construa, manualmente, mas com a ajuda do Scheme, uma tabela de senos e cossenos, de  $0$  a  $120^\circ$ , de  $30$  em  $30^\circ$ .

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Desenvolvimento de um programa para calcular os ângulos de um triângulo

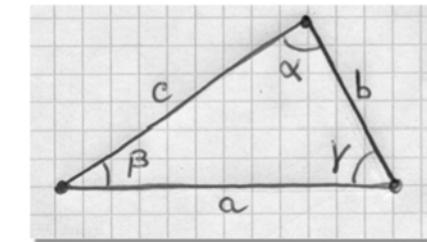
A fórmula de Briggs tem o nome do matemático inglês Henry Briggs (1561-1631), que foi o primeiro professor de geometria no Gresham House, em Londres.

Com esta fórmula é possível calcular os três ângulos internos  $\alpha$ ,  $\beta$  e  $\gamma$  de um triângulo, a partir do comprimento dos seus lados  $a$ ,  $b$  e  $c$ . Na fórmula,  $sp$  é o semi-perímetro do triângulo, ou seja,  $(a+b+c) / 2$ .

$$\sin \frac{\alpha}{2} = \sqrt{\frac{(sp-b)*(sp-c)}{b*c}}$$

$$\sin \frac{\beta}{2} = \sqrt{\frac{(sp-a)*(sp-c)}{a*c}}$$

$$\sin \frac{\gamma}{2} = \sqrt{\frac{(sp-a)*(sp-b)}{a*b}}$$



O programa em Scheme que se pretende receber os três lados de um triângulo e visualiza os seus três ângulos internos.

Da chamada do programa com os dados 10.0, 10.0 e 10.0 deve resultar:

```
O triangulo com os lados de comprimento: 10.0, 10.0, 10.0
tem os seguintes angulos: 60.00000000000001, 60.00000000000001, 60.00000000000001
```



**Tem alguma justificação para a resposta obtida, sabendo que um triângulo de lados iguais tem os três ângulos internos exactamente iguais a 60°?**



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Da chamada do programa com os dados 10.0, 15.0 e 15.0 deve resultar:

O triangulo com os lados de comprimento: 10.0, 15.0, 15.0  
tem os seguintes angulos: 38.94244126898138, 70.52877936550931, 70.52877936550931

### Procurar uma ideia para resolver o problema

Depois do problema estar bem percebido, a meta seguinte é encontrar uma ideia para o resolver - apresentar o triângulo, visualizando o comprimento dos 3 lados dados e, seguidamente, visualizar os seus 3 ângulos, cujo valor será necessário calcular.



**Surgem, de imediato, algumas perguntas:**

- que nome dará ao procedimento em que se materializa o programa desejado?
- interessa ou não conceber um procedimento auxiliar para calcular um ângulo (supondo que o mesmo será chamado 3 vezes para calcular os 3 ângulos)?
- Em caso afirmativo, que nome terá esse procedimento? E quais serão os seus parâmetros?

Em torno da ideia encontrada já é possível descrever a sequência de passos do programa a desenvolver, ou seja, o seu algoritmo.

Passo 1 - visualiza a mensagem "O triangulo com os lados de comprimento: "

Passo 2 - visualiza, separados por vírgula, os comprimentos dos 3 lados

Passo 3 - a visualização passa para o início da linha seguinte

Passo 4 - visualiza a mensagem "tem os seguintes angulos: "

Passo 5 - visualiza, separados por vírgula, os valores dos ângulos em graus

### Escrever o programa em Scheme

Atendendo ao que já ficou estabelecido, parece perfeitamente razoável aceitar que o programa pode concretizar-se num procedimento com o nome `briggs`, com 3 parâmetros que representam os comprimentos dos lados do triângulo.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

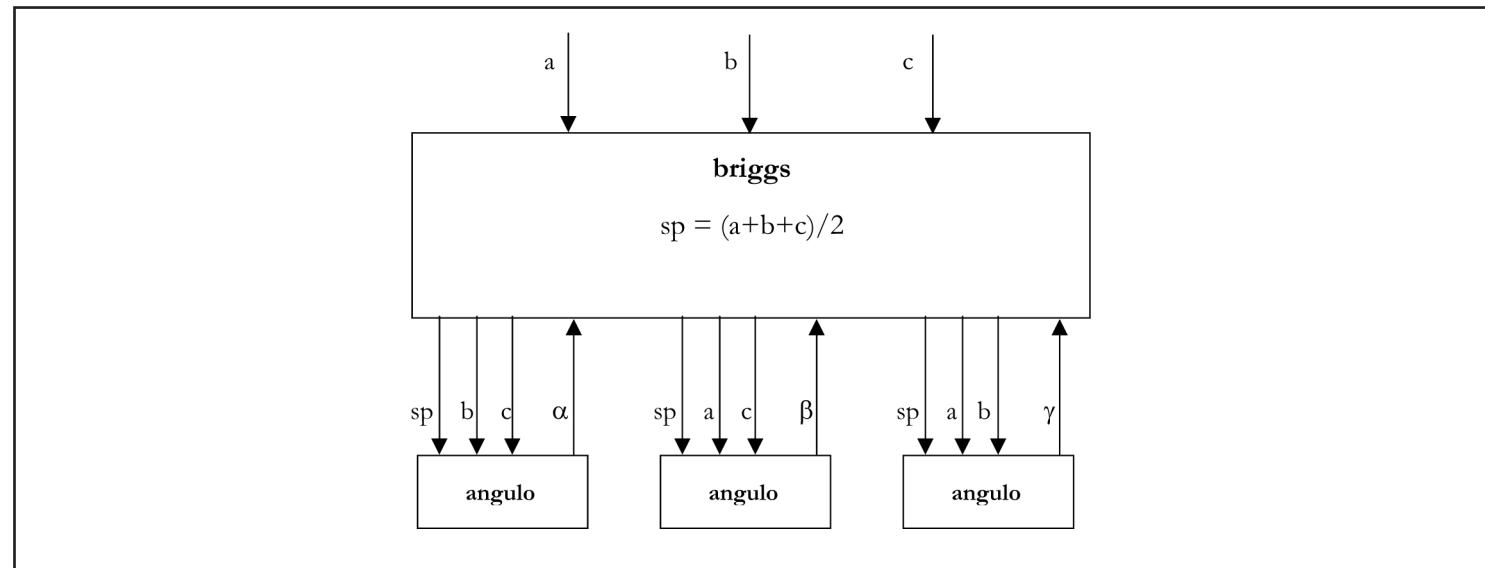
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (briggs 10 15 20)
O triangulo com os lados de comprimento: 10, 15, 20
tem os seguintes angulos: 28.955024371859846, 46.56746344221024, 104.47751218592992
>
```

Consolidando e aprofundando um pouco mais o procedimento `briggs`, poder-se-á dizer que o mesmo:

- tem como parâmetros `a`, `b` e `c`;
- usa 3 vezes um procedimento auxiliar, chamado `angulo`, com 3 parâmetros, `sp` - semiperímetro do triângulo, `lado1` e `lado2`, dois dos lados de um triângulo; o procedimento `angulo` devolve o ângulo definido pelos lados designados por `lado1` e `lado2`.



O procedimento `briggs` traz uma novidade interessante: cria uma variável local, designada por `sp`.

- indique uma razão que justifique a criação de `sp`
- indique o que aconteceria no diagrama de fluxo de dados se `briggs` não criasse `sp` no seu interior.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
; Procedimento com 3 parâmetros que representam os lados de um triângulo.  
; O valor devolvido por este procedimento não tem qualquer interesse.  
; Deste procedimento apenas se espera um conjunto de mensagens no ecrã  
; que descrevem os ângulos internos do triângulo.  
;  
(define briggs  
  (lambda (a b c)  
    (let ((sp (/ (+ a b c)  
                 2)))  
      ; definição de sp (semi-perímetro)  
      (display "O triangulo com os lados de comprimento: ")  
      (display a)  
      (display ", ")  
      (display b)  
      (display ", ")  
      (display c)  
      (newline)  
      (display "tem os seguintes angulos: ")  
      (display (angulo sp b c))  
      (display ", ")  
      (display (angulo sp a c))  
      (display ", ")  
      (display (angulo sp a b))  
      (newline))))  
  
(define angulo  
  (lambda (sp lado1 lado2)  
    (radians->degrees  
     (* 2  
        (asin (sqrt (/ (* (- sp lado1)  
                           (- sp lado2))  
                    (* lado1 lado2)))))))  
  
(define pi 3.141592653589793)  
;  
(define degrees->radians  
  (lambda (ang-deg)  
    (/ (* pi ang-deg)  
        180.0)))  
;  
(define radians->degrees  
  (lambda (ang-rad)  
    (/ (* ang-rad 180.0)  
        pi)))
```



# SCHEME

## na descoberta da programação

INTRODUÇÃO
1 - O ESSENCIAL DO SCHEME
R E 1 2 3 4 5 6 7
2 - RECURSIVIDADE
3 - ABSTRACÇÃO DE DADOS
4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE
5 - ABSTRACÇÕES COM DADOS MUTÁVEIS
6 - SCHEME E OUTRAS TECNOLOGIAS
7-EXERCÍCIOS E PROJECTOS
ANEXOS



Experimente o procedimento `briggs`.



O procedimento `briggs` começa com algumas linhas de comentários que ajudam a entender o que ele faz. Não lhe parece que o procedimento `angulo` também precisaria de uns comentários semelhantes? E os outros dois procedimentos auxiliares?

A criação da variável local `sp` foi possível graças a `let`, mais uma forma especial do Scheme.

Analise com cuidado esta forma geral:

```
(let ( [ nome-1 expressão-1 ]
      [ nome-2 expressão-2 ]
      [ ... ] )
    [ corpo-de-let ] )
```

Deve identificar duas áreas distintas na forma `let`. Numa das áreas, definida entre parêntesis, são associados aos identificadores `nome-1`, `nome-2`, ..., os valores das expressões respectivas, `expressão-1`, `expressão-2`, ... Na outra área, designada por `corpo-de-let`, e só aí, aqueles identificadores serão reconhecidos e poderão ser utilizados.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Verifique a correcta aplicação da forma `let` no procedimento `briggs`. Os parâmetros `a`, `b` e `c` são utilizáveis em todo o corpo do procedimento, no entanto, a variável local `sp` é utilizável apenas no corpo de `let`.

```
; Deste procedimento apenas se espera um conjunto de mensagens no ecrã
; que descrevem os ângulos internos do triângulo.
;
(define briggs
  (lambda (a b c)
    .....  

    (let ((sp (/ (+ a b c)
                  2)))
      .....  

      (display "O triangulo com os lados de comprimento: ")
      (display a)
      (display ", ")
      (display b)
      (display ", ")
      (display c)
      (newline)
      (display "tem os seguintes angulos: ")
      (display (angulo sp b c))
      (display ", ")
      (display (angulo sp a c))
      (display ", ")
      (display (angulo sp a b))
      (newline)
    )))
```

### Exercício 1

Analise o procedimento `experiencia-com-let-1`:

```
(define experiencia-com-let-1
  (lambda (x y)
    (let ((local1 (+ x y))
          (local2 (* x y)))
      .....  

      (+ local1 local2))))
```

- indique os nomes das variáveis definidas localmente;
- calcule manualmente as respostas às situações que se apresentam e depois teste com o Scheme.

```
> (experiencia-com-let-1 3 7)  
??
```





Experimente o procedimento `experiencia-com-let-1`.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



## Exercício 2

Analise o procedimento `experiencia-com-let-2`, que apresenta um segundo `let` dentro do corpo do primeiro `let`.

```
(define experiencia-com-let-2
  (lambda (x y)
    (let ((local1 (+ x y))
          (local2 (* x y)))
      .....
      (let ((maislocal (- local1 local2)))
        .....
        (* (+ local1 local2)
           maislocal)
        ))))
```

- identifique graficamente o corpo do segundo `let`, onde passa a ser reconhecida a variável local `maislocal`, para além das 2 variáveis criadas no primeiro `let`;
- calcule manualmente as respostas às situações que se apresentam e depois teste com o Scheme.

```
> (experiencia-com-let-2 3 7)
??
> (experiencia-com-let-2 -3 7)
??
```



Experimente o procedimento `experiencia-com-let-2`.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 3 - Um desafio relativamente pequeno... com variáveis locais

A conversão de uma temperatura em graus Fahrenheit ( $f$ ) numa em graus Centígrados ( $c$ ) pode ser realizada através da fórmula  $c = (f - 32) * 5/9$ . Por outro lado, para converter uma temperatura em graus Centígrados numa em graus Kelvin bastará somar 273.16.

Escrever em Scheme o procedimento `converte-Fahrenheit` com o parâmetro `t f` que representa uma temperatura em graus Fahrenheit e responde como se indica:

```
> (converte-Fahrenheit 14)
```

Fahrenheit: 14

Centigrados: -10

Kelvin: 263.16

O procedimento pedido cria uma variável local com a temperatura em graus Centígrados para ser aproveitada no cálculo da temperatura em graus Kelvin.



Desenvolva e teste o procedimento `converte-Fahrenheit`.

**Pista:** Elabore o algoritmo deste problema que, certamente, começará com as conversões de temperatura e terminará com as respectivas visualizações.

### Leitura de dados com origem no teclado

Até ao momento, todos os procedimentos desenvolvidos recebiam os dados através dos seus parâmetros, mas, por vezes, achará mais apropriado que os dados sejam fornecidos através do teclado.



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Imagine então uma nova situação em que um procedimento, designado por `circulo`, sem parâmetros, principia por pedir o raio de um círculo e termina visualizando no ecrã a área do círculo e o respectivo perímetro.

A screenshot of the DrScheme graphical interface. The code area shows the beginning of a procedure definition:

```
; Procedimento auxiliar. Recebe o raio
; de um circulo e devolve a sua área.
;
```

The message area displays the welcome message and language information:

Welcome to DrScheme, version 203.  
Language: Graphical (MrEd, includes MzScheme).

The interaction area shows the prompt `> (circulo)` followed by the message `raio:` in purple, indicating where the user should input a value.

A mensagem `raio:` indica que o procedimento espera a sua resposta (neste caso, o valor do raio do círculo), que deverá ser terminada com a actuação da tecla `return`. Seja 10 o valor fornecido... e a resposta é dada de imediato.

A screenshot of the DrScheme graphical interface. The code area shows the completed procedure definition:

```
; Procedimento auxiliar. Recebe o raio
; de um circulo e devolve a sua área.
;
```

The message area displays the welcome message and language information:

Welcome to DrScheme, version 203.  
Language: Graphical (MrEd, includes MzScheme).

The interaction area shows the prompt `> (circulo)` followed by the message `raio: 10` in purple. Below it, the results of the calculation are shown in green:  
`Area do circulo e': 314.1592653589793`  
`Perimetro do circulo e': 62.83185307179586`

No diagrama de fluxo de dados do procedimento `circulo` estão em evidência os três procedimentos que este utiliza. Dois deles são procedimentos auxiliares do procedimento `circulo`, destinados ao cálculo da área e ao cálculo do perímetro. O outro, `read`, é um procedimento fornecido pelo Scheme. Este procedimento primitivo, sem parâmetros, espera um valor que deverá ser fornecido através do teclado.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

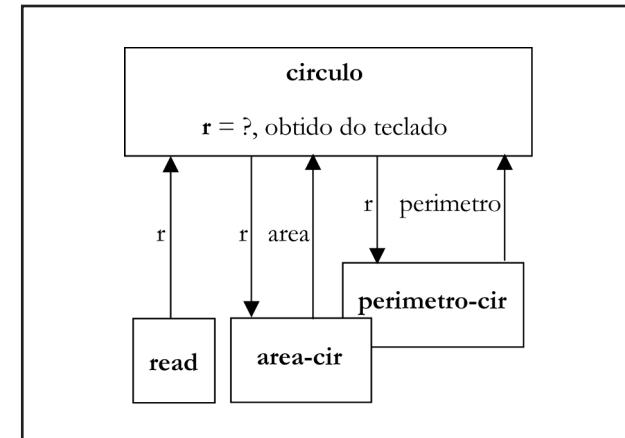
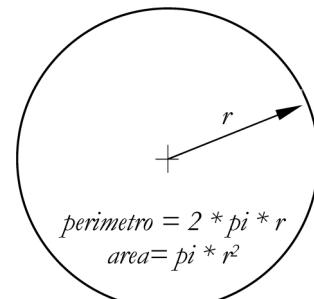
### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



A leitura de dados através do teclado é antecedida pela visualização de uma mensagem apropriada, como pode ver no procedimento `circulo`. Este tipo de situação é vulgar e recomenda-se.  
Apresente uma justificação para a existência desta mensagem.

```
; Procedimento sem parâmetros.  
; Pede o valor do raio de um círculo, a fornecer através do teclado.  
; O valor devolvido por este procedimento não tem qualquer interesse.  
; Deste procedimento apenas se espera um conjunto de mensagens no ecrã  
; que definem a área e o perímetro do círculo.  
;  
(define circulo  
  (lambda ()  
    (display "raio: ")  
    (let ((raio (read)))  
      (display "Area do circulo e': ")  
      (display (area-cir raio))  
      (newline)  
      (display "Perimetro do circulo e': ")  
      (display (perimetro-cir raio))))
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
; Procedimento auxiliar. Recebe o raio de um círculo
; e devolve a sua área.
;
(define area-cir
  (lambda (r)           ; area do círculo = pi * r * r
    (* pi r r)))

; Procedimento auxiliar. Recebe o raio de um círculo
; e devolve o seu perímetro.
;
(define perimetro-cir
  (lambda (r)           ; perimetro do círculo = 2 * pi * r
    (* 2 pi r)))
;
(define pi 3.141592653589793)
```

A definição de uma variável local com o valor devolvido por `read` é uma prática usual. Se assim não fosse, este valor ou era imediatamente utilizado - por exemplo, como na expressão: `(area-cir (read))` - ou perder-se-ia, não podendo ser mais utilizado.



Teste o procedimento `circulo`.



Há quem defina o valor de `pi` de uma maneira diferente.

`(define pi (acos -1))`

Justifique esta opção.



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



**Exercício 4** - Projecto com processamento trigonométrico e entrada de dados pelo teclado

A área de um triângulo pode determinar-se a partir do comprimento dos seus lados a, b, e c, com a fórmula que se apresenta, em que  $sp$  é o semi-perímetro do triângulo.

$$area = \sqrt{sp * (sp - a) * (sp - b) * (sp - c)}$$

Escreva em Scheme o procedimento `area-triangulo` que devolve a área do triângulo, em que a leitura dos comprimentos dos três lados é feita através do teclado.

```
> (area-triangulo)
lado1: 10
lado2: 20
lado3: 25
area do triangulo: 94.99177595981665
```

Sugere-se a definição de uma variável local `sp` para evitar a repetição do cálculo do semi-perímetro.



Desenvolva e teste o procedimento `area-triangulo`.

**Pista:** Não se esqueça de definir um algoritmo...



**Explique a resposta que se segue, dada pelo procedimento `area-triangulo`.**

```
> (area-triangulo)
lado1: 10
lado2: 20
lado3: 30
area do triangulo: 0
```

## INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Módulo 1.5 - Tomar decisões com o Scheme

Perante várias hipóteses, torna-se necessário, muitas vezes, optar por uma delas. Isto traduz uma situação que ocorre frequentemente em programação.

O objectivo deste módulo é mostrar como é possível tomar decisões em Scheme, através do uso das formas especiais `if` e `cond`, e identificar o tipo de situações em que uma forma é mais adequada do que outra.

Uma decisão é tomada quando uma certa condição se verifica. Uma condição materializa-se numa expressão designada por predicado, expressão que toma apenas o valor verdadeiro (`#t`) ou o valor falso (`#f`). Os predicados envolvem operadores de relação e operadores booleanos que serão também considerados.

No decorrer do módulo ainda aparece mais uma forma especial do Scheme, `begin`, que garante que, num bloco de expressões, estas sejam calculadas em sequência, desde a primeira até à última.

Trata-se de um tema que requer um grande domínio e, por este facto, é incentivado o uso do Scheme para a exploração de situações propostas ou outras criadas pelo próprio leitor.

### Palavras-Chave

Predicado, operadores de relação, operadores booleanos.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

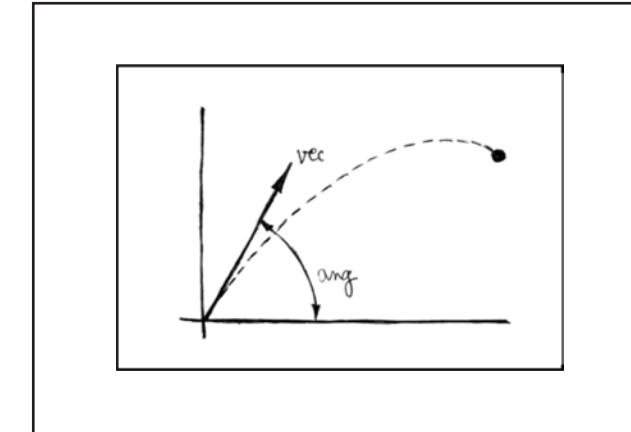
## É preciso tomar decisões

São frequentes as situações que exigem a tomada de decisões. Perante várias hipóteses, torna-se necessário, muitas vezes, optar por uma delas. A frase "Se certo facto se verificar, então tomo a decisão x, se não, tomo a decisão y" traduz bem um certo tipo de raciocínio que encontramos com enorme frequência tanto na vida real como em programação.

### Decisões com a forma especial **if**

A figura descreve a simulação do lançamento de um projétil no espaço, representado por uma pequena esfera preta e com a trajectória a tracejado.

Se o vector *vec* representar a velocidade inicial do projétil lançado da origem dos eixos e *ang* o ângulo de lançamento, pretende-se que o projétil não seja lançado para "trás". Então, o ângulo *ang* deverá ser menor que 90°.



```
(if (< ang 90)
    .....
    (display "faz o lançamento")
    .....
    (display "projectil mal orientado") )

> (define ang 30)
> (if (< ang 90)
    (display "faz o lançamento")
    (display "projectil mal orientado"))
  faz o lançamento

> (define ang 90)
> (if (< ang 90)
    (display "faz o lançamento")
    (display "projectil mal orientado"))
  projectil mal orientado
>
```



Nesta forma `if`, surge uma expressão de relação, (`< ang 90`), cujo resultado pode apenas assumir dois valores, `#t` (verdadeiro) ou `#f` (falso):

- $\#t$ , se  $\text{ang}$  for menor que 90;
  - $\#f$ , se  $\text{ang}$  for maior que ou igual a 90.

Se  $\text{ang}$  for menor que 90, da expressão de relação ( $< \text{ang} \ 90$ ) resulta  $\#t$  e então é visualizado "faz o lançamento". Contudo, se daquela expressão de relação resultar  $\#f$ , será visualizado "projectil mal orientado".

As expressões de que resultam valores booleanos, #t e #f, têm a designação de predicados. Por exemplo, ( $<$  30 40) é um predicado com valor #t, pois 30 é menor que 40, e ( $<$  30 30) é um predicado com valor #f, pois 30 não é menor que 30.

Pode agora identificar mais uma forma especial do Scheme, if

```
(if expressão-predicado  
    expressão-consequente  
    expressão-alternativa )
```

A regra de cálculo da forma especial `if` é a seguinte:

- calcula a expressão-predicado
  - se resultar  $\#t$ , calcula a expressão-consequente
  - se resultar  $\#f$ , calcula a expressão-alternativa

A forma especial `if` pode não ter expressão-alternativa, situação equivalente a "Se certo facto se verificar, então tomo a decisão x". Isto significa que "se tal facto não se verificar", nenhuma decisão será tomada, continuando a sequência no passo seguinte.



Apresente a regra de cálculo da forma especial `if` na ausência de expressão-alternativa.  
Indique um exemplo da vida real em que isto se possa aplicar.

## Operadores de relação

Os operadores de relação disponibilizados pelo Scheme são: > (maior), < (menor), = (igual), >= (maior ou igual), e <= (menor ou igual).

```
> (if (> 5 12)
      5
      (* 2 7))
```

```
> (if (<= 5 12)
      5
      (* 2 7))
5
```

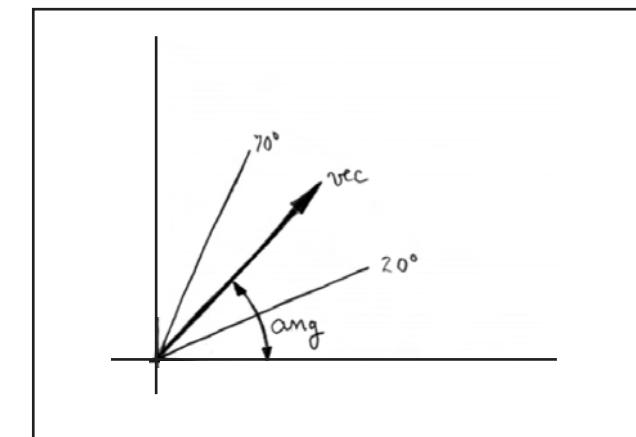


Experimente a forma especial `if` e os operadores de relação e não esqueça que pode procurar, no **Anexo A**, alguma informação adicional sobre esta matéria.

## Compor condições com operadores booleanos

Retomando o exemplo do lançamento do projétil, pretende-se agora garantir que o lançamento só deverá ocorrer se o ângulo se situar entre  $20$  e  $70^\circ$  ou, mais exactamente, se o ângulo de lançamento for maior que  $20$  e menor que  $70$ .

```
(if (and  
    (> ang 20)  
    (< ang 70))  
  
    (display "faz o lançamento")  
  
    (display "projectil mal orientado"))
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

A expressão-predicado é, neste exemplo, uma expressão booleana que, como acontecia com as expressões de relação, também origina resultados booleanos. Para que desta expressão booleana and resulte #t é necessário que se verifiquem simultaneamente as duas condições: ang maior que 20 e ang menor que 70.

Os operadores booleanos são especialmente indicados para definir condições compostas.

Se altura definir a altura de uma pessoa:

```
>(if (and
        (>= altura 160)
        (<= altura 180))
    (display "altura média")
    (display "ou muito alto ou muito baixo!!!"))

> (define altura 120)
> (if (and
        (>= altura 160)
        (<= altura 180))
    (display "altura média")
    (display "ou muito alto ou muito baixo!!!"))
ou muito alto ou muito baixo!!!
> (define altura 200)
> (if (and
        (>= altura 160)
        (<= altura 180))
    (display "altura média")
    (display "ou muito alto ou muito baixo!!!"))
ou muito alto ou muito baixo!!!
> (define altura 170)
> (if (and
        (>= altura 160)
        (<= altura 180))
    (display "altura média")
    (display "ou muito alto ou muito baixo!!!"))
altura média
>
```

Os operadores booleanos disponibilizados pelo Scheme são: and (e), or (ou), not (não).



## INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Escreva (e teste com a ajuda do Scheme) uma expressão que devolva #t quando dois valores numéricos x e y forem diferentes e devolva #f quando forem iguais.



Escreva (e teste com a ajuda do Scheme) uma expressão que devolva #t apenas quando o valor numérico x se situar entre 0 e 10 (incluindo os valores 0 e 10), excepto se o valor numérico y for negativo. Em todas as outras situações a expressão devolve #f.

Faz parte da cultura Scheme terminar com ? o nome dos procedimentos que são predicados, ou seja, que devolvem valores booleanos. O próprio Scheme dá o exemplo, com os predicados que disponibiliza: negative?, positive?, zero?, even?, odd?, boolean?, symbol?, procedure?, number?, integer?, real? e outros, como pode ver no [Anexo A](#).



Experimente a forma especial if, os operadores de relação, os operadores booleanos e predicados disponibilizados pelo Scheme.

**Pista:** No [Anexo A](#) procure os operadores de relação, os operadores booleanos e alguns predicados que o Scheme oferece.



Se conseguiu experimentar e entender o predicado symbol?, muito bem.

Se não conseguiu, não se preocupe, pois terá outras oportunidades mais adequadas para ver este assunto.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 1

Com a ajuda do [Anexo A](#), indique as respostas às seguintes expressões e confirme depois, com a ajuda do próprio Scheme, a correcção das suas respostas.

> (< 34 34)

??

> (and #t #t)

??

> (or #t #t)

??

> (not (even? 2))

??

> (number? 3.5)

??

> (boolean? 3.0)

??

> (positive? (\* 2 3.5))

??

> (<= 34 34)

??

> (and #f #f)

??

> (or #f #f)

??

> (not (odd? 2))

??

> (integer? 3.5)

??

> (boolean? (< 3 3))

??

> (negative? (\* 2 3.5))

??

> (> 34 34)

??

> (and #t #f)

??

> (or #t #f)

??

> (and (odd? 3) (even? 8))

??

> (real? 3.5)

??

> (boolean? "2")

??

> (not (zero? (\* 3 3.5)))

??



Confirme as respostas relativas às expressões do Exercício 1.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 2

O procedimento `circulo`, sem parâmetros, principia por pedir o raio de um círculo e termina visualizando no ecrã a área do círculo e o respectivo perímetro. Este procedimento verifica se o utilizador fornece um valor numérico adequado.

```
> (circulo)
raio (indicar um valor numérico positivo): dez
Erro: Deve indicar um valor numérico positivo.
> (circulo)
raio (indicar um valor numérico positivo): -10.5
Erro: Deve indicar um valor numérico positivo.
> (circulo)
raio (indicar um valor numérico positivo): 10
Área do circulo é': 314.1592653589793
Perímetro do circulo é': 62.83185307179586
```

O valor do raio é introduzido através do teclado, sendo lido com `read` (ver [Anexo A](#)).



Desenvolva e teste o procedimento `circulo`.

### Exercício 3

A área de um triângulo pode determinar-se a partir do comprimento dos seus lados  $a$ ,  $b$ , e  $c$ , com a fórmula que se apresenta, em que  $sp$  é o semi-perímetro do triângulo.

$$area = \sqrt{sp * (sp - a) * (sp - b) * (sp - c)}$$

O procedimento `area-triangulo` devolve a área do triângulo, em que os comprimentos dos três lados são fornecidos através do teclado.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

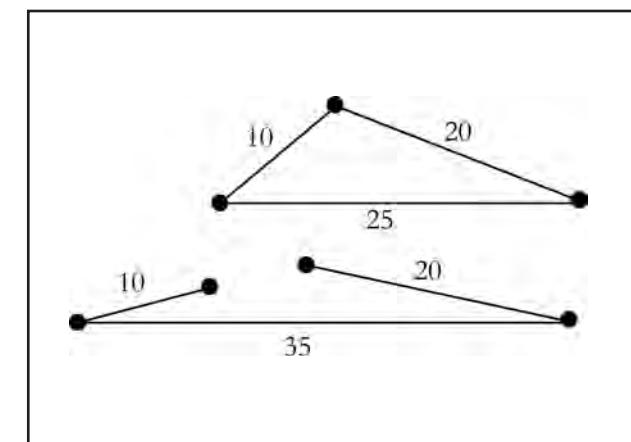
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (area-triangulo)
lado1: 10
lado2: 20
lado3: 25
94.99177595981665
```

```
> (area-triangulo)
lado1: 10
lado2: 20
lado3: 35
0
```

Este procedimento verifica se os argumentos fornecidos são compatíveis com os comprimentos dos lados de um triângulo, ou seja, nenhum deles pode ser igual ou superior à soma dos outros dois. Em caso de incompatibilidade, o resultado deverá ser 0.



Escreva em Scheme o procedimento `area-triangulo` conforme a especificação apresentada.

Apresente o diagrama de fluxo de dados que reflete a ligação entre este procedimento e outros que venha a utilizar.



Desenvolva e teste o procedimento `area-triangulo`.

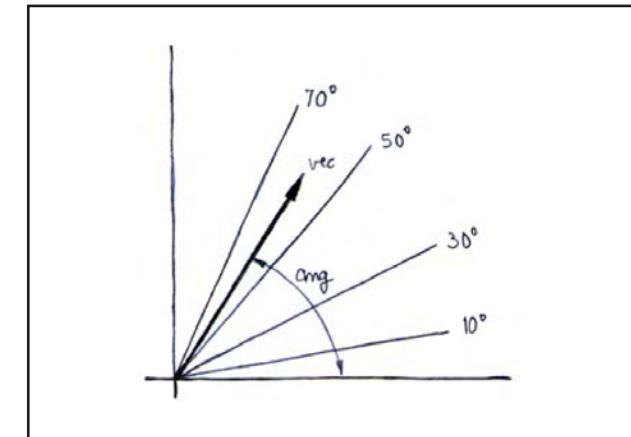
**Pista:** Sugere-se a utilização de um procedimento auxiliar para validar se os comprimentos fornecidos correspondem ou não aos lados de um triângulo.



**Sequência de ifs quando as decisões envolvem mais do que duas hipóteses**

Retomando, mais uma vez, o exemplo do lançamento do projétil, pretende-se agora garantir os seguintes tipos de lançamento:

- Se  $\alpha$  for maior que  $70^\circ$  ou menor ou igual que  $10^\circ$ , não haverá lançamento;
  - Se  $\alpha$  for menor que ou igual a  $70^\circ$  e maior que  $50^\circ$ , haverá um lançamento a grande altitude;
  - Se  $\alpha$  for menor que ou igual a  $50^\circ$  e maior que  $30^\circ$ , haverá um lançamento a média altitude;
  - Se  $\alpha$  for maior que ou igual a  $30^\circ$  e maior que  $10^\circ$ , haverá um lançamento a baixa altitude.



Para ter em conta todas estas condições, definiu-se o procedimento projectil

Verifique que a expressão-alternativa do primeiro if é, ela própria, um novo if (isto é posto em evidência pelos 3 rectângulos a tracejado).

Por sua vez, a expressão-alternativa do segundo `if` também é um `if` (situação posta em evidência pelos 3 rectângulos azuis).

```
(define projectil
  (lambda (ang)
    (if (or (> ang 70)
            (<= ang 10))
        (display "projectil mal orientado")
        (if (> ang 50)
            (display "lancamento a grande altitude")
            (if (> ang 30)
                (display "lancamento a media altitude")
                (display "lancamento a baixa altitude"))))))
```

The diagram illustrates the structure of the Scheme code. Two callouts point to specific parts of the code:

- A black callout points to the first `if` expression, labeled "expressão-alternativa do primeiro if". This `if` checks if the angle is greater than 70 or less than or equal to 10. If true, it displays "projectil mal orientado".
- A blue callout points to the second `if` expression, labeled "expressão-alternativa do segundo if". This `if` checks if the angle is greater than 50. If true, it displays "lancamento a grande altitude". If false, it proceeds to the third `if` expression.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (projectil 10)
projectil mal orientado
> (projectil 40)
lancamento a media altitude
> (projectil 70)
lancamento a grande altitude
> (projectil 120)
projectil mal orientado
```

Se `ang` for maior que 70 ou menor que ou igual a 10, então o projétil estará mal orientado e não haverá lugar a qualquer lançamento. No entanto, se nenhuma destas condições se verificar, o projétil será lançado, mas tornar-se-á necessário distinguir a situação de lançamento, sabendo que existem três hipóteses possíveis: lançamento a grande, média ou baixa altitude.

Verifique que, no segundo `if`, basta que `ang` se apresente maior que 50 para ter a garantia de que o lançamento é a grande altitude. É que, chegado a este ponto, `ang` não pode ser maior que 70, pois não foi apanhado no primeiro `if`. Se `ang` não for maior que 50, ainda há duas situações a distinguir e, por isso, justifica-se um terceiro `if`. Neste `if`, se `ang` for maior que 30, está garantido o lançamento a média altitude, caso contrário, o lançamento será necessariamente a baixa altitude.



Experimente o procedimento `projectil`.

Apenas para treino, uma vez que só vai complicar, experimente alterar o procedimento `projectil` para não utilizar operadores booleanos, mas apenas operadores de relação.



Por que razão, no terceiro `if`, a expressão-alternativa é "lancamento a baixa altitude", sem verificar se `ang` é maior que 10?



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## A forma especial cond para evitar a baixa legibilidade da sequência de ifs

Não se poderá dizer que o procedimento `projectil`, baseado numa sequência de três `if`, seja muito legível. Com algum esforço, até se entende o que faz, mas em situações como esta é preferível utilizar a forma especial `cond`, que oferece uma legibilidade muito maior.

```
(define projectil-melhorado
  (lambda (ang)
    (cond ((or
            (> ang 70)
            (<= ang 10))
           (display "projectil mal orientado"))
          ((> ang 50)
           (display "lancamento a grande altitude"))
          ((> ang 30)
           (display "lancamento a media altitude"))
          (else
           (display "lancamento a baixa altitude")))))
```

Os procedimentos `projectil` e `projectil-melhorado` respondem exactamente da mesma maneira, mas este último apresenta as várias situações de forma muito mais evidente. A leitura é muito mais fácil.



Experimente o procedimento `projectil-melhorado`.



A forma especial cond do Scheme pode sintetizar-se

```
(cond (predicado-1 exp1-1 exp1-2 ...) ; cláusula 1  
.....  
     (predicado-2 exp2-1 exp2-2 ...) ; cláusula 2  
.....  
     ...  
.....  
     (else exp-else-1 exp-else-2 ...) ) ; cláusula else
```

Quanto à regra de cálculo da forma especial cond:

- Calcula os predicados, começando por `predicado-1`, até encontrar um com valor `#t`;
  - Logo que se encontre um predicado `#t`, calcula as expressões correspondentes;
  - Se não encontrar qualquer predicado com valor `#t`, calcula as expressões de `else`.

A cláusula `else` é opcional. Se não existir e se do cálculo de todos os predicados resultar sempre `#f`, nenhuma das expressões de `cond` será calculada.

**if** versus cond... ou uma oportunidade para introduzir a forma especial begin

Sobre as formas especiais `if` e `cond`, ficou claro que se deve evitar `if` em situações com mais de duas opções, por apresentar uma leitura mais difícil do que a forma `cond`.

E mesmo com duas opções, deve pensar-se duas vezes, sobretudo quando a expressão-consequente ou a expressão-alternativa integram mais do que uma expressão, situação em que a forma `if` obriga à utilização de `begin`.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(if ((< x 5)
     (begin
       (display " x: ")
       (display x)
       (display " menor que 5"))
     (begin
       (display "neste caso, x: ")
       (display x)
       (display " maior ou igual que 5"))))
```

```
(cond ((< x 5)
        (display " x: ")
        (display x)
        (display " menor que 5"))
      (else
        (display "neste caso, x: ")
        (display x)
        (display " maior ou igual que 5"))))
```



No exemplo identificou mais uma forma especial do Scheme.  
De que forma se trata? Consegue explicar como funciona?

begin é mais uma das formas especiais do Scheme que se apresenta da seguinte maneira:

```
(begin
  expressão-1
  expressão-2
  ...
  expressão-n)
```

begin garante que estas expressões são calculadas em sequência, desde a primeira até à última, e devolve o valor da última expressão.

Este tipo de sequência, desde a primeira até à última expressão, está implícito no corpo dos procedimentos, quando contém mais do que uma expressão, e também nas cláusulas de cond, casos em que o begin é dispensável.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Exercício 4

Um procedimento recebe dois valores e visualiza uma mensagem que exprime a relação de grandeza entre eles. É apresentada uma solução para este procedimento, com o nome `comparador-com-if`, que recorre apenas à forma `if`.

```
> (comparador-com-if 2 3)
3 maior que 2
> (comparador-com-if 3 2)
3 maior que 2
> (comparador-com-if 3 3)
ambos iguais a 3
```

```
(define comparador-com-if
  (lambda (x y)
    (if (> x y)
        (begin
          (display x)
          (display " maior que ")
          (display y))
        (if (> y x)
            (begin
              (display y)
              (display " maior que ")
              (display x)))
            (begin
              (display "ambos iguais a ")
              (display x))))))
```

Escreva agora o procedimento `comparador-com-cond` que, em vez de `if`, utiliza a forma `cond` e responde da mesma maneira que o comparador apresentado.



Desenvolva e teste o procedimento `comparador-com-cond`.

**Pista:** Aproveite para avaliar o grau de legibilidade de cada uma das versões.



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

### Exercício 5

O procedimento `rectangulo-maior` tem como parâmetros `lado-a1`, `lado-a2`, `lado-b1` e `lado-b2`, em que os dois primeiros correspondem ao comprimento de lados do rectângulo A e os dois últimos correspondem ao comprimento de lados do rectângulo B.

Escreva este procedimento em Scheme, sabendo que ele calcula a área de cada um dos rectângulos, que as compara e que responde da seguinte maneira:

```
> (rectangulo-maior 10 20 15 5)
Rectangulo A: 200
Rectangulo B: 75
O rectangulo A maior 125 unidades.
```

```
> (rectangulo-maior 10 20 15 18)
Rectangulo A: 200
Rectangulo B: 270
O rectangulo B maior 70 unidades.
```

```
> (rectangulo-maior 10 20 40 5)
Rectangulo A: 200
Rectangulo B: 200
Os rectangulos apresentam igual area.
```



Desenvolva e teste o procedimento `rectangulo-maior`.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 6

Este é idêntico ao exercício 5, mas agora o procedimento não tem parâmetros e tem o nome `rectangulo-maior-sem-parametros`. O comprimento de cada lado é fornecido através do teclado. Imagine um diálogo apropriado e escreva o procedimento.

Apresente o diagrama de fluxo de dados que reflecta a ligação entre os procedimentos utilizados.



Desenvolva e teste o procedimento `rectangulo-maior-sem-parametros`.

### Exercício 7

Um desafio um pouco mais complicado... com muitas variáveis locais.

Quer saber em que dia da semana ocorreu o 25 de Abril de 1974, o dia da Revolução dos Cravos em Portugal?

Ou quer saber em que dia da semana nasceu, sem perguntar aos seus pais?

Para determinar o dia da semana de uma data do calendário, utiliza-se um algoritmo que agora se descreve. Mas, antes disso:

- $m$  = mês do ano, em que Março é o mês 1, Abril 2, até Dezembro que é o mês 10. Janeiro e Fevereiro são considerados os meses 11 e 12 do ano anterior (Esta identificação dos meses, perfeitamente anormal, é apenas utilizada dentro do algoritmo);
- $d$  = dia do mês;
- $a$  = ano do século;
- $s$  = século anterior (mais uma identificação anormal, para utilizar no algoritmo).

Por exemplo, para 4 de Julho de 1989 será  $m=5$ ,  $d=4$ ,  $a=89$ ,  $s=19$ . Por outro lado, para 4 de Janeiro do mesmo ano será  $m=11$ ,  $d=4$ ,  $a=88$ ,  $s=19$ .

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Veja agora os passos do algoritmo.

Seja:

- mint = parte inteira de  $(13*m-1)/5$ ;
- aint = parte inteira de  $a/4$ ;
- sint = parte inteira de  $s/4$ ;
- $x = \text{mint} + \text{aint} + \text{sint} + d + a - 2*s$ ;
- dia = resto da divisão inteira  $x/7$ .

dia é a resposta, de acordo com a identificação seguinte: dia=0 é o Domingo, dia=1 é 2<sup>a</sup>-feira e assim sucessivamente até dia=6 que corresponde a Sábado.

Escreva em Scheme o programa dia-da-semana que, em relação a uma data, pede o dia, mês e ano e responde com o respectivo dia da semana. Na chamada que se segue, a data em causa é 18 de Agosto de 2001, dia que ocorreu a um Sábado.

```
> (dia-da-semana)
dia: 18
mes: 8
ano: 2001
dia da semana: 6
```



Desenvolva e teste o programa dia-da-semana.

## **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

R E 1 2 3 4 5 6 7

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

## **Módulo 1.6 - Saber lidar com os erros!**

Muito dificilmente se escapa aos erros em programação, erros que são normalmente classificados nos seguintes tipos: erros de sintaxe, erros de execução e erros lógicos.

Enquanto que a identificação dos dois primeiros tipos de erros é facilitada pelo próprio sistema em que se desenvolvem os programas, os erros lógicos só o programador os poderá identificar. Para isso, deve perceber muito bem o problema que se pretende resolver e seguir uma concepção e desenvolvimento muito cuidados de todo o programa. É ainda fundamental que desenvolva manualmente um conjunto de testes, que ajudarão na identificação dos erros na fase de teste.

Sobre um exemplo, procurar-se-á, no decorrer deste módulo, abordar este tema importante da programação.

A utilização do Scheme é incentivada neste módulo, pois vai permitir um contacto mais forte com os erros, no que se refere à respectiva identificação e posterior correcção.

### **Palavras-Chave**

Erro de sintaxe, erro de execução, erro lógico.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Para tentar evitar os erros em programação

Os erros em programação são praticamente inevitáveis e podem ser classificados em vários tipos: erros de sintaxe, erros de execução e erros lógicos.

Os erros de sintaxe estão relacionados com situações em que as regras da linguagem são desrespeitadas. As falhas de parêntesis são os erros de sintaxe mais comuns em Scheme, mas também são os mais fáceis de ultrapassar. São normalmente detectados quando se activa o comando de execução do Scheme e o próprio sistema assinala a sua possível localização.

Os erros de execução só ocorrem quando a execução do programa passa por eles. Ou seja, um programa pode até ser executado várias vezes e este tipo de erro não ocorrer, basta que a execução não passe por ele. Como exemplo, aponta-se uma divisão em que o divisor é zero.

Os dois tipos de erros referidos são normalmente detectados por um bom sistema de apoio ao desenvolvimento de programas, o que já não acontece com os erros lógicos, normalmente associados a uma definição imperfeita, cometida pelo programador. No caso dos erros lógicos, o programa vai dando resultados, alguns até podem estar correctos, mas outros não, e caberá ao programador determinar se o funcionamento do programa é ou não perfeito, sendo da sua responsabilidade a identificação destes erros. Assim, só uma análise aprofundada do problema, que ajude a perceber muito bem o que se pretende resolver, seguida de uma concepção cuidada do programa poderá evitar este tipo de erro.

Para além disto, o desenvolvimento manual de um bom conjunto de testes, um que procure cobrir todos os tipos de situações em que o programa irá trabalhar, é uma boa forma de preparar o terreno para detectar os erros lógicos.

## Abordar os erros através de um exemplo

Entender correctamente o problema que se pretende resolver é fundamental para não se correr o risco de resolver, mesmo que bem, outra coisa que não a que nos interessa.

Pretende-se desenvolver um programa que calcule e visualize o vencimento semanal de um trabalhador de uma empresa, tendo como parâmetro o número de horas de trabalho semanal. O vencimento semanal é definido pela aplicação de três tabelas.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

- Tabela 1 - Menos de 30 horas por semana, o pagamento é a 10€/hora.
- Tabela 2 - Entre 30 e 36 horas por semana, as primeiras 29 horas são pagas pela Tabela 1 e as restantes têm um bónus adicional de 3€, ou seja, 13€/hora.
- Tabela 3 - Mais de 36 horas por semana, as primeiras 36 são pagas pelas Tabelas 1 e 2 e as restantes têm um bónus que é metade do bónus considerado para a Tabela 2.

A visualização terá em conta o nome do trabalhador e assumirá o aspecto que se indica:

Da chamada do programa com os argumentos "António Silva" e 30 resulta

**António Silva trabalhou 30 horas durante a semana.  
O seu vencimento semanal foi de 303.0 euros.**

### Preparar um conjunto de testes

O desenvolvimento manual de algumas situações do problema apresenta duas vantagens importantes, ajuda a perceber melhor o problema em questão e origina um conjunto de testes a que se submeterá o programa durante ou no final do seu desenvolvimento, no sentido de se verificar se funciona correctamente para todas as situações.

Convém prever, pelo menos, uma situação em cada intervalo de pagamento e também dar realce às situações de fronteira. Com esta preocupação em mente, consideraram-se as seguintes situações:

- 29 horas - último valor da Tabela 1, portanto  $29 * 10 = 290\text{€}$ .
- 30 horas - primeiro valor da Tabela 2, ou seja  $29 * 10 + 1 * 13 = 303\text{€}$ .
- 36 horas - último valor da Tabela 2, portanto  $29 * 10 + 7 * 13 = 381\text{€}$ .
- 37 horas - primeiro valor da Tabela 2, portanto  $29 * 10 + 7 * 13 + 1 * 11.5 = 392.5\text{€}$ .



**Acrescente mais algumas situações de teste e justifique a escolha que fez.**



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Procurar uma ideia para resolver o problema do vencimento semanal

Se achar que entendeu bem o problema proposto, é altura de tentar encontrar uma ideia para o resolver. Se desenvolver um procedimento auxiliar que calcule o vencimento semanal, chega a um algoritmo relativamente simples para o problema que quer resolver.

Passo 1 - visualiza linha com o nome do trabalhador e com o número de horas que trabalhou

Passo 2 - visualiza linha com o vencimento semanal, calculado com uma chamada ao referido procedimento auxiliar

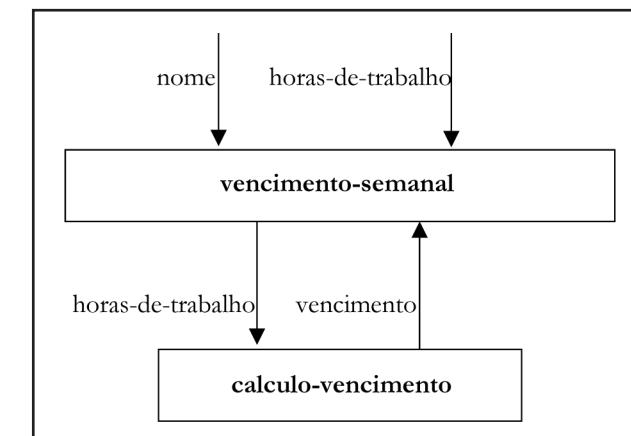
### Escrever o programa em Scheme

Tendo em conta o que ficou estabelecido, opta-se por um programa que se concretiza num procedimento com o nome `vencimento-semanal`, com os parâmetros `nome` e `horas-de-trabalho`.

O procedimento `vencimento-semanal` pode apoiar-se num procedimento auxiliar, designado por `calculo-vencimento`, com o parâmetro `horas-de-trabalho`. Este procedimento devolve o valor do vencimento semanal. O diagrama de fluxo de dados mostra a relação entre estes dois procedimentos.

Vamos tentar uma primeira solução para o problema.

```
; calcula e visualiza o vencimento semanal
; parâmetros: nome (do trabalhador) e
;              horas-de-trabalho
;
(define vencimento-semanal
  (lambda (nome horas-de-trabalho)
    (display nome)
    (display " trabalhou ")
    (display horas-de-trabalho)
    (display " horas durante a semana.")
    (newline)
    (display "O seu vencimento semanal foi de ")
    (display (calculo-vencimento horas-de-trabalho))
    (display " euros.")))
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
; calcula o vencimento semanal
; parâmetro: horas (horas de trabalho realizadas)
; devolve o vencimento semanal
;
(define calculo-vencimento
  (lambda (horas)
    (let ((bonus1 3.0)                                ; bonus
          (limite1 29)
          (limite2 36)
          (tabela1 10.0))                             ; Tabela 1
      (let ((tabela2 (+ tabela1 bonus1))            ; Tabela 2 (bonus)
            (tabela3 (+ tabela1 (/ bonus1 0)))))     ; Tabela 3 (metade do bonus)
        ;
        (cond ((<= horas limite1)
                (* horas tabela1))      ; menos de 30 horas
              ;
              ((<= horas limite2)
               (+ (* limite1 tabela1)           ; entre 30 e 36 horas
                   (* (- horas limite1)
                       tabela2)))
              ;
              (else
               (+ (* limite1 tabela1)           ; mais de 36 hora
                   (* 6 tabela2)
                   (* (- horas limite2)
                       tabela3))))))))
```



Experimente os procedimentos vencimento-semanal e calculo-vencimento.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Ao activar o comando de execução do Scheme resultou uma mensagem de erro, semelhante à que se segue.

```
Welcome to DrScheme, version 203.  
Language: Graphical (MrEd, includes MzScheme).  
read: expected a ')'
```

Este erro de sintaxe foi facilmente ultrapassado, juntando um parêntesis, que está em falta, na última linha do procedimento.

```
Welcome to DrScheme, version 203.  
Language: Graphical (MrEd, includes MzScheme).  
>
```

Passemos agora à primeira situação de teste, ou seja, a determinação do vencimento para 29 horas. Uma vez que o problema foi decomposto por forma a utilizar-se o procedimento auxiliar `calculo-vencimento`, será por este procedimento que os testes principiarão. Depois de garantido o bom funcionamento deste procedimento, passar-se-á ao teste do procedimento `vencimento-semanal`.



**Justifique a decisão referente à sequência de teste dos procedimentos.**

Passemos agora à primeira situação de teste, ou seja, a determinação do vencimento para 29 horas de trabalho semanal.

```
> (calculo-vencimento 29)  
/: division by zero  
>
```

A execução do procedimento passou por uma situação de erro, no cálculo do pagamento correspondente à Tabela 3. Por engano, em vez de dividir por 2, dividiu-se por zero.



**Se em vez de zero fosse, por exemplo, 5 teríamos um erro que não seria detectado pelo sistema! Não seria um erro de execução. De que tipo de erro se trataria?**

# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Corrigido o erro de execução referido, tente de novo, o comando de execução do Scheme, para retomar o primeiro teste e passar aos seguintes.

```
> (calculo-vencimento 29)  
290.0  
> (calculo-vencimento 30)  
303.0  
> (calculo-vencimento 36)  
381.0  
> (calculo-vencimento 37)  
379.5  
>
```

O último teste não está de acordo com o valor calculado manualmente, pois deveria dele resultar 392.5€. Aliás, o erro é de tal ordem que o pagamento de 37 horas apresenta um valor inferior a 36 horas, o que é impossível de acordo com as regras de pagamento estabelecidas.



**Antes de continuar a leitura do texto, procure o erro e uma solução para o mesmo.  
Se conseguir, o parágrafo que se segue é perfeitamente dispensável.**

Fazendo incidir a nossa atenção no extracto do procedimento correspondente à Tabela 3, com mais ou menos dificuldade concluirá que o intervalo da Tabela 2, de 30 a 36 horas, não é de 6 mas sim de 7 horas, o que não está de acordo com (\* 6 tabela2). O erro lógico está no 6 que deveria ser 7, ficando (\* 7 tabela2). Com esta correção e testando novamente todas as situações verificar-se-ia um sucesso completo. Assim, poderíamos concluir, com forte probabilidade de não nos enganarmos, que o procedimento cumpre cabalmente o que dele se espera, uma vez que o conjunto de testes reflecte bem a realidade do problema que se pretendia resolver.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Em vez da correcção (\* 7 tabela2) proponha uma solução mais genérica, que não exija alteração se limite1 ou limite2 vierem a ser alterados.

Finalmente, o teste do procedimento vencimento-semanal:

```
> (vencimento-semanal ''Joaquim Silva'' 29)  
Joaquim Silva trabalhou 29 horas durante a semana.  
O seu vencimento semanal foi de 290.0 euros.  
>
```



Diga por que razão bastou apenas um teste do procedimento vencimento-semanal, não adiantando testar as restantes situações.



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 1 - vender parafusos

Escreva um procedimento em Scheme imaginando que um pequeno lojista o pretende usar no cálculo do custo de parafusos. O procedimento tem como parâmetros a quantidade de parafusos e o custo unitário. Até 10 parafusos, o lojista não faz qualquer desconto. Acima de 10 e até 50, faz um desconto constante equivalente a 3 parafusos. Acima de 50, o desconto é de 10% sobre o custo total.

Recomenda-se o desenvolvimento manual de situações que servirão de base aos testes do procedimento, e não se surpreenda se, com os descontos definidos, surjam alguns casos em que a uma quantidade maior corresponde um custo menor!...



Desenvolva e teste o procedimento pedido.

### Exercício 2 - com um pouco mais de dificuldade...

Desenvolva um procedimento idêntico ao do exercício anterior, mas agora imaginando que o lojista introduz algumas alterações aos descontos, de tal forma que a uma quantidade maior que outra nunca corresponda um custo menor. Segundo as novas regras de descontos, pagar-se-á pelo menos tanto quanto o correspondente ao valor máximo do patamar anterior. Por exemplo, acima de 10 parafusos, no patamar de desconto de 3 unidades, o custo de 11, 12 ou 13 parafusos será o mesmo que o custo de 10. Também neste caso se recomenda o desenvolvimento manual de situações que servirão de base aos testes do novo procedimento.



Desenvolva e teste o procedimento pedido.

## **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

R E 1 2 3 4 5 6 7

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

## **Módulo 1.7 - Introdução à programação através de abstracções**

Convém entender, desde já, que a programação não se resume à codificação de um programa numa determinada linguagem. A programação é, sobretudo, uma actividade de reflexão, onde a linguagem de implementação não é sequer o elemento mais importante. Perante um problema, é necessário, em primeiro lugar, encontrar uma ideia que apoie a sua resolução e, de seguida, traçar uma estratégia para pôr essa ideia em movimento.

Neste módulo pretende-se mostrar que a programação é, de facto, uma actividade criativa, que se alimenta de ideias, onde, com um conhecimento mínimo de uma linguagem (neste caso, o Scheme) e através de algumas abstracções orientadas para os problemas, é possível programar situações já com alguma complexidade, nomeadamente atravessar percursos e labirintos, muitas vezes com o acompanhamento de sons a ilustrar o que vai ocorrendo em cada momento.

### **Palavras-Chave**

Produção de sons, abordagem de-cima-para-baixo (*top-down*), labirintos, procedimentos recursivos.



# **S C H E M E**

## **na descoberta da programação**

**INTRODUÇÃO**

**1 - O ESSENCIAL DO SCHEME**

R E 1 2 3 4 5 6 7

**2 - RECURSIVIDADE**

**3 - ABSTRACÇÃO DE DADOS**

**4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE**

**5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS**

**6 - SCHEME E OUTRAS  
TECNOLOGIAS**

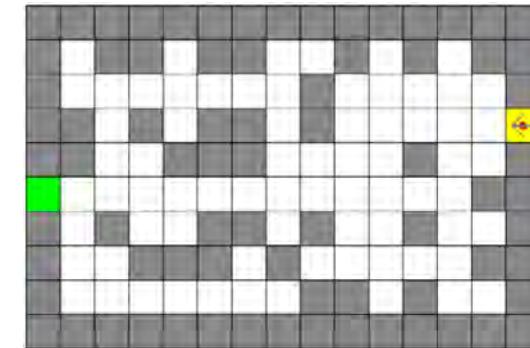
**7-EXERCÍCIOS  
E  
PROJECTOS**

**ANEXOS**

**Programação é, em primeiro lugar, a procura de ideias para resolver problemas**

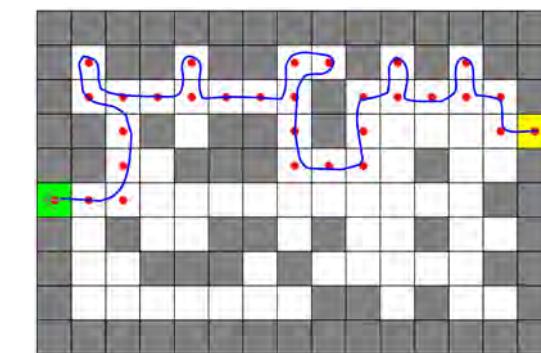
O problema que tem agora na sua frente é um labirinto visualizado num tabuleiro. As células pintadas a cinzento representam obstáculos, paredes que não permitem a passagem. Pelo contrário, as células pintadas a branco mostram os caminhos livres.

A célula verde é o objectivo, pois é necessário conduzir uma nave até esta célula, nave essa inicialmente colocada na célula amarela e já a apontar para dentro do labirinto.



**Pode constatar que há mais do que um percurso possível entre a célula amarela e a célula verde.  
Como programar a nave para seguir um deles, mesmo que não seja o melhor?**

Parece que alguém já descobriu uma ideia para levar a nave da célula amarela até à célula verde. Terá ainda pegado nessa ideia e, com a linguagem Scheme, programou a nave, tendo obtido o resultado que pode observar na figura. A linha azul foi desenhada para mostrar o percurso seguido pela nave, notando-se a passagem repetida em várias células.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

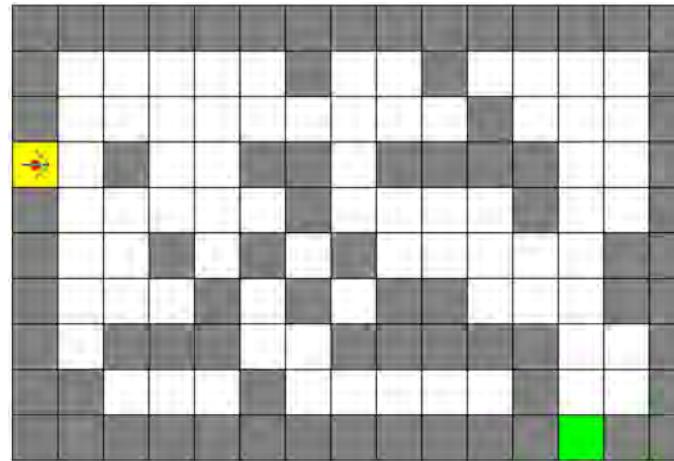
### ANEXOS



Imagine que a nave não só se descola em frente, mas pode também rodar (90°), tanto para a direita como para a esquerda, ou seja, pode usar os três comandos seguintes: **frente**, **roda-dir** e **roda-esq**.

O desafio que agora lhe é colocado é o de descobrir uma ideia própria, o que seria o ideal, para atravessar o labirinto, ou então tentar descobrir a ideia utilizada anteriormente.

Utilize neste exercício a figura que se segue.



Se não conseguiu encontrar uma ideia para resolver o labirinto, adianta-se uma sugestão.

No novo labirinto, "coloque-se em cima da nave", mantenha a sua mão direita sempre encostada à parede e vá fazendo avançar a nave, utilizando os três comandos referidos...



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Se conseguiu encontrar uma ideia que lhe permita conduzir a nave até à célula verde, tanto melhor. Mas se não conseguiu também não é motivo para grande preocupação, pois voltará a encontrar este problema mais à frente, ainda neste módulo. O importante é ficar a saber que, com um pequeno conjunto de comandos relacionados com naves, o labirinto pode ser atravessado, atacando o problema no domínio de uma linguagem próxima do problema concreto (controlo de naves: frente, roda, ...) e não no domínio de uma linguagem de programação que, de tão genérica, pouco tem a ver com esse problema. E isto acaba por resumir a importância das abstracções na programação, neste caso particular, a abstracção naves, que será, em breve, alvo da nossa atenção.

### Produção de sons - Abstracção áudio

Imagine que pretendia desenvolver um programa que expressasse alguns dos seus resultados através de sons. Uma calculadora que diz o resultado em vez de usar o ecrã, uma nave que emite sons quando se desloca ou a personagem de um jogo que exprime emoções de contentamento ou agradecimento. Provavelmente, gostaria de ter acesso a alguma funcionalidade básica, que lhe permitisse este tipo de programação.



Observe a figura e tente perceber o funcionamento do procedimento `som...` o Scheme não disponibiliza esta funcionalidade e teria que a desenvolver se alguém ainda não o tivesse feito.

O acesso ao procedimento `som` faz-se através de:

```
(require (lib "audio.scm" "user-feup"))
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Por agora, bastar-lhe-á saber que o ficheiro `audio.scm`, com a definição do procedimento `som`, se encontra no directório do Scheme designado por `PLT\collects\user-feup`.

Caso não possua equipamento para escutar os sons reproduzidos no seu computador ou se não quiser incomodar as pessoas que estejam próximas, substitua

(`require (lib "audio.scm" "user-feup")`)

por

(`require (lib "noaudio.scm" "user-feup")`)



Teste a abstracção áudio.

Experimente chamadas ao procedimento `som` como as indicadas:

```
> (som "sorry")
> (som "direita")
> (som "andal")
```

A funcionalidade desta abstracção resume-se a

```
> (som som-pretendido)
```

em que `som-pretendido` é uma cadeia de caracteres que define o som pretendido e, de momento, pode tomar um dos valores entre:

"andal"	"anda2"	"bomba1"
"chia1"	"chia2"	
"direita"	"e"	"esquerda"
"poing1"	"ri1"	"ri2" "ri3" "rotunda"
"sorry"	"thank_you"	"yaahoo" "yehaaa" "yipee"
"0" "1" "2" "3" "4" "5" "6" "7" "8" "9"		
"10" "11" "12" "13" "14" "15" "16" "17" "18" "19"		
"20" "30" "40" "50" "60" "70" "80" "90"		
"100" - cem "c100" - cento		
"200" "300" "400" "500" "600" "700" "800" "900" "1000"		

qualquer outro valor produzirá o som equivalente a "oh\_no"



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



No Scheme que está a utilizar, pode procurar o directório PLT\collects\user-feup e abrir alguns dos ficheiros, todos eles preparados não só por professores da FEUP, mas também por estudantes...

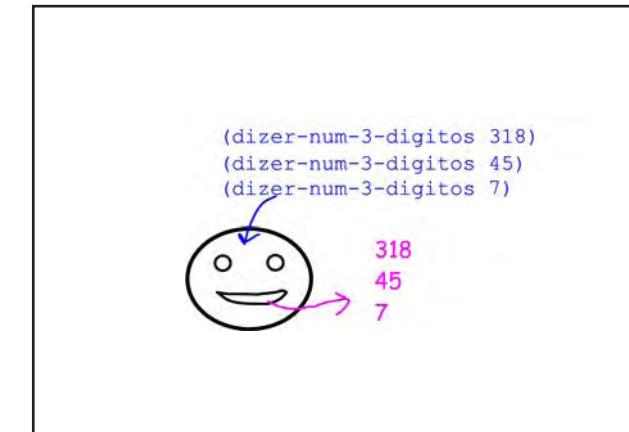
Neste directório estão colocados outros ficheiros que agrupam conjuntos de procedimentos orientados para a resolução de alguma tarefa. Ou seja, cada um desses ficheiros contém, normalmente, a linguagem própria de um certo problema. No caso dos sons, a linguagem resume-se a som, que é o suficiente para pedir a produção de um som. Desta forma, o programador não tem que se preocupar com os pormenores de implementação da produção de sons e dizemos que estamos perante a abstracção áudio.

Bastará, para tal, escrever nos seus programas  
(require (lib "audio.scm" "user-feup"))

Teve um primeiro contacto com a abstracção áudio, e, mais à frente, vai encontrar a abstracção naves e com ela poderá controlar naves. Através de uma linguagem que terá a ver com naves, vai poder rodar e avançar as naves ou ver quem são os seus vizinhos, etc.

### Exemplo 1 - dizer números até 999

O problema que agora tem entre mãos é programar o computador para dizer números inteiros de 1 a 999, como se mostra na figura.



Observe a figura e tente encontrar uma justificação para o nome do procedimento e para o significado dos argumentos que esse procedimento recebe.



# SCHEME na descoberta da programação

## INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Será possível compor os 999 sons com os sons disponibilizados pela abstracção audio?

Antes de responder, analise a forma como se dizem os números... e verifique:

- Uma certa falta de uniformidade ao dizer os números inferiores a 20... cada caso é um caso...
- Depois uma certa regularidade... até 99
- O número 100 é dito "cem", mas de 101 a 199 é dito " cento"...
- Finalmente, de 200 a 999, verifica-se uma certa regularidade.

Tem pela frente o problema de dizer os números de 3 dígitos, números que poderão ir de 1 a 999, problema que apresenta já um certo grau de complexidade. Nestes casos, a estratégia é tentar decompor o problema em problemas mais simples, começando por cima e ir descendo até encontrar problemas de fácil solução. Esta abordagem dá pelo nome de-cima-para-baixo (*top-down*).

Mas antes de atacar o problema de dizer os números de 3 dígitos, comece com um exemplo, com o objectivo de ilustrar a abordagem de-cima-para-baixo.

Imagine que pretende fazer o percurso da casa do José até à casa da Maria, tendo um rio pelo meio.



# SCHEME na descoberta da programação

## INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

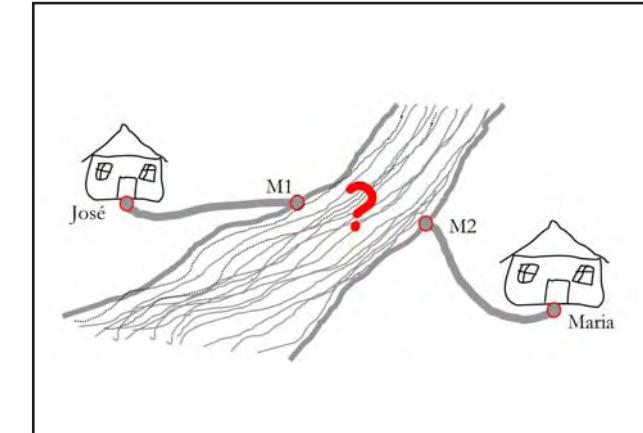
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A parte do percurso terrestre é para fazer a pé e a parte do rio, entre as margens M1 e M2, é mais complicada e ainda não se sabe muito bem como a fazer. Talvez a nado, talvez de barco... Mas para já, sem entrar em pormenores, imagine que tem alguma maneira de atravessar o rio e, sem se importar como, já pode escrever:

Problema - Fazer o percurso da casa do José até à casa da Maria

- ir a pé da casa do José até à margem M1
- sub-problema - atravessar o rio de M1 para M2
- ir a pé da margem M2 até à casa da Maria



O problema inicial é assim considerado resolvido e é chegada a altura de focar a atenção apenas no sub-problema da travessia do rio.

Depois de se reflectir um pouco, em vez de se atravessar a nado, optou-se por fazer a travessia de barco, pois até há barcos para alugar junto às margens M1 e M2.

Atravessar o rio de M1 para M2

- alugar um barco
- entrar no barco em M1
- atravessar o rio de barco de M1 até M2
- sair do barco em M2

Se agora juntar tudo, obtém um algoritmo para o problema inicial.

## INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Problema - Ir da casa do José à casa da Maria

- ir a pé da casa do José até à margem M1
- sub-problema - atravessar o rio de M1 para M2
  - alugar um barco
  - entrar no barco em M1
  - atravessar o rio de barco de M1 até M2
  - sair do barco em M2
- ir a pé da margem M2 até à casa da Maria



Depois de ter entendido este exemplo conseguirá determinar os sub-problemas do problema dizer números de 3 dígitos?

Dizer as centenas do número, dizer as dezenas do número e dizer as unidades do número serão os sub-problemas do problema dizer número de 3 dígitos?

Sem ver o que se segue, tente esboçar um algoritmo para este problema, imaginando que aqueles 3 sub-problemas estão já resolvidos.

Apresenta-se uma hipótese de algoritmo para o problema dizer número de 3 dígitos, após a identificação dos sub-problemas: dizer as centenas do número, dizer as dezenas do número e dizer as unidades do número.

### Problema - dizer número de 3 dígitos

- sub-problema - dizer as centenas do número
- se necessário, ligar centenas com dezenas do número (através do som "e" )
- sub-problema - dizer as dezenas do número...
  - ...incluindo os casos especiais de 10 a 19
- se não for caso de 10 a 19
  - se necessário, ligar dezenas com unidades do número (através do som "e" )
  - sub-problema - dizer as unidades do número

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Foque agora a sua atenção apenas no sub-problema identificado como sendo dizer as centenas do número. Um possível algoritmo será o que se apresenta de seguida.

#### sub-problema - dizer as centenas do número

- se for o caso especial do número 100
  - diz "cem"
- se não, temos os casos que concatenam com as dezenas ou as unidades...
  - separa dígito das centenas
  - se o dígito das centenas for 9, 8, ..., ou 1,
    - assim dirá "900" "800", ..., ou "cento".

```
(define dizer-centenas
  (lambda (n)
    ; se for caso especial 100
    (if (= n 100)
        (som "100")
        ; se não, temos os casos que concatenam com as dezenas ou as unidades... e
        ; separa digito das centenas
        (let ((c (quotient n 100)))
            ; se o dígito das centenas for 9, 8, ..., ou 1,
            ; assim dirá "900", "800", ..., ou "cento".
            (cond
              ((= c 9)
               (som "900"))
              ((= c 8)
               (som "800"))
              ((= c 7)
               (som "700"))
              ((= c 6)
               (som "600"))
              ((= c 5)
               (som "500"))
              ((= c 4)
               (som "400"))
              ((= c 3)
               (som "300"))
              ((= c 2)
               (som "200"))
              ((= c 1)
               (som "100"))))))))
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Teste o procedimento dizer-centenas.

Experimente chamadas ao procedimento dizer-centenas como as indicadas

- > (dizer-centenas 987)
- > (dizer-centenas 100)
- > (dizer-centenas 56)
- > (dizer-centenas 3)



**Em algumas destas chamadas nada se ouve. Será falha do procedimento dizer-centenas ?**

Passe agora para o sub-problema dizer as dezenas do número, tendo como primeiro objectivo encontrar um algoritmo adequado.

sub-problema - dizer as dezenas do número

- separa dígitos das dezenas e das unidades
- se casos especiais de 19 a 10...
  - se o dígito das unidades for 9, 8, ... ou 0
    - assim dirá "19", "18", ... ou "10"
  - se não, temos os casos que concatenam com as unidades, 20, 30, ... ou 90
    - se o dígito das dezenas for 2, 3, ... ou 9
      - assim dirá "20", "30", ... ou "90".

```
(define dizer-dezenas
  (lambda (n)
    ; separa digitos das dezenas e das unidades
    (let ((d (remainder (quotient n 10) 10))
          (u (remainder n 10))))
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
; se casos especiais de 19 a 10...
; se o dígito das unidades for 9, 8, ..., ou 0
; assim dirá "19", "18", ... ou "10"
(if (= d 1)
  (cond
    ((= u 9) (som "19"))
    ((= u 8) (som "18"))
    ((= u 7) (som "17"))
    ((= u 6) (som "16"))
    ((= u 5) (som "15"))
    ((= u 4) (som "14"))
    ((= u 3) (som "13"))
    ((= u 2) (som "12"))
    ((= u 1) (som "11"))
    ((= u 0) (som "10"))))

; se não, temos os casos que concatenam com as unidades: 20, 30, ... ou 90
; se o dígito das dezenas for 2, 3, ... ou 9
; assim dirá "20", "30", ... ou "90".
(cond
  ((= d 2) (som "20"))
  ((= d 3) (som "30"))
  ((= d 4) (som "40"))
  ((= d 5) (som "50"))
  ((= d 6) (som "60"))
  ((= d 7) (som "70"))
  ((= d 8) (som "80"))
  ((= d 9) (som "90"))))))))
```



Teste o procedimento `dizer-dezenas`.

Experimente chamadas ao procedimento `dizer-dezenas` como as indicadas:

```
> (dizer-dezenas 987)
> (dizer-dezenas 100)
> (dizer-dezenas 16)
> (dizer-dezenas 10)
> (dizer-dezenas 3)
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



E agora tem pela frente o sub-problema dizer as unidades do número.

#### sub-problema - dizer as unidades do número

- separa o dígito das unidades
- se o dígito das unidades for 9, 8, ... ou 1  
assim dirá "9", "8", ... ou "1"

```
(define dizer-unidades
  (lambda (n)
    ; separa digito das unidades
    ... ... ... para completar
```



Complete e teste o procedimento dizer-unidades.

Alcançada a solução para os três sub-problemas do problema inicial dizer número de 3 dígitos, recorde o algoritmo anteriormente definido.

#### Problema - dizer número de 3 dígitos

- sub-problema - dizer as centenas do número
- se necessário, ligar centenas com dezenas do número (através do som "e")
- sub-problema - dizer as dezenas do número...
  - ...incluindo os casos especiais de 10 a 19
- se não for caso de 10 a 19
  - se necessário, ligar dezenas com unidades do número (através do som "e")
  - sub-problema - dizer as unidades do número



No algoritmo correspondente a dizer número de 3 dígitos, faltará apenas encontrar solução para as seguintes situações:

1- ligar centenas com dezenas do número

2- ligar dezenas com unidades do número

Estas duas situações resolvem-se através do som "e"... Por exemplo, 523 será dito:

5 - "quinientos" "e" 2 - "vinte" "e" 3 - "três"

Como desafio, é-lhe agora pedido que especifique melhor estas situações, certamente em função dos valores das centenas, dezenas e unidades de cada número.

```
(define dizer-num-3-digitos
  (lambda (n)
    ; separar os dígitos
    ; c - das centenas, d - dezenas e u - unidades
    (let ((c (quotient n 100))
          (d (remainder (quotient n 10) 10))
          (u (remainder n 10)))

      ; dizer as centenas do número
      (dizer-centenas n)
      ; se necessário, liga centenas com dezenas
      (if (and (> c 0) (> d 0))
          (som "e"))
      ; dizer as dezenas do número...
      ; incluindo os casos especiais de 10 a 19
      (dizer-dezenas n)

      ; se não for caso de 10 a 19
      ; se necessário, liga dezenas com unidades do número
      ; e diz as unidades do número
      (if (not (= d 1))
          (begin
            (if (and
                  (or (> c 0) (> d 0))
                  (> u 0))
                (som "e"))
              (dizer-unidades n))))))
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



INTRODUÇÃO
1 - O ESSENCIAL DO SCHEME
R E 1 2 3 4 5 6 7
2 - RECURSIVIDADE
3 - ABSTRACÇÃO DE DADOS
4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE
5 - ABSTRACÇÕES COM DADOS MUTÁVEIS
6 - SCHEME E OUTRAS TECNOLOGIAS
7-EXERCÍCIOS E PROJECTOS
ANEXOS



Teste o procedimento `dizer-num-3-digitos`.



Nada se ouve com

> (`dizer-num-3-digitos 0`)

Fará sentido ser assim?

Se não for antes, certamente que encontrará resposta no exercício que se segue!

### Exercício 1 - dizer números até 999999

Desenvolva um procedimento designado por `dizer-numero`, que diz números de 0 a 999999.

Comece por escrever um algoritmo, tentando identificar os sub-problemas do problema que lhe é colocado.

Pista: Certamente concluirá que dizer números de 3 dígitos é um sub-problema importante no algoritmo e para esse sub-problema tem já a solução apresentada no exemplo anterior.



Desenvolva e teste o procedimento `dizer-numero`.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

**Exemplo 2** - dizer números em ordem decrescente... uma forma de introduzir a recursividade

Dizer os números em ordem decrescente até atingir zero é uma acção muito conhecida no lançamento de naves para o espaço: dez, nove, oito, sete, ..., um, partiiida.

Um algoritmo para este problema é apresentado.

Problema - diz números em ordem decrescente de n a zero

- se n é zero
  - diz qualquer coisa que signifique fim de contagem
- se não
  - diz o número n
  - diz números em ordem decrescente de n-1 a zero



Repare que, na parte final, o algoritmo chama-se a si próprio!

Acha isto correcto?

Acha que este algoritmo terá fim?

Tente "executar manualmente" o algoritmo para n = 3 e depois dê a sua opinião.

```
(define diz-numeros-em-ordem-decrescente
  (lambda (n-partida)
    (if (zero? n-partida)
        (som "bomba!")
        (begin
          (dizer-num-3-digitos n-partida)
          (diz-numeros-em-ordem-decrescente (sub1 n-partida))))))
```



Teste o procedimento diz-numeros-em-ordem-decrescente.





Um procedimento que se chama a si próprio é um procedimento recursivo.  
E `diz-numeros-em-ordem-decrescente` é um exemplo de um procedimento recursivo.

A recursividade, pela importância que tem em programação, é um tema que vai encontrar mais à frente...

## INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

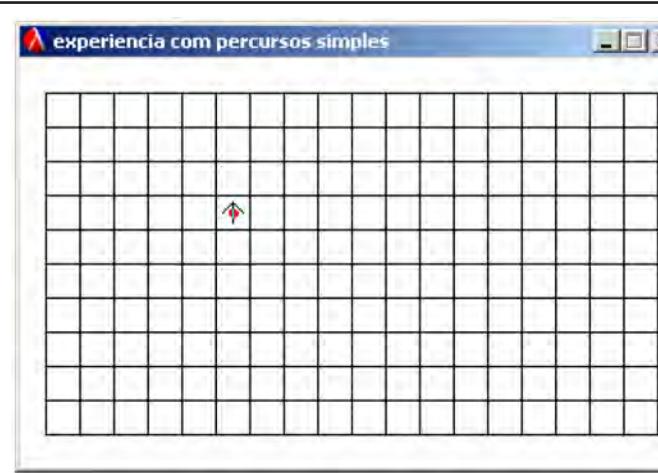
### ANEXOS



## A condução de naves

Vai agora envolver-se com a abstracção naves e treinar alguma da sua funcionalidade, usando comandos sobre uma nave.

Numa janela gráfica, com a designação `experiencia com percursos simples`, é apresentado um tabuleiro rectangular de 18 x 10 células, com uma nave na célula (5, 3) orientada para Norte (para cima).



Estando a nave na célula de coordenadas (5, 3);

- 1- identifique a célula de coordenadas (0, 0);
- 2- quais são as coordenadas da célula situada no canto inferior-direito?

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Um primeiro teste da abstracção naves.

Suponha que a nave da figura foi designada por nave-k e que agora é convidado a conduzi-la, através dos comandos:

```
> (roda-dir nave-k vezes)
    roda para a direita um ângulo equivalente a vezes×90°
> (roda-esq nave-k vezes)
    roda para a esquerda um ângulo equivalente a vezes×90°
> (frente nave-k n)
    avança nave num percurso equivalente a n células.
    o percurso é marcado com a cor associada à nave.
    se tentar passar as fronteiras do tabuleiro, avança até à fronteira
    e devolve -1
    se tiver êxito, devolve o índice de cor da última célula ocupada
    pela nave, cor existente antes de ser alterada pelo rastro e pela
    cor da nave.
```

Se quiser saber como surgiu o tabuleiro e a nave, o que se recomenda, observe o código que agora se apresenta, com algumas explicações.

Para aceder à abstracção tabuleiro.

```
(require (lib "tabuleiro.scm" "user-feup"))
```

Esta abstracção permite abrir uma janela gráfica, onde será visualizado um tabuleiro.



**Não fique com a ideia de que para usar a funcionalidade da janela gráfica precisa da abstracção tabuleiro. Aliás, por este meio apenas terá acesso a uma parte reduzida da funcionalidade associada à janela gráfica. Será pouco mais do que criar uma janela...**

**Para ter um acesso completo àquela funcionalidade, o seu programa deverá incluir**

```
(require (lib "swgr.scm" "user-feup"))
```

**mas, para já, não tem que se preocupar com este assunto.**



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

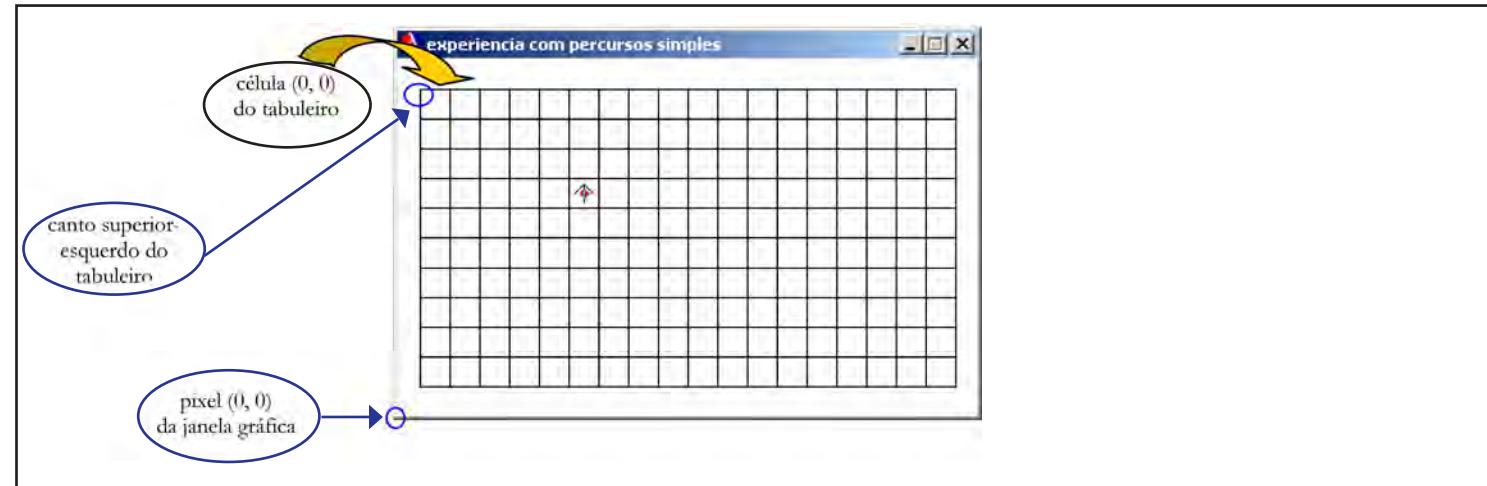
## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
; -- prepara uma janela gráfica onde será visualizado um tabuleiro -----  
(define largura-jan 390) ; largura da janela em pixels  
(define altura-jan 240) ; altura da janela em pixels  
(define titulo-jan "experiencia com percursos simples") ; titulo da janela  
;  
; cria uma janela com a designação jan-tab, usando o procedimento janela  
(define jan-tab (janela largura-jan altura-jan titulo-jan))
```

Agora, segue-se a preparação do tabuleiro.

```
; -- prepara um tabuleiro onde se deslocará a nave -----  
(define lado-cel 20) ; número de pixels do lado de cada uma das células quadradas  
(define num-cel-x 18) ; número de células em x  
(define num-cel-y 10) ; número de células em y  
(define org-x 15) ; coordenadas, da janela gráfica, do ponto correspondente  
(define org-y 220) ; ao canto superior-esquerdo do tabuleiro  
;  
; cria um tabuleiro com a designação tab-teste, usando o procedimento tabuleiro  
(define tab-teste (tabuleiro org-x org-y lado-cel num-cel-x num-cel-y))
```



Finalmente, surge a preparação de naves com as quais irá continuar o teste desta abstracção.

```
(require (lib "naves.scm" "user-feup"))  
(define cor-rasto 18) ; vermelho - cor do rastro da nave
```



# SCHEME

## na descoberta da programação



Na abstracção da janela gráfica, para além do índice, a cor também pode ser especificada através do símbolo '`temp`' ('`temp` corresponde a cor temporária).

Pintando um objecto com esta cor, ele aparece com as cores complementares das cores sobre as quais está a ser visualizado. Ou seja, se o objecto, num ponto, está a ser visualizado sobre a cor de índice 5, que é o azul cyan, cujos coeficientes RGB são 0.0, 0.5 e 1.0, a cor visualizada será aquela cujos coeficientes se obtêm trocando 0.0 por 1.0, 1.0 por 0.0 e mantendo 0.5. Ou seja, a cor obtida, de coeficientes 1.0, 0.5 e 0.0, corresponde à cor de índice 21, o vermelho amarelado.

Qual o interesse desta aparente "confusão"?

Se voltar a pintar aquele objecto, no mesmo local, com a mesma cor '`temp`', o vermelho amarelado do ponto referido transforma-se em azul cyan... e volta à cor original.

A cor '`temp`' é um truque que poderá vir a utilizar sempre que pretenda visualizar um objecto dinâmico. Bastará pintá-lo uma segunda vez, no mesmo local, para ele desaparecer e pintá-lo, a seguir, na sua nova posição...

É isto o que acontece com as naves. Quando uma nave se desloca para uma nova célula, desenrolam-se as seguintes acções:

- 1- A nave, previamente pintada com a cor '`temp`', volta a ser pintada com esta cor e desaparece, pois os pontos, debaixo da nave, retomam as cores originais.
- 2- Na nova célula, começa por ser pintado o rasto da nave e, posteriormente, é pintada a nave com a cor '`temp`'...

Tabela de cores  
(definida na abstracção janela-gráfica)

índice	R	G	B	cor
0	0.0	0.0	0.0	preto
1	0.0	0.0	0.5	azul escuro
2	0.0	0.0	1.0	azul
3	0.0	0.5	0.0	verde escuro
4	0.0	0.5	0.5	cyan escuro
5	0.0	0.5	1.0	azul cyan
6	0.0	1.0	0.0	verde
7	0.0	1.0	0.5	verde cyan
8	0.0	1.0	1.0	cyan
9	0.5	0.0	0.0	vermelho escuro
10	0.5	0.0	0.5	magenta escuro
11	0.5	0.0	1.0	azul magenta
12	0.5	0.5	0.0	amarelo escuro
13	0.5	0.5	0.5	cinzento
14	0.5	0.5	1.0	azul cinza
15	0.5	1.0	0.0	verde amaregado
16	0.5	1.0	0.5	verde cinza
17	0.5	1.0	1.0	cyan pálido
18	1.0	0.0	0.0	vermelho
19	1.0	0.0	0.5	vermelho magenta
20	1.0	0.0	1.0	magenta
21	1.0	0.5	0.0	vermelho amaregado
22	1.0	0.5	0.5	vermelho cinza
23	1.0	0.5	1.0	magenta pálido
24	1.0	1.0	0.0	amarelo
25	1.0	1.0	0.5	amarelo pálido
26	1.0	1.0	1.0	branco

```
(define x-nave 5) ; entre 0 e 17 - coordenada x inicial da nave
(define y-nave 3) ; entre 0 e 9 - coordenada y inicial da nave
(define ori-inicial 'n) ; pode ser 'n, 's, 'e, 'o - orientação inicial da nave

(define t-resposta 200) ; tempo de resposta da nave, em ms, ao rodar ou
; ao avançar para a próxima célula
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

; cria uma nave com a designação nave-k, usando o procedimento cria-nave  
(define nave-k (cria-nave tab-teste x-nave y-nave ori-inicial cor-rasto t-resposta))



Experimente controlar a nave designada por nave-k, criar outras naves, controlando-as através da funcionalidade da abstracção naves.

```
> (cria-nave tab cel-x cel-y orient c-rasto t-resposta)
    cria uma nave e devolve uma estrutura equivalente à lista
    (tab cel-x cel-y orient c-rasto t-resposta)
    em que:
        orient pode ser 'n - norte, 's - sul, 'e - este, 'o - oeste
    > (roda-dir nave vezes)
        roda para a direita um ângulo de vezes×90°
    > (roda-esq nave vezes)
        roda para a esquerda um ângulo de vezes×90°
    > (frente nave comp)
        avança nave num percurso equivalente a comp células.
        o percurso é marcado com a cor associada à nave.
        se tentar passar as fronteiras do tabuleiro, avança até à fronteira
        e devolve -1.
        se tiver êxito, devolve o índice de cor da última célula ocupada
        pela nave, cor antes de ser alterada pelo rastro e pela cor da nave.
    > (nave-tab nave)
        selector que devolve o tabuleiro em que a nave foi criada
    > (nave-pos-x nave)
        selector que devolve a coordenada x da nave, na posição actual
    > (nave-pos-y nave)
        selector que devolve a coordenada y da nave, na posição actual
    > (nave-ori nave)
        selector que devolve símbolo associado à orientação da nave
    > (nave-cor-rasto nave)
        selector que devolve a cor do rastro associada à nave
    > (nave-t-espera nave)
        selector que devolve o tempo de resposta da nave
    > (ve-frente nave)
        devolve o índice de cor da célula à frente da nave ou
        devolve -1 se tentar ver fora do tabuleiro
    > (ve-tras nave)
        devolve o índice de cor da célula atrás da nave ou
        devolve -1 se tentar ver fora do tabuleiro
    > (ve-dir nave)
        devolve o índice de cor da célula à direita da nave ou
        devolve -1 se tentar ver fora do tabuleiro
    > (ve-esq nave)
        devolve o índice de cor da célula à esquerda da nave ou
        devolve -1 se tentar ver fora do tabuleiro
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Certamente verificou a presença de sons quando as naves rodavam ou se deslocavam.

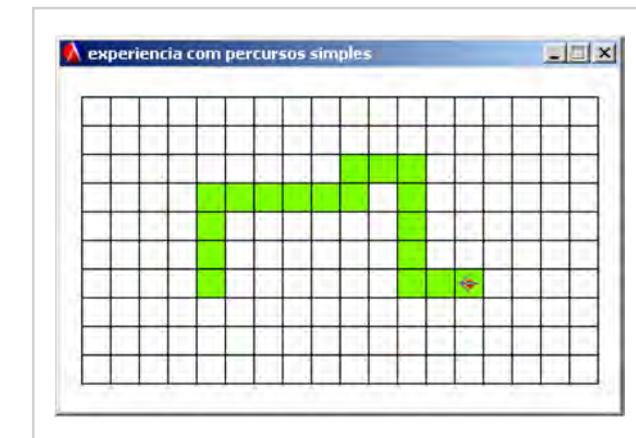
Esta funcionalidade está acessível, ao nível da abstracção naves, pela inclusão da abstracção audio, com  
`(require (lib "audio.scm" "user-feup"))`

No entanto, se pretender usar a abstracção audio no seu programa, também deverá incluir este último `require`.

### Exemplo 3 - condução de uma nave num percurso conhecido

Numa janela gráfica é apresentado um tabuleiro com um pequeno percurso pintado a verde. Numa extremidade do percurso já se encontra uma nave.

Desenvolva o procedimento `segue-o-percurso`, que faz com que a nave atinja a outra extremidade do percurso, emitindo um som de alegria quando atinge esse objectivo.



A nave colocada numa das extremidades do percurso, foi criada com

`(define n (cria-nave tab-teste 13 6 'o 18 200))`

Tente uma justificação para os vários argumentos utilizados na chamada `cria-nave`.



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

Neste caso, o problema resolve-se com um algoritmo muito simples, pois bastará conduzir a nave com comandos de frente e rodar, através de um percurso fixo, previamente conhecido.

```
; para ter acesso à abstracção audio, pois na última linha é usado som...
(require (lib "audio.scm" "user-feup"))
;
(define segue-o-percurso
  (lambda (nave)
    (frete nave 2)
    (roda-dir nave 1)
    (frete nave 4)
    (roda-esq nave 1)
    (frete nave 2)
    (roda-esq nave 1)
    (frete nave 1)
    (roda-dir nave 1)
    (frete nave 5)
    (roda-esq nave 1)
    (frete nave 3)
    (som "ri3")))
```



Teste o procedimento `segue-o-percurso`.

Uma nave foi criada com:

```
(define n (cria-nave tab-teste 13 6 'o 18 200))
```

Experimente o comando:

```
> (segue-o-percurso n)
```

Aproveite para criar outras naves, com outras características, e aplique o mesmo comando...

É agora convidado a analisar o código que visualiza, numa janela gráfica, um tabuleiro com um percurso pintado a verde e com uma nave colocada numa das extremidades desse percurso.





Numa primeira análise deste código, não se preocupe muito com a definição e visualização do percurso, operação que faz uso do procedimento `células` da abstracção tabuleiro.

No entanto, pode já observar a relação entre os argumentos que surgem nas várias chamadas do procedimento `cons` do Scheme (que ainda não conhece) com as coordenadas das células que definem o percurso...

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
; acesso à abstracção tabuleiro
(require (lib "tabuleiro.scm" "user-feup"))

; -- preparação de uma janela gráfica onde será visualizado um tabuleiro -----
(define largura-jan 390) ; largura da janela em pixels
(define altura-jan 240) ; altura da janela em pixels
(define jan-tab (janela largura-jan altura-jan "experiencia com percursos simples"))

; -- preparação de um tabuleiro onde será visualizado um percurso -----
(define lado-cel 20) ; número de pixels do lado da célula quadrada
(define num-cel-x 18) ; número de células em x
(define num-cel-y 10) ; número de células em y
(define org-x 15) ; coordenadas do canto inferior-esquerdo
(define org-y 220) ; do tabuleiro
;
(define tab-teste (tabuleiro org-x org-y lado-cel num-cel-x num-cel-y))

; ---- definição e visualização do percurso -----
(define cor-perc 15) ; verde amarelado

(celulas tab-teste
  (list (cons 13 6) (cons 12 6) (cons 11 6) (cons 11 5)
        (cons 11 4) (cons 11 3)
        (cons 11 2) (cons 10 2) (cons 9 2)
        (cons 9 3) (cons 8 3) (cons 7 3) (cons 6 3)
        (cons 5 3) (cons 4 3) (cons 4 3) (cons 4 4)
        (cons 4 5) (cons 4 6))
  'l 300 cor-perc) ; 'l- limpa com cor-perc
;
; --- fim da definição do percurso

; acesso à abstracção naves e criação de uma nave
(require (lib "naves.scm" "user-feup"))
(define n (cria-nave tab-teste 13 6 'o 18 200))
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



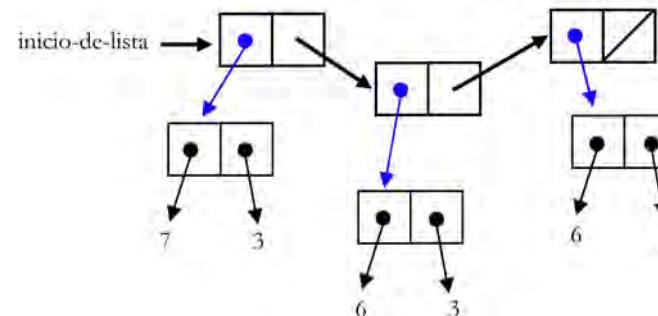
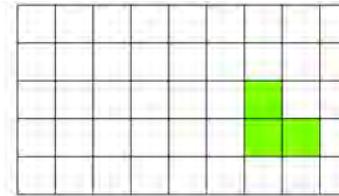
É certo que, em todo este código, apenas terá tido alguma dificuldade, aliás previsível, na parte relativa ao percurso. Mais propriamente na parte em que se definem as células que constituem o percurso.

Um percurso é definido por uma sequência de células, pintadas com uma certa cor.

A representação dessa sequência de células é feita através de uma lista (criada pelo procedimento `list` do Scheme, que ainda não conhece) de elementos.

No exemplo que se segue, o percurso de três células seria definido por uma lista de três elementos:

`(list (cons 7 3) (cons 6 3) (cons 6 2))`.



Cada um destes três elementos representa uma célula do percurso através das suas coordenadas reunidas num par (criado pelo procedimento `cons` do Scheme).

Não sendo para já muito necessário, este tema é tratado com alguma profundidade na [Parte Abstracção de Dados](#) desta obra.

Para uma consulta rápida pode utilizar o Anexo A, em Processamento de Pares e Listas.



# SCHEME

## na descoberta da programação



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



No contexto do espaço aberto, pode usar o tabuleiro com o percurso pintado a verde para experimentar a funcionalidade da abstracção tabuleiro.

Comece por identificar, no código, a criação do tabuleiro, do percurso e da nave.

Sugere-se a alteração não só da cor do percurso, mas também da sua estrutura.

Apresenta-se agora um resumo da funcionalidade da abstracção tabuleiro.

```
> (tabuleiro org-x org-y lado num-cel-x num-cel-y)
    cria um tabuleiro e devolve uma estrutura equivalente à lista
        (org-x org-y lado num-cel-x num-cel-y)
> (celula tab x y simbolo cor-actual)
    altera o aspecto de uma célula, de acordo com cor-actual, em que
        simbolo pode ser:
        simbolo   acção
        +         desenha uma cruz +
        x         desenha uma crux x
        c         desenha círculo O
        p         desenha ponto .
        n         desenha seta para norte
        s         desenha seta para sul
        e         desenha seta para este
        o         desenha seta para oeste
        l         limpa célula
> (indice-celula tab x y)
    fornece o índice de cor do ponto central da célula
> (limpa-tab tab cor-actual)
    limpa o tabuleiro com cor-actual
> (celulas tab lista simb t-espera cor-actual)
    aplica celula (indicada anteriormente) à lista de células
        (especificadas por pares de coordenadas x e y)
> (cel-x tabul)
    devolve num-cel-x
> (cel-y tabul)
    devolve num-cel-y

; limpa e janela são importados da abstracção janela gráfica
; e também disponibilizadas através da abstracção tabuleiro
> (limpa)
    limpa a janela gráfica onde se encontrará o tabuleiro
> (janela larg alt titulo)
    cria uma janela gráfica e devolve uma estrutura equivalente à
        lista (larg alt titulo)
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

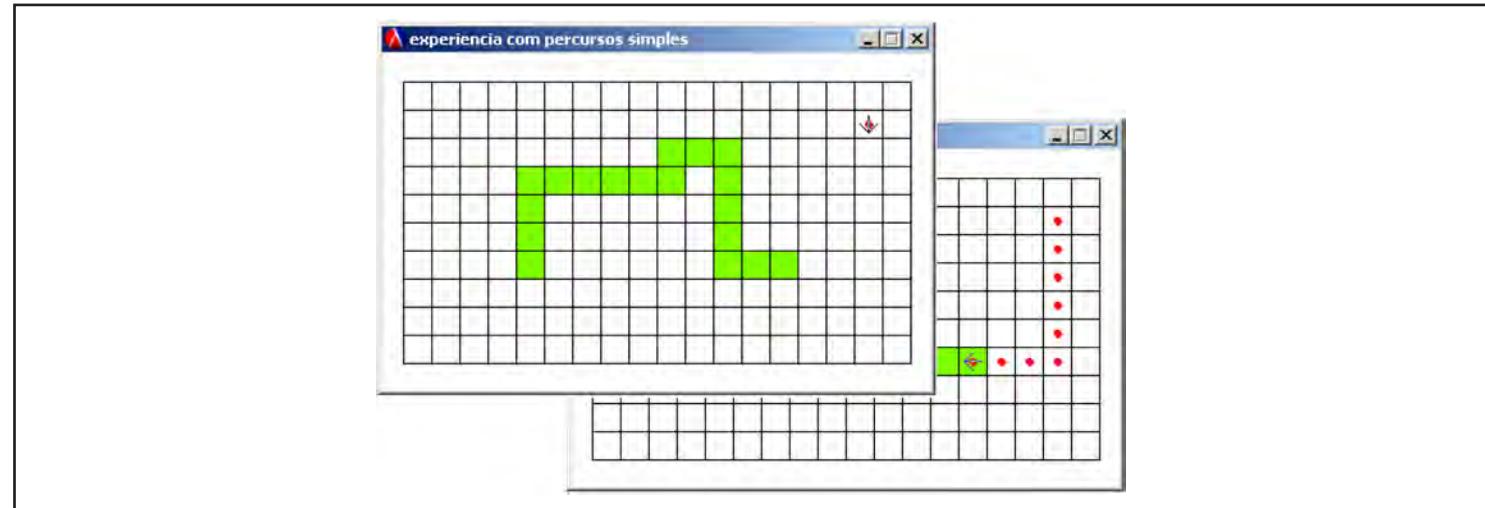
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 2 - deslocação de uma nave para outra célula

Imagine o contexto do exemplo anterior, sendo que agora não é garantido que a nave surja numa das extremidades do percurso.



Assim, pretende-se que desenvolva o procedimento `posiciona-nave` que tem como parâmetros: `nave`, `x-nova-posi` e `y-nova-posi`. A nave deverá ser deslocada para uma nova célula, cujas coordenadas são representadas pelos dois últimos parâmetros do procedimento.



Para este exercício, tem agora algumas pistas.

A figura mostra:

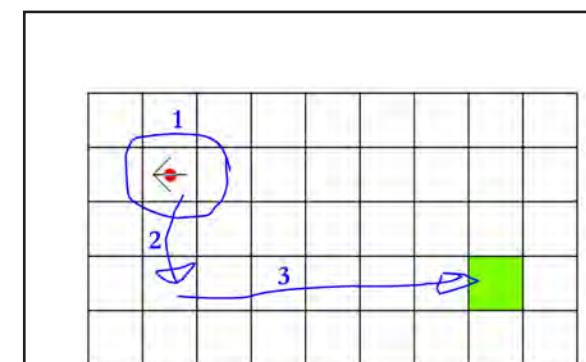
- as três operações que deverão ser programadas para levar a nave da posição actual à nova posição.

Identifique essas três operações.

A figura mostra ainda:

- uma das quatro possíveis posições relativas da nave e a nova posição, ou seja, neste caso, a nave está para cima e para a esquerda da nova posição

Identifique as restantes três posições relativas.





Desenvolva e teste o procedimento `posiciona-nave`.

A nave visualizada foi designada por `n`, está colocada na célula (16, 1) e aponta para sul.

Experimente o procedimento desenvolvido, posicionando a nave no início do percurso ou noutras células à escolha.

Crie naves noutras posições e experimente o mesmo procedimento a partir delas.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Se tiver encontrado dificuldades inultrapassáveis no exercício, sugere-se que analise uma possível solução no código que se segue:

```
(define posiciona-nave
  (lambda (nave x-nova-posicao y-nova-posicao)
    (let ((desloca-x (- x-nova-posicao (nave-pos-x nave)))
          (desloca-y (- y-nova-posicao (nave-pos-y nave))))
        (cond ((zero? desloca-y)
               ; não desloca na vertical
               )
              ((positive? desloca-y)
               (orienta-nave nave 's)
               (frente nave desloca-y))
              (else
               (orienta-nave nave 'n)
               (frente nave (abs desloca-y))))
        (cond ((zero? desloca-x)
               ; não desloca na horizontal
               )
              ((positive? desloca-x)
               (orienta-nave nave 'e)
               (frente nave desloca-x))
              (else
               (orienta-nave nave 'o)
               (frente nave (abs desloca-x)))))))

(define orienta-nave
  (lambda (nave nova-orientacao)
    (if (not (equal? (nave-ori nave) nova-orientacao))
        (begin
          (roda-dir nave 1)
          (orienta-nave nave nova-orientacao)))))
```



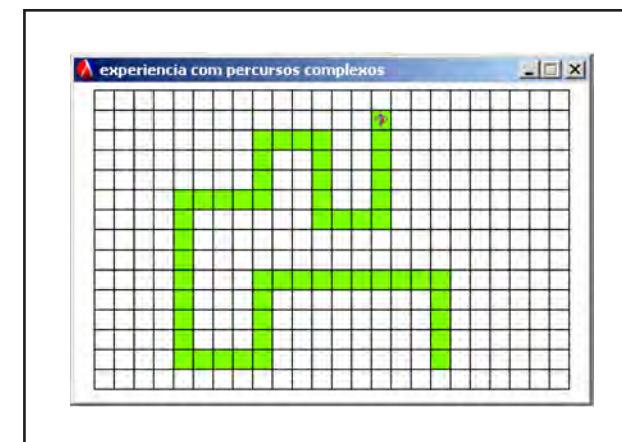
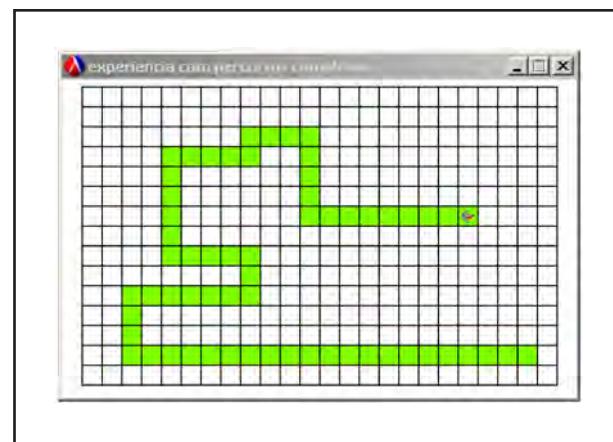


**Exemplo 4** - condução de uma nave em percursos desconhecidos

O problema que agora se coloca apresenta-se com um nível de dificuldade bastante superior à dos anteriores, que apareciam sempre com o mesmo percurso. Na situação actual, o percurso é desconhecido e a nave tem que ser programada para o atravessar. Só se sabe que a nave é colocada numa das extremidades do tal percurso, orientada ou não para avançar.

Imaginando que o percurso está pintado com uma cor fixa, a ideia é programar a nave para seguir esse percurso analisando a cor das células vizinhas, avançando para a célula que for da cor do percurso.

A figura mostra dois percursos possíveis entre tantos outros.



**Que funcionalidade das naves utilizaria para fazer a análise das células vizinhas? Como faria para detectar o fim do percurso?**

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Sabe-se que a nave está colocada no início do percurso. Então, um algoritmo possível poderá ser:

#### Problema - segue percurso desconhecido

- sub-problema - determina nova direcção da nave
- se chegou ao fim
  - expressa o seu contentamento, através de um som adequado
  - se não
    - orienta a nave de acordo com a nova direcção
    - avança a nave para a próxima célula
    - segue percurso desconhecido



**Tem aqui um algoritmo recursivo. Porquê?**

#### sub-problema - determina nova direcção da nave

- se a célula em frente é da cor cor-percurso, devolve 0
- se não,
  - se a célula da direita é da cor cor-percurso, devolve 1
  - se não,
    - se a célula de trás é da cor cor-percurso, devolve 2
    - se não,
      - se a célula da esquerda é da cor cor-percurso, devolve 3
  - se não, devolve -1



**Tente explicar a escolha dos valores devolvidos no algoritmo que determina a nova direcção da nave, sabendo que a nave pode rodar para a direita ou para a esquerda de ângulos de 90°.**





Finalmente, apresenta-se a codificação do algoritmo do sub-problema, seguida pela codificação do algoritmo do problema designado por segue percurso desconhecido.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Teste o procedimento `segue-percurso-desconhecido`.

Verifique que o procedimento a testar conduz a nave n até ao fim do percurso, com a chamada

> (`segue-percurso-desconhecido n cor-perc`)

Pode experimentar alterar a posição da nave, deslocando-a com os comandos adequados, ou então tentar alterar o percurso, fazendo modificações na lista que o define, e verificar se o procedimento continua a funcionar correctamente.

**Exercício 3** - condução de uma nave em percursos desconhecidos e com cruzamentos

A nave está colocada no início do percurso pintado a verde (cor de índice 15) e deve ser programada para atingir a célula final, pintada a cinzento (cor de índice 13).

Esta nave segue normalmente em frente, excepto quando encontra uma célula azul (cor de índice 14), sinal colocado no percurso para indicar que a nave deve rodar à direita, ou uma célula vermelha (cor de índice 22), sinal para rodar à esquerda.



Coloque-se "em cima da nave" e faça o percurso, obedecendo aos sinais indicados.  
Chega ou não à célula pintada a cinzento?



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Agora pretende-se que desenvolva o procedimento segue-percurso-com-sinais que tem nave como parâmetro único.



Desenvolva e teste o procedimento segue-percurso-com-sinais.

Verifique se o procedimento que desenvolveu conduz a nave até ao fim do percurso, sabendo que esta nave foi criada com  
`(define n ( cria-nave tab-teste 19 6 'o cor-rastro t-resposta))`

Pode também experimentar alterar o percurso, colocando, por exemplo, os sinais noutras células, e verificar se o procedimento continua a funcionar correctamente.

### Projectos com labirintos

Para esta etapa são-lhe apresentadas duas propostas de projectos, ambas já com um nível de dificuldade bastante elevado.

Aliás, em relação à segunda proposta Projecto 2, será melhor deixá-la como um desafio em aberto, para uma altura em que já se sinta com a capacidade necessária para o enfrentar... Combinado?

#### Projecto 1 - Labirinto

Num tabuleiro é gerado, aleatoriamente, um labirinto. Repare que a fronteira do tabuleiro é constituída por um muro, no qual há apenas duas aberturas devidamente identificadas por cores distintas. A entrada, pintada a amarelo, onde é colocada uma nave a apontar para dentro do labirinto, e a saída, pintada a verde.

O número de obstáculos que formam o interior do labirinto é definido percentualmente em relação à ocupação. Por exemplo, para uma ocupação de 100%, o labirinto ficaria completamente bloqueado, sem qualquer célula por onde a nave pudesse transitar.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



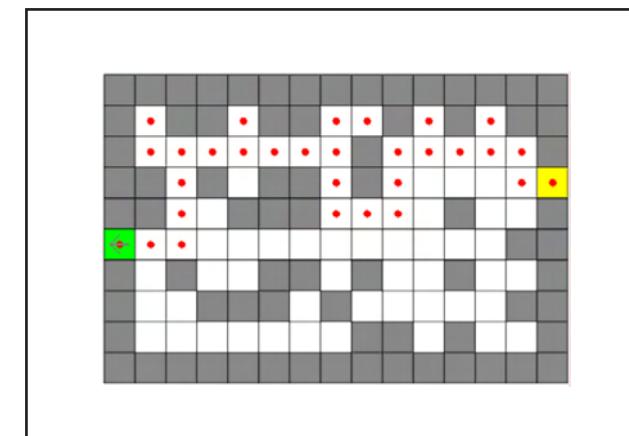
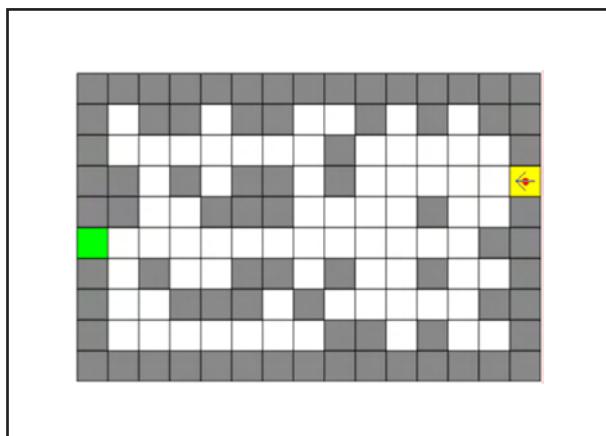
Para preencher aleatoriamente o interior do labirinto é aceitável determinar um  $x$  e um  $y$  aleatoriamente e verificar se a célula  $(x, y)$  ainda não está ocupada. Se já estiver ocupada, o melhor será gerar outro  $x$  e outro  $y$  aleatoriamente...

Reflecta um pouco e diga, em que momento, esta forma de preenchimento poderá apresentar algum problema de funcionamento. Consegue indicar outra forma de preenchimento que evite o referido problema, mesmo que não saiba, para já, como a programar em Scheme?



Também se aceita que alguns labirintos propostos não sejam muito interessantes, por não darem qualquer hipótese à nave. Por exemplo, a célula imediatamente após a entrada ou imediatamente antes da saída estar bloqueada. A nave também deverá estar preparada para este tipo de situação.

Analise com atenção as figuras que seguem.



Neste caso, a nave encontra um caminho entre a entrada e a saída.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

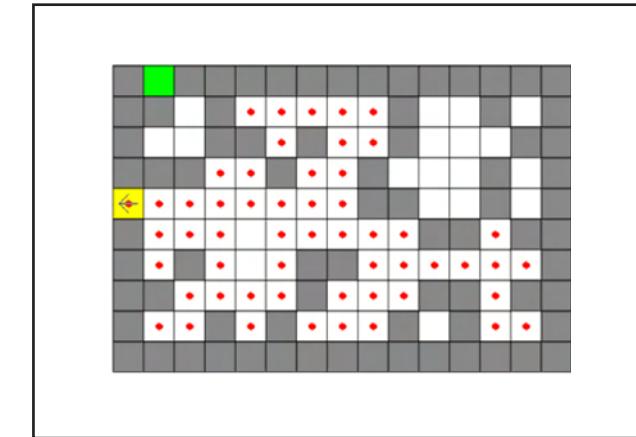
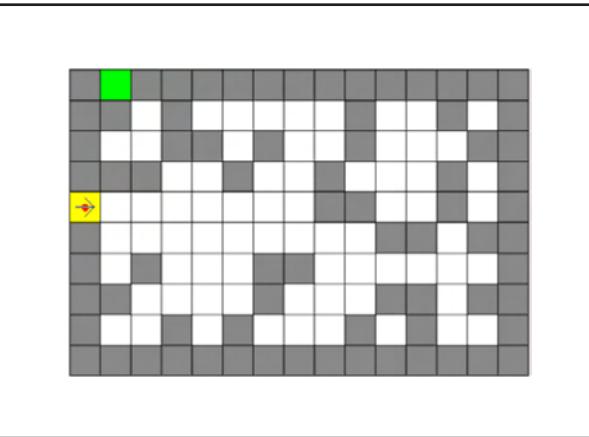
### ANEXOS



Observe o "ar" de satisfação com que a nave acaba...

```
> (nave-no-labirinto n)
Desta vez consegui atingir o objectivo
Sou o Maior... ou quase...
>
```

No labirinto que se segue o insucesso era perfeitamente previsível...



e a expressão final surge adequada.

```
> (nave-no-labirinto n)
Voltei ao início!!!!...
Não consegui sair!!!!
Bem me esforcei (só às vezes), mas sem êxito!!!
E não vale rir! Será que conseguiram melhor que eu?
>
```



Desenvolva e teste o projecto apresentado.

Analise com muita atenção o enunciado e, se detectar alguma falha na especificação, sugira uma especificação adicional.

Tente resolver o problema seguindo uma abordagem de-cima-para-baixo.

Vá testando as soluções para os sub-problemas que identificar até chegar à solução do problema proposto...

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Projecto 2 - Perseguição

Num tabuleiro são colocadas 2 naves! Ambas dão passos para a frente de comprimento 1. Uma das naves é muito mais lenta do que a outra, ou seja, enquanto a mais lenta dá um passo a outra consegue dar dois (mas sempre um de cada vez...). As duas naves são colocadas num tabuleiro, em células calculadas aleatoriamente e também com orientações aleatórias.

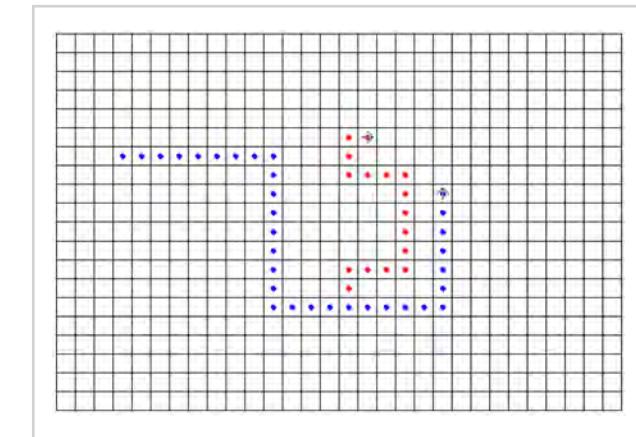
A nave lenta faz um percurso aleatório, como se disse, de passos unitários. Isto significa que a sua orientação, antes de cada passo, é recalculada...

A nave mais rápida tem por objectivo apanhar a nave lenta, tendo por base as seguintes regras:

- 1- de acordo com uma estratégia a imaginar, começa por tentar encontrar o rasto da nave lenta
- 2- encontrado o rasto, segue-o numa direcção tentando apanhar a nave lenta.
- 3- se verificar, em algum momento, que a direcção escolhida no passo anterior não foi a melhor, inverte o seguinte de perseguição...

Como acaba a perseguição?

- 1- Com a vitória da nave lenta, se ao fim de  $n$  passos ( $n$  poderá ser um parâmetro, mas um valor razoável será 100) não tiver sido apanhada. E o programa termina com a mensagem:  
Ganhou a nave lenta...
- 2- Com a vitória da nave rápida, se conseguir apanhar a nave lenta ainda em andamento. E o programa termina com a mensagem:  
Ganhou a nave rápida...



Na figura, a nave lenta apresenta o rastro vermelho e a nave rápida o rastro azul.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

R E 1 2 3 4 5 6 7

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Desenvolva e teste o Projecto apresentado.

Analise com muita atenção o enunciado apresentado e, se detectar alguma falha na especificação, sugira uma especificação adicional.

Tente resolver o problema seguindo uma abordagem de-cima-para-baixo.

Vá testando as soluções para os sub-problemas que identificar até chegar à solução do problema proposto...

Dois pontos prévios:

1-

Levanta-se aqui um pequeno problema que convém resolver de imediato. No tabuleiro onde decorre a perseguição, as células, em cada momento, poderão apresentar uma das seguintes situações:

- ainda não visitada (índice 26 - branco)
- visitada mais recentemente pela nave lenta. Tem, ao centro, a cor do rastro da nave lenta (por exemplo, índice 18 - vermelho)
- visitada mais recentemente pela nave rápida. Tem, ao centro, a cor do rastro da nave rápida (por exemplo, índice 2 - azul)
- visitada, nesse momento, por uma das naves, o que corresponde à sobreposição da cor do rastro dessa nave com a cor 'temp', a cor com que todas as naves são pintadas.

Em situações como esta, vá à tabela de cores e tome nota dos coeficientes RGB da cor do rastro. Depois, os coeficientes 0.0 troca por 1.0, e 1.0 troca por 0.0 e mantém os coeficientes 0.5. No caso do vermelho, índice 18, os coeficientes são 1.0, 0.0, 0.0. Trocando como indicado, aparece, 0.0, 1.0, 1.0 que corresponde ao índice 8. Ou seja, o índice 8 corresponde à célula onde se encontra a nave com o rastro de índice 18.

**Já agora, como exercício, qual será o índice da célula onde se encontra a nave rápida de cor correspondente ao índice 2? Será a cor de índice 24?**

**Que índice de cor não deve ser usada como rastro de nave, por não permitir distinguir entre uma célula com rastro e uma célula com rastro e nave?**

2-

Pense numa estratégia que permita à nave rápida encontrar um ponto da trajectória da nave lenta.

**E encontrado esse ponto, poderá ajudar a nave rápida a tomar a melhor decisão, ou seja, a decisão que vá na direcção da nave lenta?**

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Resumo dos módulos 2.1, 2.2 e 2.3

Resumo dos assuntos abordados nos módulos

Módulo 2.1 - **Resolver problemas, passo a passo, na direcção de um caso conhecido!**

Módulo 2.2 - **Processos e os recursos computacionais que consomem**

Módulo 2.3 - **Projecto - Jogo "O computador-adivinho"**



### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECURSIVIDADE**

**R E 1 2 3**

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

## **Resumo dos assuntos abordados nos módulos 2.1, 2.2 e 2.3**

Neste conjunto de módulos é introduzido o conceito de recursividade e posta em evidência a importância que este conceito tem na resolução de problemas de programação. A abordagem *de-cima-para-baixo (top-down)* é apresentada como uma forma sistemática de atacar os problemas, dividindo-os em sub-problemas mais simples.

### **Módulo 2.1**

A recursividade é o tema fulcral deste módulo, amplamente justificado pela sua enorme importância no âmbito da programação. Trata-se de um conceito cuja utilização nos acompanha quase sem darmos por isso em variadíssimas situações do nosso quotidiano. Os procedimentos recursivos usam a recursividade e identificam-se por se chamarem a si próprios.

### **Módulo 2.2**

É muito importante saber distinguir entre procedimento e processo. Procedimento é uma porção de texto que se pode ler, enquanto o processo existe no interior do computador, criado pela chamada do procedimento, e que consome recursos computacionais para gerar resultados. Os procedimentos recursivos podem criar processos recursivos e processos iterativos. Os primeiros caracterizam-se por deixar operações suspensas em memória, até que se verifique uma certa condição, a condição de terminação. Os processos iterativos não deixam operações suspensas em memória. Assim, é de prever que os processos recursivos sejam mais consumidores de recursos computacionais, especialmente de memória, do que os iterativos. Com a Ordem de Crescimento ou Ordem de Complexidade procura-se determinar o comportamento dos processos e os recursos computacionais que gastam face à dimensão dos problemas, sendo apresentados exemplos de ordem de crescimento  $O(1)$  - constante,  $O(n)$  - linear,  $O(M^n)$  - exponencial,  $O(\log_M n)$  - logarítmica e  $O(n^2)$  - quadrática.

### **Módulo 2.3**

O Scheme disponibiliza mecanismos para a escrita de procedimentos como blocos ou caixas-pretas, permitindo definir no interior dos procedimentos outros procedimentos, através das formas especiais `let`, `letrec` e `let*`. Esta característica dos procedimentos é especialmente importante na resolução de problemas de média ou elevada complexidade.



# **S C H E M E**

## **na descoberta da programação**

### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECURSIVIDADE**

**R E 1 2 3**

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

A segunda parte do módulo corresponde ao desenvolvimento de um projecto, que aqui surge para introduzir a abordagem de-cima-para-baixo (*top-down*) na programação de problemas com alguma complexidade. Esta abordagem passa por várias fases: especificação do problema, procura de uma ideia de solução, definição do algoritmo, desdobramento do problema em sub-problemas e, finalmente, codificação e teste da solução encontrada para os vários sub-problemas e para o problema na sua globalidade.



## **INTRODUÇÃO**

## **1 - O ESSENCIAL DO SCHEME**

## **2 - RECURSIVIDADE**

R E 1 2 3

## **3 - ABSTRACÇÃO DE DADOS**

## **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

## **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

## **6 - SCHEME E OUTRAS TECNOLOGIAS**

## **7-EXERCÍCIOS E PROJECTOS**

## **ANEXOS**

## **Exercícios e exemplos**

São aqui apresentados alguns exercícios e exemplos para consolidação da matéria abordada nos Módulos 2.1, 2.2 e 2.3.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME



2 - RECUSIVIDADE

R E 1 2 3

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



### Exemplo 1

O procedimento `impar-cima-par-baixo` espera um argumento inteiro positivo. Se esse inteiro for par, divide-o por 2. Se for ímpar, multiplica-o por 3 e depois soma 1. O resultado assim obtido é sujeito a um tratamento idêntico ao indicado e só pára quando o resultado for 1.

```
> (impar-cima-par-baixo 6)
6 3 10 5 16 8 4 2 1 OK
```

**Desenvolva manualmente mais algumas situações do problema. Identifique a ideia de solução indicada no código que se segue.**  
**Apresente o algoritmo respectivo.**

```
(define impar-cima-par-baixo
  (lambda (num)
    (display num)
    (display " ")
    (cond ((= num 1)
           (display "OK")
           (newline))
          ((even? num)
           (impar-cima-par-baixo (quotient num 2)))
          (else
           (impar-cima-par-baixo (add1 (* num 3)))))))
```



Teste o procedimento `impar-cima-par-baixo`.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECUSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exemplo 2

O procedimento imprime-bin espera um argumento inteiro positivo, na base 10. Este procedimento imprime o inteiro dado, convertido para a base 2.

```
> (imprime-bin 76)
1001100
> (imprime-bin 6)
110
```



**Desenvolva manualmente mais algumas situações do problema.**

A visualização do inteiro na base 2 começa pelo dígito mais significativo (no sentido da esquerda para a direita), mas sabemos que é muito mais fácil começar pelo dígito menos significativo. Por exemplo, para determinar o dígito menos significativo de um inteiro na base 2, bastará determinar o resto da divisão inteira desse número por 2. E para retirar o dígito menos significativo ao inteiro, bastará calcular o quociente daquela divisão. Uma solução recursiva, que deixe suspensa a operação de visualização, resolve facilmente o facto de pretendermos começar a visualização pelo dígito mais significativo quando o menos significativo está "muito mais à mão".



**Identifique a ideia de solução indicada no código que se segue.  
Apresente o algoritmo respectivo.**

```
(define imprime-bin
  (lambda (num)
    (cond ((< num 2) (display num))
          (else
            (imprime-bin (quotient num 2))
            (display (remainder num 2))))))
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECUSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

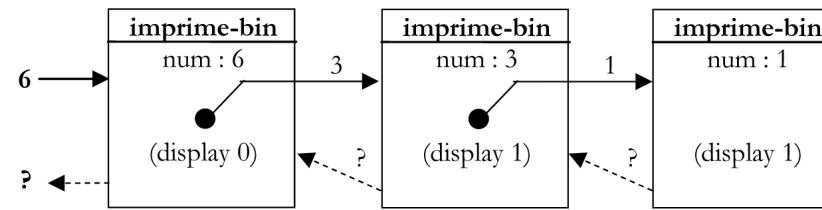
## ANEXOS

Os processos gerados não são iterativos, mas sim recursivos, com  $O(n)$  em tempo e espaço, conclusão que também se poderá retirar da representação gráfica relativa à chamada (`imprime-bin 6`).



**É dito que os processos gerados são recursivos. Justifique.**

Verifique, pela representação gráfica, que o valor devolvido pelos processos não é importante, ao contrário do que acontece com o efeito colateral correspondente à visualização no ecrã.



Teste o procedimento `imprime-bin`.

Experimente na cláusula `else` trocar a posição da chamada recursiva `imprime-bin` com `display` e explique o que acontece.



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

R E 1 2 3

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

### Variante

O procedimento imprime-na-base tem 2 parâmetros, num e base, para os quais espera dois argumentos inteiros positivos, ambos na base 10. Este procedimento imprime num na base especificada por base.

```
> (imprime-na-base 76 2)  
1001100  
> (imprime-na-base 76 3)  
2211  
> (imprime-na-base 76 10)  
76
```



Desenvolva manualmente mais algumas situações do problema.  
Identifique a ideia de solução indicada no código que se segue.  
Apresente o algoritmo respectivo.

```
(define imprime-na-base  
  (lambda (num base)  
    (cond ((< num base) (display num))  
          (else  
            (imprime-na-base (quotient num base) base)  
            (display (remainder num base))))))
```



Teste o procedimento imprime-na-base.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

### Exemplo 3

O procedimento caminho-errante espera, como argumento, um valor inteiro que é considerado o ponto de partida e gera, aleatoriamente, 1, 2 ou 3. Quando o número gerado é 1, subtrai 1 ao ponto de partida, quando é 2, soma-lhe 1, e quando é 3, nada altera. O ponto de partida é assim actualizado e o procedimento chama-se a si próprio, pois não tem fim.

```
> (caminho-errante 20)
20 19 19 20 21 22 22 21 20 21 22 22 23 24 23 22 23
23 24 25 26 25 26 26 26 26 25 25 25 26
user break
```



Tente desenvolver manualmente mais algumas situações do problema.

Que tipo de problema encontra?

Identifique a ideia de solução indicada no código que se segue.

Apresente o algoritmo respectivo.

Notar que neste procedimento recursivo não existe caso base, pois não tem fim.

```
(define caminho-errante
  (lambda (origem)
    (display origem)
    (display " ")
    (let ((numero-aleat (aleatorio-de-1-ate-limite 3)))
      (cond ((= numero-aleat 1)
              (caminho-errante (sub1 origem)))
            ((= numero-aleat 2)
              (caminho-errante (add1 origem)))
            (else
              (caminho-errante origem))))))

(define aleatorio-de-1-ate-limite      ; gera, aleatoriamente, um valor
  (lambda (limite)                      ; entre 1 e limite
    (add1 (random limite))))
```



<b>INTRODUÇÃO</b>
<b>1 - O ESSENCIAL DO SCHEME</b>
<b>2 - RECURSIVIDADE</b>
R E 1 2 3
<b>3 - ABSTRACÇÃO DE DADOS</b>
<b>4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE</b>
<b>5 - ABSTRACÇÕES COM DADOS MUTÁVEIS</b>
<b>6 - SCHEME E OUTRAS TECNOLOGIAS</b>
<b>7-EXERCÍCIOS E PROJECTOS</b>
<b>ANEXOS</b>



Teste o procedimento caminho-errante.

## Exercício 1

Em relação ao exemplo anterior (procedimento caminho-errante), apresente uma solução que prevê enganos do utilizador, como quando este indica um argumento que não obedece às condições enunciadas.

A solução deverá ser um procedimento com a designação caminho-errante-melhorado, apresentado como uma caixa-preta, logo com todos os seus procedimentos auxiliares definidos localmente e não definidos no ambiente global do Scheme.



Desenvolva e teste o procedimento caminho-errante-melhorado.

## Exercício 2

Escreva em Scheme o procedimento caminho-errante-2d que espera, como argumentos, dois valores inteiros, considerados as coordenadas  $x$  e  $y$  de um ponto 2D, tomado como local de partida. O procedimento gera, aleatoriamente, 1, 2 ou 3. Quando o número gerado é 1, subtrai 1 à coordenada  $x$ , quando é 2, soma 1 a essa mesma coordenada, e quando é 3, nada altera. Depois gera mais um inteiro aleatório entre 1 e 3 e faz a mesma coisa para a coordenada  $y$ . Este procedimento não tem fim.

```
> (caminho-errante-2d 20 20)
20:20 21:21 22:20 21:21 21:21 21:20 21:20 22:20 22:19
22:18 21:18 21:19 20:19 21:20 22:20 22:19 21:19

user break
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Desenvolva e teste o procedimento caminho-errante-2d.

### Exercício 3

Observe o comportamento do procedimento imprime-na-base apresentado num dos exemplos anteriores, quando o argumento correspondente à base é superior a 10.

```
> (imprime-na-base 76 11)
610                                     <---- deveria ser 6a
> (imprime-na-base 76 16)
412                                     <---- deveria ser 4c
```



Se achar necessário, teste o procedimento imprime-na-base.



**Explique os resultados obtidos com os argumentos 76 e 11 e com os argumentos 76 e 16.**

Agora desenvolva um procedimento designado por imprime-ate-base-20, que responda correctamente até à base 20, ou seja, que para além dos dígitos decimais (0 a 9) ainda utilize as letras a, b, c, d, e, f, g, h, i e j.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (imprime-ate-base-20 76 11)  
6a  
> (imprime-ate-base-20 76 16)  
4c  
> (imprime-ate-base-20 76 20)  
3g
```



Desenvolva e teste o procedimento `imprime-ate-base-20`.

### Exercício 4

Escreva em Scheme um procedimento para calcular a função de *Ackermann*, sabendo que esta é definida para inteiros não negativos, recorrendo à recursividade.

$$A(m, n) = \begin{cases} n + 1 & \text{se } m = 0 \\ A(m - 1, 1) & \text{se } m > 0 \text{ e } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{se } m > 0 \text{ e } n > 0 \end{cases}$$



Desenvolva e teste o procedimento pedido.

Experimente aumentar o valor dos argumentos, primeiro segundo `m` mantendo `n` constante e depois fazendo o contrário. Tente interpretar o comportamento do procedimento especialmente em termos do tempo de resposta.



**Identifique e justifique a Ordem de Crescimento da solução encontrada, em relação ao espaço e ao tempo.**



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 5

Escreva em Scheme um procedimento para calcular o n-ésimo número na sequência de Mewman-Conway definida por:

$$P(n) = \begin{cases} 1 & \text{se } n = 1 \text{ ou } n = 2 \\ P(P(n-1)) + P(n - P(n-1)) & \text{se } n > 2 \end{cases}$$



Desenvolva e teste o procedimento pedido.

Experimente aumentar o valor do argumento e tente interpretar o comportamento do procedimento, especialmente em termos do tempo de resposta.



**Identifique e justifique a Ordem de Crescimento da solução encontrada em relação ao espaço e ao tempo.**

### Exercício 6 - este não é nada fácil...

Pretende-se determinar o número de hipóteses de trocar n cêntimos do euro, assumindo apenas a existência de moedas de 1, 2, 5, 10, 20 e 50 cêntimos, 1 e 2 euros.

Por exemplo,

1 cêntimo - 1 hipótese: 1 moeda de 1

2 cêntimos - 2 hipóteses: 1 moeda de 2, 2 moedas de 1

3 cêntimos - 2 hipóteses: 3 moedas de 1, 1 moeda de 1 + 1 moeda de 2

4 cêntimos - 3 hipóteses: 4 moedas de 1, 2 moedas de 1 + 1 moeda de 2, 2 moedas de 2

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECUSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Determine manualmente o número de hipóteses de troca para outras situações, nomeadamente, 25, 75 e 500 cêntimos.



Apresente uma definição recursiva para determinar o número de hipóteses de trocar n cêntimos do euro, assumindo apenas a existência das moedas de 1, 2, 5, 10, 20 e 50 cêntimos, 1 e 2 euros.

Tendo por base a definição recursiva desenvolvida, escreva em Scheme o procedimento conta-trocas com um único parâmetro, quantia, que representa uma quantia em cêntimos e devolve o número de trocas possíveis dessa quantia com as moedas indicadas.

```
> (conta-trocas 1)
1
> (conta-trocas 2)
2
> (conta-trocas 3)
2
> (conta-trocas 4)
3
> (conta-trocas 0)
0
>
```

Segue-se uma hipótese de procedimento auxiliar que poderá ser importante para este problema:

```
(define outra-moeda
  (lambda (pos)
    (cond ((= pos 1) 200) ; moeda de 2 euros em centimos
          ((= pos 2) 100) ; moeda de 1 euro em centimos
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
((= pos 3) 50)      ; moeda de 50 centimos
((= pos 4) 20)      ; moeda de 20 centimos
((= pos 5) 10)      ; moeda de 10 centimos
((= pos 6) 5)       ; moeda de 5 centimos
((= pos 7) 2)       ; moeda de 2 centimos
((= pos 8) 1))))   ; moeda de 1 centimo
```



Desenvolva e teste o procedimento `conta-trocas`.

Experimente aumentar o valor do argumento e tente interpretar o comportamento do procedimento, especialmente em termos do tempo de resposta.



**Identifique e justifique a Ordem de Crescimento da solução encontrada em relação ao espaço e ao tempo.**

### Exercício 7

O procedimento `somas-iguais` tem dois parâmetros, `n` e `soma`, ambos inteiros positivos. A chamada `(somas-iguais 9 40)` determina e devolve o número de hipóteses distintas de adicionar números de 1 a 9, sem repetições, sendo 40 a soma deles.

> `(somas-iguais 9 40)`  
3

$$\begin{aligned} 40 &= 1+4+5+6+7+8+9 \\ 40 &= 2+3+5+6+7+8+9 \\ 40 &= 1+2+3+4+6+7+8+9 \end{aligned}$$

- Apresente uma definição recursiva para determinar o número de hipóteses distintas, de adicionar números de 1 a `n`, sem repetições, perfazendo `soma`.
- Escreva em Scheme o procedimento `somas-iguais` com base na definição apresentada.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Desenvolva e teste o procedimento `somas-iguais`.

Experimente aumentar o argumento e tente interpretar o comportamento do procedimento, especialmente em termos do tempo de resposta.



Indique se a recursividade utilizada é linear ou em árvore. Justifique.



Identifique e justifique a Ordem de Crescimento da solução encontrada em relação ao espaço e ao tempo.

### Exercício 8

Escrever em Scheme o programa `numero-de-somas-iguais` com três parâmetros, `n`, `lim-inf` e `lim-sup`, que se baseia no procedimento do exercício anterior, `somas-iguais`. Este programa responde do seguinte modo:

```
> (numero-de-somas-iguais 9 20 24)
Numeros de 1 a 9
20: 22
21: 23
22: 23
23: 23
24: 23
```

em que 22, 23, 23, 23, e 23 representam, respectivamente, o número de somas iguais a 20, 21, 22, 23, e 24, conseguidas com os números de 1 a 9.





Desenvolva e teste o procedimento numero-de-somas-iguais.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



### Exercício 9

Escreva em Scheme um programa designado por teste-da-tabuada, com um só parâmetro, n, que põe n questões sobre a tabuada de multiplicar (tabuadas de 1 a 10). Os números que surgem nas questões são gerados aleatoriamente.

> (teste-da-tabuada 10)

3 x 8 = 24

Resposta certa

Na primeira questão, o programa visualiza 3 x 8 e o utilizador escreveu 24. Portanto, resposta certa.

2 x 7 = 15

Resposta errada

...

No final das questões, se o número de erros for inferior a 2, a mensagem será:

Muito bem

Caso contrário será:

Deve estudar melhor a tabuada



Desenvolva e teste o procedimento teste-da-tabuada.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Exercício 10

Projecto - Um jogo de dados

Desenvolva um programa em Scheme que simule um jogo de dados, cujas regras são apresentadas em seguida. Sugere-se que se utilize uma abordagem do tipo de-cima-para-baixo (*top-down*).

O jogador lança 2 dados e os números das duas faces são adicionados, originando uma soma que se situa entre 2 e 12.

Se a soma for 7 ou 11, o jogador ganha uma quantia equivalente à soma.

Se a soma for 2, 3, ou 12, o jogador perde essa mesma quantia.

Mas se a soma for 4, 5, 6, 8, 9, ou 10, então a regra complica-se um pouco. Nestes casos, a soma adquire a designação de ponto e a jogada continua, ou seja, o jogador deverá lançar novamente os dois dados tantas vezes quantas as necessárias até conseguir obter uma soma igual a:

- ponto e ganha uma quantia equivalente a ponto;
- 7 e perde uma quantia equivalente a ponto.

Apresenta-se agora uma sessão com o programa descrito.

```
> (jogo-de-dados)

jogar (lançar dados: 1; terminar: 0): 1
dados: 6 + 5 = 11
O jogador ganhou 11 e o saldo é': 11

jogar (lançar dados: 1; terminar: 0): 1
dados: 6 + 6 = 12
O jogador perdeu 12 e o saldo é': -1

jogar (lançar dados: 1; terminar: 0): 1
dados: 4 + 6 = 10
O ponto é' 10 e continua a jogada.
jogar (lançar dados: 1): 1
dados: 2 + 4 = 6
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
O ponto e' 10 e continua a jogada.  
jogar (lançar dados: 1): 1  
dados: 6 + 5 = 11  
O ponto e' 10 e continua a jogada.  
jogar (lançar dados: 1): 1  
dados: 6 + 4 = 10  
O jogador ganhou 10 e o saldo e': 9
```

```
jogar (lançar dados: 1; terminar: 0): 1  
dados: 1 + 3 = 4
```

O ponto e' 4 e continua a jogada.

```
jogar (lançar dados: 1): 1
```

```
dados: 1 + 4 = 5
```

O ponto e' 4 e continua a jogada.

```
jogar (lançar dados: 1): 1
```

```
dados: 5 + 4 = 9
```

O ponto e' 4 e continua a jogada.

```
jogar (lançar dados: 1): 1
```

```
dados: 5 + 2 = 7
```

O jogador perdeu 4 e o saldo e': 5

```
jogar (lançar dados: 1; terminar: 0): 0
```

O jogo terminou e o jogador ficou com o saldo: 5



Desenvolva e teste o procedimento `jogo-de-dados`.



## **INTRODUÇÃO**

## **1 - O ESSENCIAL DO SCHEME**

## **2 - RECURSIVIDADE**

R E 1 2 3

## **3 - ABSTRACÇÃO DE DADOS**

## **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

## **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

## **6 - SCHEME E OUTRAS TECNOLOGIAS**

## **7-EXERCÍCIOS E PROJECTOS**

## **ANEXOS**

## **Módulo 2.1 - Resolver problemas, passo a passo, na direcção de um caso conhecido!**

Na definição de um procedimento podem surgir chamadas a outros procedimentos, situação perfeitamente normal e fácil de entender. No entanto, já não parece tão natural que um procedimento se chame a si próprio! Através de um raciocínio mais ligeiro, até poderia parecer que este procedimento nunca mais terminaria... Falamos de recursividade, a qual está na base da definição dos procedimentos recursivos, isto é, procedimentos que se chamam a si próprios e que nos irão surpreender pelas soluções simples e elegantes que proporcionam.

O objectivo deste módulo é mostrar como se definem procedimentos recursivos e como este tipo de procedimentos resolve problemas, caminhando passo a passo na direcção de um caso conhecido!

É recorrendo à recursividade que o Scheme implementa tarefas repetitivas, pois não possui as estruturas de repetição, como acontece noutras linguagens de programação (tais como `for` ou `while`).

Atenção, estamos perante um tema muito importante e que é necessário dominar na perfeição.

### **Palavras-Chave**

Recursividade, caso base ou condição de terminação, operação de redução, caso geral ou passo recursivo.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

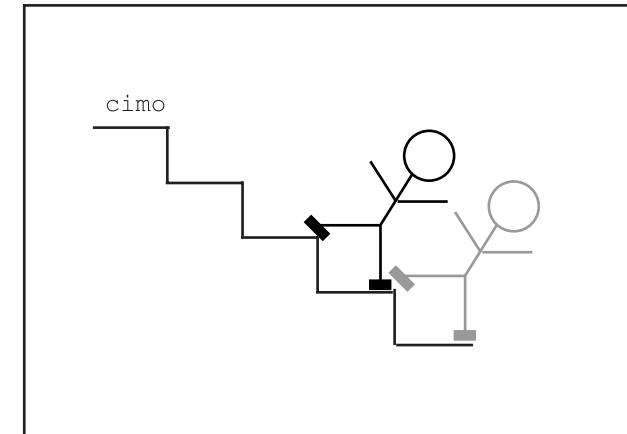
## Passo a passo, na direcção de um caso conhecido

O desafio que se coloca, para já, é a análise de situações do nosso quotidiano, resolvidas tão naturalmente que quase não damos por elas.

Se pretender subir umas escadas até ao cimo, procederá quase sem pensar, da seguinte maneira:

Tarefa subir as escadas

- Se atingiu o cimo das escadas, terminou a tarefa **subir as escadas**
- Pelo contrário, se ainda não atingiu o cimo
  - avança um degrau na direcção do cimo das escadas e
  - retoma a tarefa **subir as escadas**



Nota, certamente, algo de estranho nesta forma de actuar, pois no interior da tarefa **subir as escadas**, esta mesma tarefa chama-se a si própria.

Acabou por identificar um exemplo de uma tarefa recursiva ou, dito de outra maneira, está perante uma forma de actuar que usa a recursividade.



Como pode garantir que a tarefa **subir as escadas** tem fim?

Mesmo que tenha que dispensar algum tempo, procure dois factores que, em conjunto, dão esta garantia.

A garantia de que esta tarefa recursiva tem fim resulta da conjugação de dois factores:

- Começa sempre por testar a condição de terminação (já atingiu o cimo das escadas?) e, em caso afirmativo, a tarefa termina.
- Antes de cada nova chamada da tarefa recursiva, a dimensão do problema é reduzida (avançar um degrau na direcção do cimo das escadas) e assim fica mais próximo do fim.



Numa tarefa recursiva há sempre três aspectos que a caracterizam e que deve identificar de uma forma muito clara:

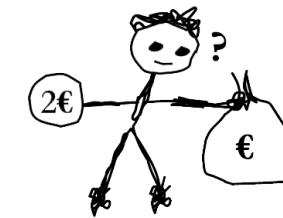
- Caso base ou condição de terminação - já atingiu o cimo das escadas?
  - Operação de redução da dimensão do problema - avança um degrau na direcção do cimo das escadas
  - Caso geral ou passo recursivo - depois de ter subido um degrau, retoma a tarefa subir as escadas.

Tome atenção ao exemplo que se segue, pois vai ter que identificar os três aspectos que caracterizam a respectiva tarefa recursiva.

Suponha que tem um saco com moedas e que pretende determinar a quantia em dinheiro nele contida.

Tarefa determinar a quantia contida no saco

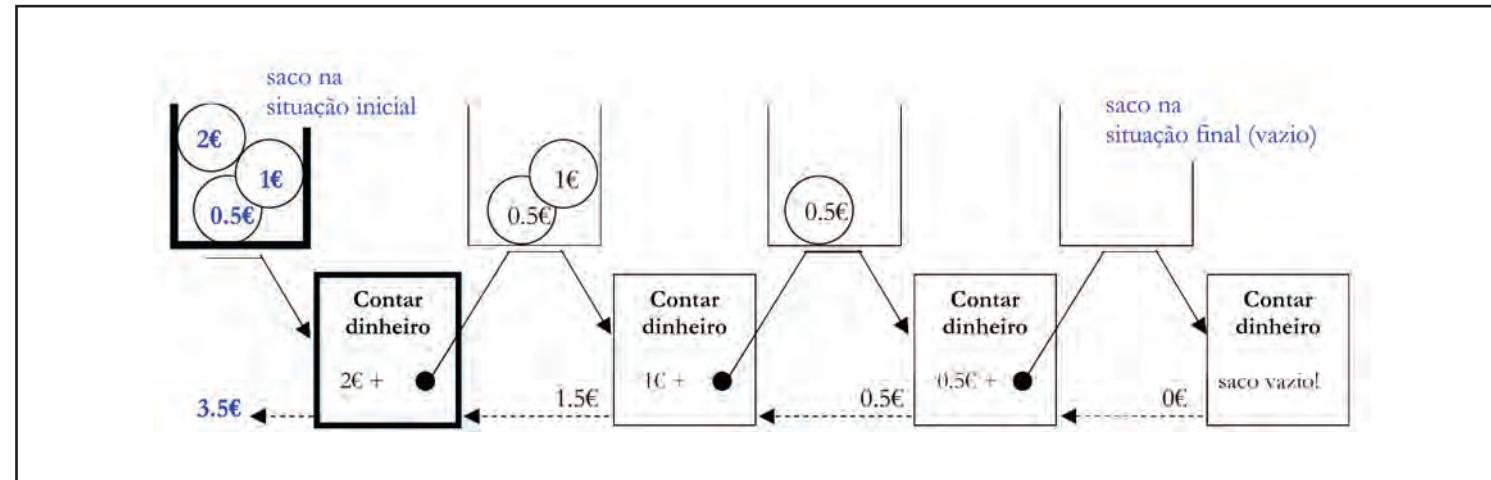
- Se o saco estiver vazio, sem moedas, isso corresponde à quantia 0€
  - Pelo contrário, se o saco contiver moedas
    - retira uma moeda do saco
    - adiciona o valor da moeda retirada ao valor da quantia em dinheiro que ainda continua no saco. O que requer retomar a tarefa determinar a quantia contida no saco



Também aqui, ao retomar a tarefa determinar a quantia contida no saco, a dimensão da tarefa surge mais reduzida. Justifique.

Neste exemplo,

- Caso base ou condição de terminação - saco vazio?, a que corresponde a quantia de 0€
  - Operação de redução da dimensão do problema - retira uma moeda do saco
  - Caso geral ou passo recursivo - adiciona o valor da moeda retirada ao valor da quantia que ainda resta no saco



Observe na figura que a contagem do dinheiro que se encontra no saco começa com o lançamento do processo **Contar dinheiro**, representado pelo rectângulo mais à esquerda.

Este processo de contagem identifica o valor da moeda retirada, mas necessita também de conhecer o valor corresponde às moedas que ainda se encontram no saco. Por este facto, e reconhecendo que esta tarefa é idêntica à que procura resolver, lança um novo processo de contagem, tendo agora o saco menos uma moeda. Isto é representado pelo segundo rectângulo a contar da esquerda.

Um aspecto muito importante é o seguinte: o processo correspondente ao rectângulo mais à esquerda vai ficar suspenso até que lhe devolvam a informação pedida, ou seja, o valor corresponde às moedas ainda no saco.

O segundo processo, pelas razões expostas para o primeiro, vai também lançar um novo processo e ficar suspenso, à espera que lhe devolvam a informação pedida.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



E esta sequência de lançamento de novos processos continua até ao processo que detecta que o saco está vazio. Este é o único que conhece imediatamente a quantia que se encontra no saco, 0€, e por isso devolve a informação respectiva ao processo que o lançou. Este processo, representado pelo rectângulo mais à direita, não fica suspenso, pois resolveu completamente a tarefa que lhe foi solicitada. Por isso morre definitivamente e liberta os recursos computacionais de memória que utilizava.

Entretanto, o processo suspenso que recebe a informação de 0€ pode agora completar a sua tarefa devolvendo a quantia de  $0\text{€} + 0.5\text{€} = 0.5\text{€}$ . Assiste-se assim ao finalizar dos processos suspensos. Quando o processo mais à esquerda recebe a informação correspondente à quantia de 1.5€, já todos os outros processos morreram. E também ele morrerá, logo após ter devolvido a quantia de  $1.5\text{€} + 2\text{€} = 3.5\text{€}$ , o valor da totalidade das moedas contidas no saco.



Como pode observar, um processo lançado vai ficar suspenso à espera de uma resposta que lhe será dada por um outro processo por ele lançado, e isto repetir-se-á até que se encontre um caso conhecido.

Imagine que o saco era enorme e que tinha mil moedas.

Que recursos computacionais se poderiam esgotar? Justifique.

### Exercício 1

Imagine duas situações da vida real que possam ser resolvidas recorrendo à recursividade, e identifique, para cada uma delas, o caso base ou condição de terminação, a operação de redução e o caso geral ou passo recursivo.

### Exprimir a recursividade em Scheme

O factorial de um inteiro positivo  $n$  é o exemplo clássico utilizado para ilustrar a recursividade. É representado por  $n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$ .

Vejamos o valor de factorial para várias situações de  $n$ .

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
0! = 1 (por convenção)
1! = 1
2! = 2 * 1
3! = 3 * 2 * 1
4! = 4 * 3 * 2 * 1
...
n! = n * (n-1) * (n-2) * ... * 1      = n * (n - 1)!
```

Já pode daqui retirar as características desta tarefa recursiva:

Caso base ou condição de terminação

Se  $n$  igual a 0                            $0! = 1$  (definido por convenção)

Caso geral ou passo recursivo

Se  $n$  diferente de 0                            $n! = n * (n-1)!$

Operação de redução

$n-1$  (subtrai 1 a  $n$ )

Notar que calcular  $(n-1)!$ , relativamente ao caso que pretende resolver, que é  $n!$ , representa uma redução da dimensão do problema, pois  $(n-1)!$  está mais perto do caso base do que  $n!$ .

A partir da definição de factorial já pode escrever o respectivo procedimento em Scheme.

```
; Procedimento recursivo com um parâmetro, num.
; Devolve o factorial de num.
;
(define factorial
  (lambda (num)
    (if (zero? num)
        1
        (* num (factorial (sub1 num)))))) ; o caso base
                                              ; o caso geral

> (factorial 0)
1
> (factorial 4)
24
```



Experimente o procedimento recursivo factorial.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECUSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Por que razão, na sua opinião, na primeira linha dos comentários associados ao procedimento `factorial` é dito que o procedimento é recursivo?

Para melhor entender como funciona este procedimento recursivo, siga, passo a passo, o desenrolar da chamada (`factorial 4`) a que corresponde a resposta 24.

Seguindo a expressão `if` do procedimento `factorial` e uma vez que `num = 4`, portanto, diferente de zero

```
(factorial 4) = (* 4 (factorial (sub1 4)))
              = (* 4 (factorial 3))
```

Um dos operandos da expressão obtida é (`factorial 3`), cujo cálculo implica nova chamada ao procedimento `factorial`, mas agora com `num = 3`, também diferente de zero. Seguindo novamente a expressão `if`:

```
(factorial 4) = (* 4 (factorial 3))
              = (* 4 (* 3 (factorial (sub1 3))))
              = (* 4 (* 3 (factorial 2)))
```

O comprimento da expressão vai aumentando, mas, em contrapartida, as chamadas ao procedimento `factorial` vão-se aproximando do caso base, o único de que se conhece a solução.

Continuando a seguir a chamada a `factorial` com `num = 2`, depois com `num = 1` e, finalmente, com `num = 0`:

```
(factorial 4) = (* 4 (* 3 (* 2 (* 1 (factorial 0)))))
```



Observe que as operações de multiplicação vão ficando suspensas.  
Poderá daqui concluir que um procedimento recursivo impõe sempre uma reserva de recursos computacionais de memória, que poderão ser enormes para dimensões grandes do problema que resolve?



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

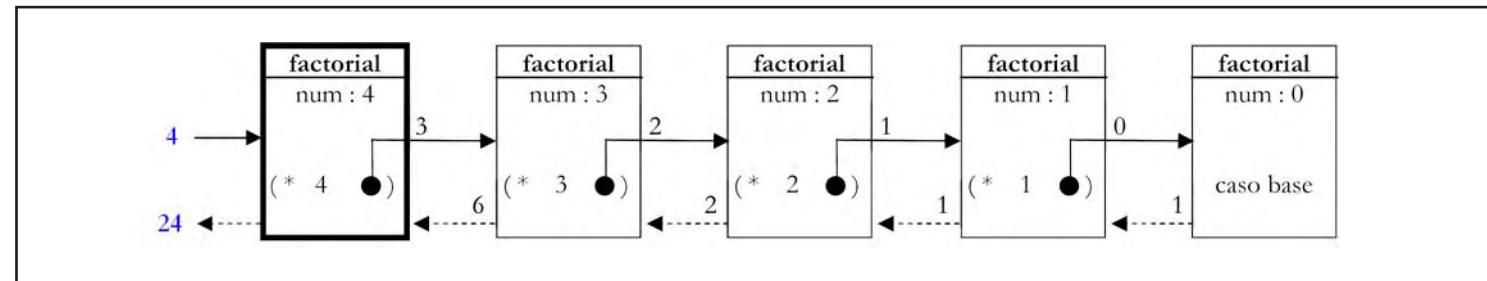
Neste ponto, atingiu-se o caso base,  $0! = 1$ , e a expressão vai começar a reduzir, pois os processos suspensos vão começar a receber a informação que pediram.

```
(factorial 4) = (* 4 (* 3 (* 2 (* 1 (factorial 0)))))  
= (* 4 (* 3 (* 2 (* 1 1))))  
= (* 4 (* 3 (* 2 1)))  
= (* 4 (* 3 2))  
= (* 4 6)  
= 24
```



**Identifique a multiplicação que esteve mais tempo suspensa.  
Caso não consiga, preste atenção ao texto e à figura que se seguem...**

O funcionamento do procedimento factorial, para a chamada (factorial 4), pode também ser observado na figura, que deverá começar a ser analisada da esquerda para a direita.



Enquanto não se atinge o caso base (representado pelo rectângulo mais à direita) vão sendo gerados processos, todos com vista a realizarem tarefas idênticas, mas de dimensão sucessivamente mais pequena. A chamada (factorial 4) gera o processo representado pelo rectângulo mais à esquerda, em que num = 4. Este processo, ao calcular  $(* num (factorial 3))$ , gera um novo processo correspondente à chamada (factorial 3) e suspende a sua actividade até obter uma resposta.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECUSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

A geração sequencial de novos processos continuará até se atingir o caso base, com `num = 0`. Este processo pode imediatamente responder, devolvendo o valor 1. Observa-se agora que os processos suspensos vão poder terminar a sua tarefa, o que ocorrerá pela ordem inversa à criação desses mesmos processos. Assim, o primeiro processo criado para responder à chamada (`factorial 4`) será o último a terminar, situação que ocorre quando devolver o valor 24, depois de ter recebido o valor 6, do processo anterior.

É agora possível delinear a estratégia para escrever procedimentos recursivos:

- Identifique o caso base ou condição de terminação, caso para o qual se conhece a solução do problema. No exemplo `factorial`, a condição de terminação é `num = 0`, para a qual  $0! = 1$
- Identifique a operação de redução a aplicar sucessivamente, até se atingir o caso base. No mesmo exemplo, a operação de redução é `sub1` aplicada a `num`
- Escreva o procedimento recursivo começando pelo caso base, passando seguidamente ao caso geral. O caso geral envolve uma chamada recursiva ao procedimento que se pretende definir, chamada essa que deverá aproximar-se do caso base. Ainda, no exemplo `factorial`, na situação correspondente a `num`, a chamada recursiva passaria a ser sobre (`sub1 num`)

### Exemplo 1

Escreva em Scheme o procedimento `soma-digitos` que aceita um argumento inteiro e positivo, `numero`, e devolve a soma dos dígitos decimais do argumento.

Vamos começar com o desenvolvimento manual de algumas situações de funcionamento do procedimento pretendido.

```
> (soma-digitos 823)
13
> (soma-digitos 000000)
0
> (soma-digitos -823)
Número negativo!!!
```

Para definir este procedimento, a ideia a explorar resume-se a somar os dígitos do argumento, um a um, começando pelo menos significativo (o da direita), até atingir o mais significativo (o da esquerda).



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Para aplicar a recursividade, toma-se o dígito menos significativo, que será somado à soma dos dígitos restantes, começando assim uma operação de redução do problema. Por exemplo,

```
(soma-digitos 823) = (+ 3 (soma-digitos 82))
```

Neste exemplo, a dimensão inicial do problema é 3, pois o número 823 é composto por 3 dígitos. Depois da redução, passou a um problema de dimensão 2, correspondente ao número 82. A identificação do caso base é fácil. Perante um número inferior a 10, portanto, com um só dígito, a soma dos dígitos desse número é conhecida, pois é o próprio número.

```
(soma-digitos 823) = (+ 3 (soma-digitos 82))
                     = (+ 3 (+ 2 (soma-digitos 8)))      <--- caso base

                     = (+ 3 (+ 2 8))
                     = (+ 3 10)
                     = 13
```

Não esqueça, neste exemplo, que é necessário verificar se o número é negativo e, se assim for, é apenas visualizada a mensagem correspondente.

Mas suponha, para já, que o número é positivo.

- O caso base - número menor que 10, pois a resposta torna-se imediatamente conhecida: a soma dos dígitos é o próprio número.
- A operação de redução - quociente da divisão inteira (quotient em Scheme) do número por 10 , que equivale a retirar o dígito menos significativo ao número, reduzindo assim a dimensão do problema.
- O caso geral - somar o dígito menos significativo à soma dos dígitos do número entretanto reduzido. O dígito menos significativo poderá ser obtido através do resto da divisão inteira (remainder em Scheme) do número por 10.

Na solução que se segue foram identificadas duas tarefas auxiliares em que se decompõe a tarefa soma-digitos, designadas por digito-menos-significativo e retira-digito-menos-significativo.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
(define digito-menos-significativo
  (lambda (n)
    (remainder n 10))) ; o resto é o dígito menos significativo

> (digito-menos-significativo 823)
3

(define retira-digito-menos-significativo
  (lambda (n) ; este quociente resulta o número
    (quotient n 10))) ; sem o dígito menos significativo

> (retira-digito-menos-significativo 823)
82
```



Experimente os procedimentos auxiliares `digito-menos-significativo` e `retira-digito-menos-significativo`.

Finalmente, a definição e teste do procedimento `soma-digitos`.

```
(define soma-digitos
  (lambda (num)
    (cond ((negative? num)
           (display "Número negativo!!!"))
          ;
          ((< num 10) num) ; o caso base
          ;
          (else (+ (digito-menos-significativo num) ; o caso geral
                    (soma-digitos (retira-digito-menos-significativo num)))))))

> (soma-digitos 823)
13
> (soma-digitos -823)
Número negativo!!!
> (soma-digitos 1234567890)
45
>
```



Experimente o procedimento `soma-digitos`.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

R E 1 2 3

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



## Exercício 2

Verifique que, em soma-digitos, o teste negative? é repetido sucessivas vezes, quando, na realidade, apenas seria necessário da primeira vez.



O teste referido quantas vezes tem lugar inutilmente?

Para resolver esta situação, bem como outras do mesmo tipo, basta que imagine um procedimento auxiliar que trate apenas o caso ideal, em que o argumento nunca é negativo. Seja esse o procedimento soma-digitos-de-positivo:

```
(define soma-digitos-de-positivo
  (lambda (num)
    (cond ((< num 10) num)
          (else (+ (digito-menos-significativo num)
                    (soma-digitos-de-positivo
                      (retira-digito-menos-significativo num)))))))
```

Tendo por base soma-digitos-de-positivo, escreva o procedimento soma-digitos-melhorado que não repete o teste negative? como acontecia em soma-digitos.

```
> (soma-digitos-melhorado 823)
13
> (soma-digitos-melhorado -823)
Número negativo!!!
> (soma-digitos-melhorado 1234567890)
45
>
```



Desenvolva e experimente o procedimento soma-digitos-melhorado.

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

R E 1 2 3

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

### Exercício 3

Escreva em Scheme o procedimento `soma-n-digitos`, com os parâmetros `n` e `nd` que são números inteiros positivos, que devolve a soma nos `nd` dígitos menos significativos de `n`.

Apresente o diagrama de fluxo de dados que relate os vários procedimentos que venha a utilizar. Encare os exemplos que se seguem como um potencial conjunto de situações de teste e interprete cuidadosamente os respectivos resultados.

```
> (soma-n-digitos 123 2)  
5  
> (soma-n-digitos 123 3)  
6  
> (soma-n-digitos 123 50)  
6  
> (soma-n-digitos 123 0)  
0
```

Pista

Recorra a um procedimento auxiliar `nd-menos-significativos` com os parâmetros `numero` e `nd`, que devolve o valor correspondente ao número composto pelos `nd` dígitos menos significativos de `numero`. Depois, bastará chamar `soma-digitos-melhorado` do exercício anterior.

```
> (nd-menos-significativos 123 2)  
23  
> (nd-menos-significativos 123 3)  
123  
> (nd-menos-significativos 123 50)  
123  
> (nd-menos-significativos 123 0)  
0
```



Desenvolva e teste o procedimento pedido.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

R E 1 2 3

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



### Exercício 4

O procedimento soma-digitos-mais-significativos aceita dois argumentos *n* e *nd*, e devolve a soma dos *nd* dígitos decimais mais significativos do número *n*. Escreva em Scheme o procedimento soma-digitos-mais-significativos.

Pista

Comece por escrever o procedimento *n-digitos* que determina o número de dígitos de um número *n*. Por exemplo, (*n-digitos* 4100) devolve 4. Para além desta pista, verifique que o dígito mais significativo de *n* pode ser determinado através de (*quotient n (expt 10 (- (n-digitos n) 1))*).



Desenvolva e teste o procedimento pedido.

### Exercício 5

- Este não é muito simples, pois dá muito que pensar!

O procedimento *a-subir* tem apenas um parâmetro, *num*, que é um número inteiro positivo. Este procedimento analisa os dígitos de *num*, do menos significativo para o mais significativo, e visualiza *sobe*, se os dígitos se apresentarem em ordem ascendente, ou *nao sobe*, se não se verificar a ordem ascendente dos dígitos.

Escreva em Scheme o procedimento *a-subir* e apresente o diagrama de fluxo de dados que relate os procedimentos utilizados.

Considere as seguintes situações de funcionamento do procedimento *a-subir*, que deverão ser posteriormente utilizadas como testes.

```
> (a-subir 97431)
  sobe
> (a-subir 79431)
  nao sobe
> (a-subir 10)
  sobe
> (a-subir 11)
  nao sobe
> (a-subir 0)
  sobe
```

# SCHEME

## na descoberta da programação



Desenvolva e teste o procedimento pedido.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECUSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Módulo 2.2 - Processos e os recursos computacionais que consomem

Um procedimento é um pedaço de texto que pode ser lido, mas que nada faz! Por outro lado, a chamada de um procedimento gera um processo no interior do computador que, normalmente, produzirá resultados, à custa obviamente dos recursos computacionais que consome.

Neste módulo é mostrado que os procedimentos recursivos geram processos recursivos ou processos iterativos e, conforme o caso, são normalmente muito diferentes os recursos computacionais que consomem. São também apresentadas as recursividades linear e em árvore, e como exemplo desta última surge a sequência de Fibonacci. Para avaliar o comportamento dos processos face à dimensão do problema que resolvem, introduz-se a ordem de crescimento ou ordem de complexidade e apresentam-se exemplos de ordem de crescimento constante, linear, exponencial, logarítmica e quadrática.

### Palavras-Chave

Procedimento recursivo, processo recursivo, processo iterativo, recursividade em cauda (*tail recursion*), recursos computacionais de tempo e espaço de memória, recursividade linear, recursividade em árvore, nó, raiz, folha, ramos, árvore binária, sequência de Fibonacci, permutações de  $n$  elementos, arranjos de  $n$  elementos  $k$  a  $k$ , combinações de  $n$  elementos  $k$  a  $k$ , ordem de crescimento ou ordem de complexidade, dimensão do problema, ordem de crescimento constante, linear, exponencial, logarítmica e quadrática.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECUSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Procedimento recursivo chama-se a si próprio!

Saberá facilmente se um procedimento é recursivo verificando no corpo desse procedimento a existência de alguma chamada para si próprio, como acontece no procedimento factorial.

```
(define factorial
  (lambda (num)
    (if (zero? num)
        1
        (* num (factorial (sub1 num))))))
```

```
> (factorial 4)
24
>
```



Teste o procedimento factorial.

Em factorial, experimente substituir (sub1 num) por (- num 2) e explique o que acontece.

## Processo recursivo suspende actividade e cativa memória

Um procedimento ao ser chamado cria um processo que executará a tarefa definida nesse procedimento.

Por exemplo, uma chamada do procedimento factorial cria um processo que toma um inteiro como argumento, processa-o e no final devolve o factorial desse inteiro. Pela definição do procedimento factorial percebe-se que uma chamada deste desencadeia uma sequência de processos, até se atingir o caso base. O que se segue corresponde à sequência de processos criados pela chamada (factorial 4), sequência que vai crescendo até ao caso base, correspondente à chamada (factorial 0). Atingido este ponto, verifique a morte dos processos da sequência, na ordem inversa à sua criação.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(factorial 4) = (* 4 (factorial 3))
= (* 4 (* 3 (factorial 2)))
= (* 4 (* 3 (* 2 (factorial 1))))
= (* 4 (* 3 (* 2 (* 1 (factorial 0)))))    <-- caso base

= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
= 24
```

A chamada (factorial 4) cria um processo que provoca a chamada (factorial 3) e fica suspenso até receber a resposta desta última chamada. Por sua vez, a chamada (factorial 3) cria também um processo que chama (factorial 2) e assim, sucessivamente, vão sendo criados outros processos até se atingir o caso base, (factorial 0). Verifique a existência de uma cadeia de multiplicações suspensas, que significa que esta cadeia vai progressivamente ocupando memória do computador.

A partir do caso base, as multiplicações vão sendo calculadas, a expressão vai reduzindo de tamanho e a memória vai sendo libertada.

Pode facilmente deduzir que os processos gerados pelas chamadas do procedimento factorial gastam recursos computacionais (tempo de cálculo e memória) que são proporcionais ao valor do argumento. Quanto maior for o valor do argumento maior será o tempo de cálculo e maior será o espaço de memória ocupado. Não será difícil aceitar que uma chamada (factorial 8) gaste aproximadamente o dobro do tempo e da memória que é gasto pela chamada (factorial 4).



**Os recursos de tempo e espaço gastos por (factorial 8) correspondem ao dobro dos recursos gastos por (factorial 4)**

**1- mostre que assim deve ser.**

**E agora um pouco mais difícil...**

**2- e se lhe dissermos que esta previsão de gastos é mais correcta para valores elevados do argumento. Por exemplo, em vez de 4 e 8, 40 e 80, ou 4000 e 8000.**

**Concorda com esta afirmação?**



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECUSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Esta característica, que consiste em diferir ou suspender a execução das operações, guardando-as em memória até encontrar o caso base, é típica dos chamados processos recursivos.



Já consegue distinguir um procedimento do respectivo processo?

Convém aqui frisar a diferença entre procedimento e processo. O procedimento é o próprio texto, uma entidade estática que se pretende suficientemente legível pelo humano e que expressa uma ideia numa certa linguagem. O processo é uma entidade dinâmica que existe no interior do computador, gasta recursos, mas fornece resultados.

### Processo iterativo gerado por procedimento recursivo!

Vamos encarar o cálculo do factorial de uma outra forma, que permitirá definir um novo procedimento diferente de `factorial`. O novo procedimento cria um processo com outras características que não as indicadas para `factorial`, mas no final continuará a fornecer o resultado pretendido, ou seja, o factorial de n.

Em vez da definição recursiva, utilizada em `factorial`     $n! = n * (n-1) !$

agora a ideia a explorar baseia-se em     $n! = n * (n-1) * (n - 2) * \dots * 3 * 2 * 1$

A implementação desta nova ideia resume-se a uma sequência de multiplicações, n que multiplica por  $(n-1)$ , o respectivo resultado multiplicado por  $(n-2)$  e assim sucessivamente até que se encontre  $n = 0$ , que já não será multiplicado. Esta implementação baseia-se na existência de um acumulador que inicialmente toma o valor 1 (valor neutro da multiplicação). Seguidamente, o valor do acumulador será multiplicado por n, resultado que vai actualizar o valor do acumulador. Este processo repete-se com  $n-1$ ,  $n-2$  e assim sucessivamente até  $n=0$ , altura em que o valor de factorial de n se encontrará no acumulador.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECUSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Tomando (factorial 4) como exemplo, n vai começar com o valor 4 e acumulador com o valor 1.

iteração	n	acumulador	acumulador (no final da iteração)
1 <sup>a</sup>	4	1	4 * 1 = 4
2 <sup>a</sup>	3	4	3 * 4 = 12
3 <sup>a</sup>	2	12	2 * 12 = 24
4 <sup>a</sup>	1	24	1 * 24 = 24
5 <sup>a</sup>	0	24	

resultado

O acumulador é actualizado, no final de cada iteração, com o resultado do seu próprio valor multiplicado por n. Este n vai sendo reduzido de 1 em cada iteração, até se atingir o caso base, n = 0.

Podemos agora escrever procedimento factorial-novo, baseado na ideia acabada de expor, introduzindo um segundo parâmetro designado por acumulador.

```
(define factorial-novo
  (lambda (n acumulador)
    (if (zero? n)
        acumulador ; caso base, em que n=0...
        ;
        (factorial-novo (sub1 n) (* n acumulador)))))) ; caso geral

> (factorial-novo 4 1)
24
```



Teste o procedimento factorial-novo.

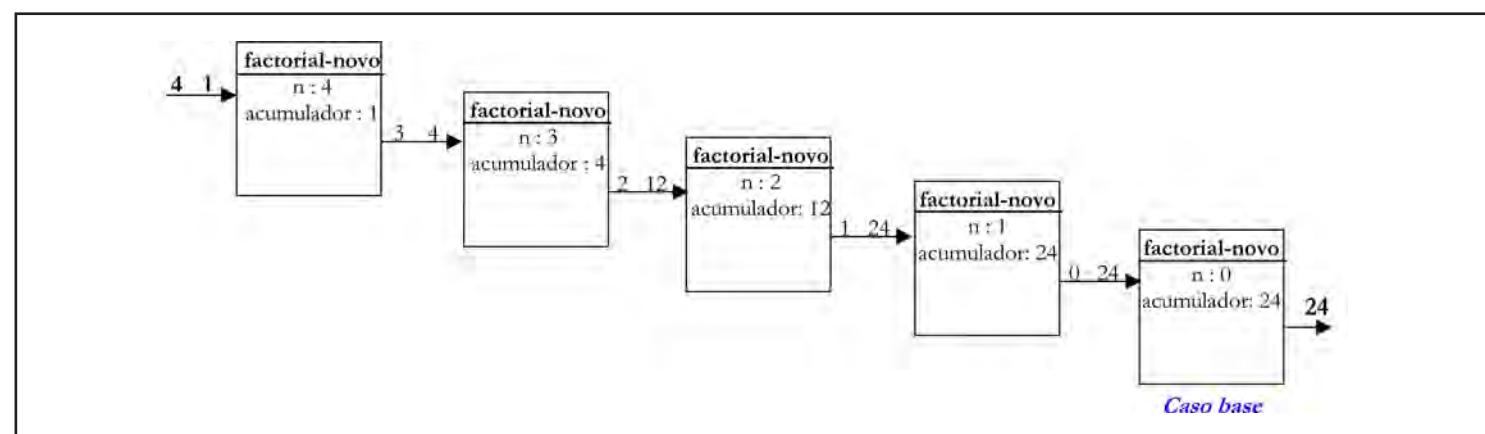
Experimente dar ao segundo argumento valores diferentes de 1 e justifique os resultados.



Analise agora como se desenvolve a chamada (factorial-novo 4 1) seguindo a definição de factorial-novo.

(factorial-novo 4 1)	como n não é 0, resulta	(factorial-novo 3 (* 4 1))
(factorial-novo 3 4)	como n não é 0, resulta	(factorial-novo 2 (* 3 4))
(factorial-novo 2 12)	como n não é 0, resulta	(factorial-novo 1 (* 2 12))
(factorial-novo 1 24)	como n não é 0, resulta	(factorial-novo 0 (* 1 24))
(factorial-novo 0 24)	como n = 0, resulta	24, valor no acumulador

Na representação gráfica do mesmo exemplo, facilmente se observa que os sucessivos processos criados não suspendem a actividade (como acontecia com os processos recursivos), pois não ficam à espera de uma resposta. Limitam-se a lançar um processo idêntico a eles próprios, mas com outros argumentos, sempre na direcção do caso base, morrendo em seguida.



Esta característica, que consiste em não suspender a execução das operações, como acontecia nos processos recursivos, é típica dos chamados processos iterativos.

Processo iterativo menos consumidor de recursos que o processo recursivo

O procedimento factorial-novo continua a ser um procedimento recursivo, pois encontrará no seu corpo uma chamada a si próprio. Verifique, contudo, que uma chamada deste procedimento, por exemplo (`factorial-novo 4 1`), cria um processo que não fica à espera de qualquer resposta. Este processo, antes de morrer, provoca a chamada (`factorial 3 4`).

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Este modo de funcionamento continua com (`factorial 2 12`), depois com (`factorial 1 24`) e, finalmente, com (`factorial 0 24`) que, por corresponder ao caso base, responde devolvendo o valor 24. Contrariamente ao que acontecia com `factorial`, o procedimento `factorial-novo` vai executando as multiplicações à medida que aparecem, o que implica um gasto constante em memória (essencialmente, apenas a memória para guardar `num` e acumulador).

Pode deduzir que os processos gerados pelas chamadas do procedimento `factorial-novo` gastam recursos constantes em memória e recursos em tempo de máquina proporcionais ao valor do argumento. Ou seja, o processo gerado pela chamada (`factorial-novo 8 1`) deve gastar aproximadamente o dobro do tempo e a mesma memória que são gastos pelo processo gerado pela chamada (`factorial-novo 4 1`).



**Mostre que no procedimento `factorial-novo`**

- 1- o gasto em memória é constante (contrariamente ao que acontecia com o procedimento `factorial`, cujo gasto em memória era proporcional a  $n$ )**
- 2- e que o gasto em tempo é proporcional a  $n$  (como acontecia com o procedimento `factorial`)**

O procedimento `factorial-novo` gera processos iterativos que, em geral, são menos consumidores de recursos computacionais do que os correspondentes processos recursivos, como os gerados por `factorial`.

Não parece muito correcto fazer chamadas a procedimentos que calculam factoriais, fornecendo dois argumentos: o número de que se pretende conhecer o factorial e o valor que inicializa o acumulador, normalmente o valor 1. Para ultrapassar este tipo de situação, pode definir o procedimento `factorial-iter`, baseado em `factorial-novo`.

```
(define factorial-iter
  (lambda (num)
    (factorial-novo num 1)))
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECUSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Forma expedita de identificar o tipo de processo gerado por um procedimento

Reconhecer o tipo de processo (recursivo ou iterativo) gerado por um procedimento recursivo surge assim como uma necessidade importante para quem programa, uma vez que para a mesma tarefa os recursos consumidos podem ser muito diferentes, num e noutro casos.



**Em que situação se torna praticamente indiferente usar procedimentos que criam processos recursivos ou iterativos?**

Compare cuidadosamente os procedimentos `factorial` e `factorial-novo`, ou, mais especificamente, o caso geral ou passo recursivo destes procedimentos, pois irá encontrar uma forma expedita de reconhecer se um procedimento recursivo gera processos recursivos ou iterativos.

Em `factorial`, que gerava processos recursivos, o caso geral corresponde a uma multiplicação

```
(* n (factorial (sub1 n)))
```

em que um dos operandos é a chamada recursiva `(factorial (sub1 n))`. Como os operandos são calculados em primeiro lugar, a multiplicação terá que ficar suspensa até que se conheça o valor correspondente a `(factorial (sub1 n))`.

Assim, o processo correspondente à execução da multiplicação entra em suspensão e é mantido em memória, ao mesmo tempo que é lançado um novo processo correspondente a `(factorial (sub1 n))`. Estamos claramente perante a criação de uma sequência de processos recursivos que vão ficando suspensos até se chegar ao caso base.



**Analisando o caso geral do procedimento recursivo `factorial`, já sabe como identificar que ele gera uma sequência de processos recursivos?**



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Por outro lado, em `factorial-novo`, que gera processos iterativos, o caso geral corresponde a uma chamada recursiva

```
(factorial-novo (sub1 n) (* n acumulador)))
```

Agora, a multiplicação encontra-se num dos argumentos da chamada recursiva e terá, por este motivo, de ser calculada em primeiro lugar. Só depois dos argumentos calculados é que a chamada recursiva terá lugar, sendo assim o último passo do procedimento. Esta recursividade, em que a chamada recursiva se encontra na cauda do procedimento, é designada por recursividade em cauda (*tail recursion*).

Na recursividade em cauda, quando um novo processo é criado, o anterior não fica suspenso, porque morre e não continua a exigir espaço de memória. Esta situação cria uma sequência de processos iterativos que termina com o caso base, ainda que estes processos não fiquem suspensos como acontecia com os processos recursivos.



**Analisando o caso geral do procedimento recursivo `factorial-novo`, já sabe como identificar que ele gera uma sequência de processos iterativos?**

Em termos de recursos de memória, os procedimentos recursivos que geram processos iterativos (recursividade em cauda, como se verifica em `factorial-novo`) são mais eficientes que os procedimentos recursivos que geram processos recursivos (como em `factorial`). Pelo menos assim acontece em Scheme, cujo interpretador, segundo norma do IEEE, deve apresentar uma implementação com recursividade em cauda (*tail-recursive*), em que um processo iterativo pode ser gerado por um procedimento recursivo. Noutras linguagens, os processos iterativos são, normalmente, gerados a partir de estruturas de repetição: `do`, `repeat`, `for`, `while`, e não através de procedimentos recursivos.

Em termos de tempo de cálculo, o comportamento dos processos iterativos pode ser menos consumidor do que nos correspondentes processos recursivos. Não é o caso dos procedimentos `factorial` e `factorial-novo`, já que em ambos os recursos em tempo são semelhantes, ou seja, proporcionais ao valor do argumento.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECUSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Em situações em que os recursos computacionais não constituem uma grande preocupação, o melhor será recorrer a procedimentos recursivos que geram processos recursivos, já que se baseiam, normalmente, em ideias muito mais simples e mais fáceis de implementar do que os procedimentos recursivos que geram processos iterativos. Isto pode confirmar-se analisando as ideias que estiveram na base das soluções encontradas para factorial e factorial-novo, bem como as que irão ser utilizadas nos exemplos que se seguem.

### Exemplo 1

Baseando-se numa solução recursiva, escrever um procedimento que toma um argumento  $n$ , inteiro positivo, e que gera um processo recursivo para calcular  $n + (n-1) + \dots + 1 + 0$ .

Neste exemplo o caso base é  $n = 0$ , a operação de redução é sub1 aplicada a  $n$  e o caso geral é adicionar  $n$  à soma dos restantes números da série, suposta conhecida. Da tradução desta ideia em Scheme resulta o procedimento soma-sequencia.

```
(define soma-sequencia
  (lambda (n)
    (if (zero? n)
        0
        (+ n
            (soma-sequencia (sub1 n))))))
```

> (soma-sequencia 3)

6

Reconhece-se nesta solução a geração de processos recursivos pois, no caso geral ou passo recursivo, a chamada recursiva é um dos operandos da expressão, não sendo a última operação do procedimento. Isto significa que as operações de adição vão ficar suspensas e guardadas em memória até que se encontre o caso base.

Para o mesmo problema, tenta-se agora um procedimento que execute a mesma tarefa mas que gere processos iterativos. Com este objectivo, introduz-se um acumulador para guardar o resultado das sucessivas adições, acumulador que na chamada do procedimento será inicializado com zero, o elemento neutro da adição.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
(define soma-sequencia-aux
  (lambda (n acumulador)
    (if (zero? n)
        acumulador
        (soma-sequencia-aux (sub1 n)
                            (+ n acumulador))))))
```

Para esconder o parâmetro acumulador, poderia ser utilizado o procedimento soma-sequencia-iter.

```
(define soma-sequencia-iter
  (lambda (numero)
    (soma-sequencia-aux numero 0)))
```



Teste os procedimentos soma-sequencia e soma-sequencia-iter.

Experimente o procedimento auxiliar dando ao segundo argumento valores diferentes de 0 e justifique os resultados.

### Exercício 1

Vamos agora imaginar uma operação que designaremos por misturar-dois-inteiros.

Em que consistirá a referida operação? Comece por analisar o diálogo que se segue.

```
> (misturar-dois-inteiros 123 789)  
718293  
> (misturar-dois-inteiros 12345 89)  
1238495  
> (misturar-dois-inteiros 89 12345)  
1234859
```

a- Escreva em Scheme um procedimento recursivo para misturar dois inteiros, o qual deverá gerar processos recursivos.

b- Escreva um segundo procedimento para fazer a mesma coisa, mas gerando processos iterativos.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Pista: Será necessário, na alínea b, utilizar um procedimento auxiliar com um terceiro parâmetro para servir de acumulador?



Desenvolva e teste os dois procedimentos pedidos.

### Exercício 2 - Um desafio muito mais difícil...

Vamos agora imaginar uma operação que designaremos por `misturar-dois-numeros`.

Em que consistirá a referida operação? Comece por analisar o diálogo que se segue.

```
> (misturar-dois-numeros 123 789)  
718293  
> (misturar-dois-numeros 12345.67 89.1)  
1238495.167  
> (misturar-dois-numeros 89.1 12345.67)  
1234859.617
```

a- Escreva em Scheme um procedimento recursivo para misturar dois números, o qual deverá gerar processos recursivos.

b- Escreva um segundo procedimento para fazer a mesma coisa, mas gerando processos iterativos.



Desenvolva e teste os dois procedimentos pedidos.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECUSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

### Exemplo 2

Escreva em Scheme o procedimento `troca-posi` que espera um único argumento, inteiro positivo, e que o visualiza com os dígitos na ordem inversa e com o carácter "." entre eles.

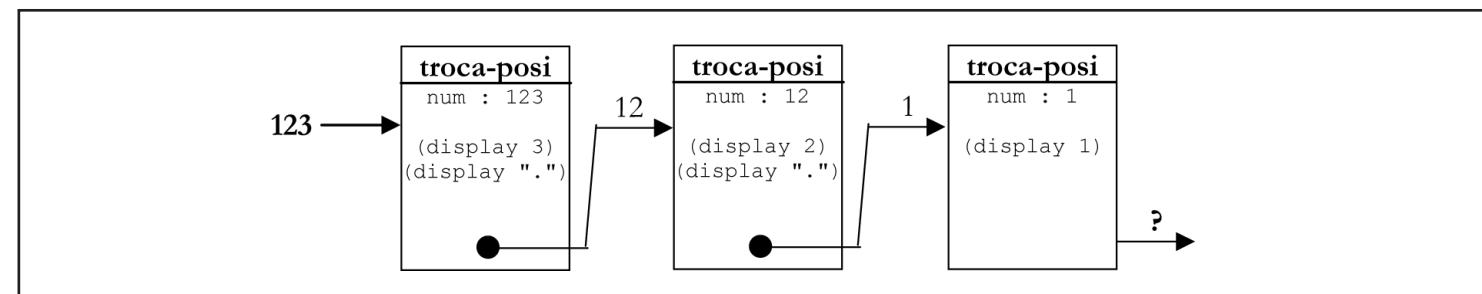
```
> (troca-posi 123)
3.2.1
> (troca-posi 5)
5
```

A solução que se apresenta baseia-se em dois procedimentos auxiliares, `digito-menos-significativo` e `retira-digito-menos-significativo`.

```
(define troca-posi
  (lambda (num)
    (cond
      ((< num 10)           ; caso base
       (display num))
      (else                  ; caso geral
       (display (digito-menos-significativo num))
       (display "."))
       (troca-posi (retira-digito-menos-significativo num))))))
```

O caso base corresponde a um número menor que 10, representado por um único dígito. O caso geral traduz-se pela visualização do dígito menos significativo seguido de um "." e finalizando como uma chamada recursiva a `troca-posi`, com um argumento equivalente ao actual retirado o dígito menos significativo.

Neste procedimento é utilizada recursividade em cauda, pois a última operação é a chamada recursiva. Assim, o processo gerado é iterativo, o que se comprova com a representação gráfica da chamada `(troca-posi 123)`. Também se verifica, pela representação gráfica, que o valor devolvido pelos processos não é importante, ao contrário do que acontece com o efeito colateral correspondente à visualização no ecrã.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Desenvolva e teste os procedimentos `digito-menos-significativo` e `retira-dígito-menos-significativo`, e, depois, teste o procedimento `troca-posi`.

No procedimento `troca-posi`, experimente substituir 10 por 100 e justifique os resultados.

### Exemplo 3

Escreva em Scheme o procedimento `nao-troca-posi` que espera um único argumento, inteiro positivo, e que o visualiza com os dígitos na ordem normal e com o carácter `".` entre eles.

```
> (nao-troca-posi 123)
1.2.3
> (nao-troca-posi 5)
5
```

A solução que se apresenta parece idêntica à utilizada para `troca-posi`, do exemplo anterior, com umas trocas de posição de algumas instruções, mas uma análise mais cuidada leva a concluir que o processo agora gerado tem um comportamento muito diferente.

```
(define nao-troca-posi
  (lambda (num)
    (cond
      ((< num 10)
       (display num))
      (else (nao-troca-posi (retira-dígito-menos-significativo num))
            (display "."))
            (display (digito-menos-significativo num))))))
```

Neste caso não aparece recursividade em cauda, ou seja, os processos gerados não são iterativos, mas sim recursivos, uma vez que a chamada recursiva não é o último passo de `nao-troca-posi`. Como pode verificar, esta chamada é seguida por duas chamadas a `display`. A execução dos `display` vai ser suspensa e a morte do processo vai ser adiada, enquanto outro processo idêntico é criado. E isto acontece até se atingir o caso base.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECUSIVIDADE

R E 1 2 3

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

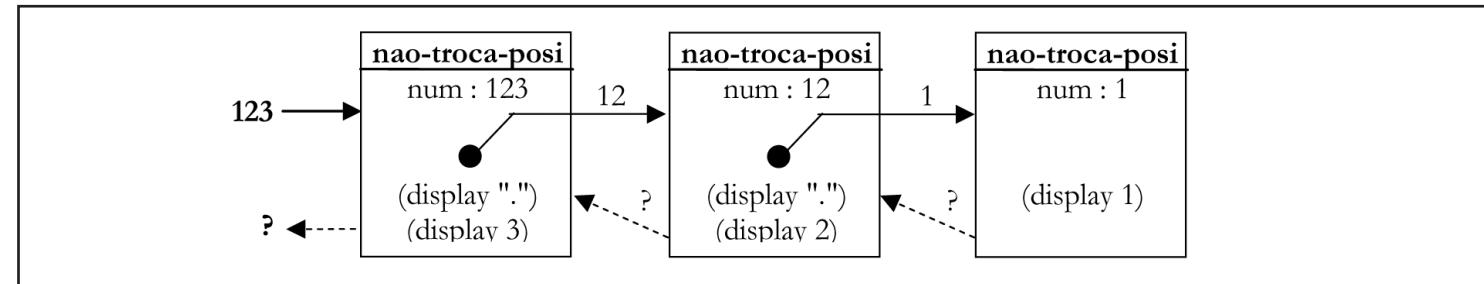
5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

Verá, nesta representação gráfica, que o valor devolvido pelos processos não é importante, ao contrário do que acontece com o efeito colateral correspondente à visualização no ecrã. A primeira visualização a acontecer situa-se no processo mais à direita, (*display 1*).



Teste o procedimento *nao-troca-posi*.

### Recursividade linear e em árvore

No caso geral ou passo recursivo dos procedimentos *factorial* e *factorial-novo*, estava presente apenas uma chamada recursiva dos respectivos procedimentos. Por exemplo, no passo recursivo de *factorial*, `(* n (factorial (sub1 n)))`, a chamada recursiva presente, `(factorial (sub1))`, é única. A isto se chama recursividade linear. As várias representações gráficas da sequência dos processos gerados são bem prova desta linearidade.

Alguns casos, que serão analisados nesta secção, usam no passo recursivo duas ou mais chamadas do procedimento, aquilo a que se chama recursividade em árvore (*tree recursion* ou *multiway recursion*), designação que advém da forma típica das figuras utilizadas para esboçar a sequência dos processos que criam, ou seja, uma árvore em posição invertida, com a raiz para cima e a copa para baixo.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

R E 1 2 3

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

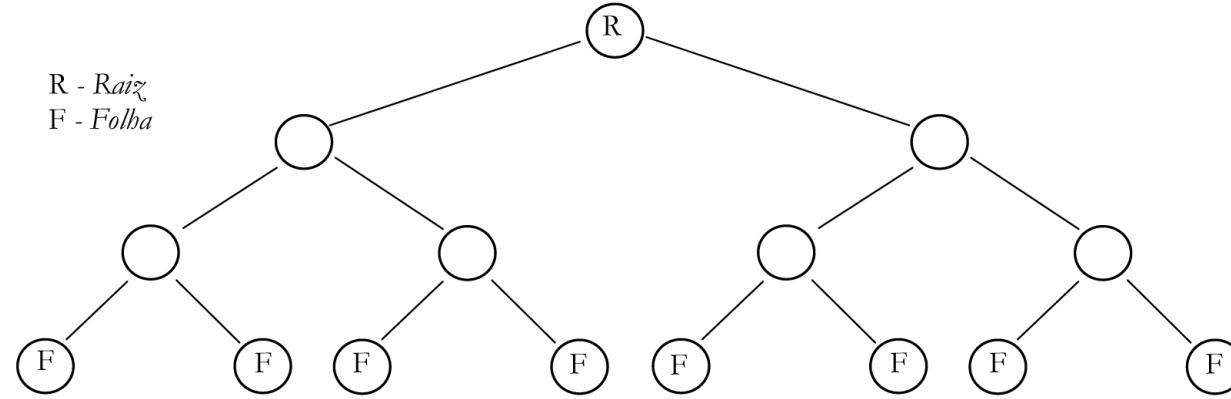
5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

R - Raiz  
F - Folha



Na figura os círculos representam os nós da árvore, sendo que o mais acima é a raiz e os mais abaixo, os nós terminais, são as folhas. De cada nó não terminal saem dois ramos ou duas sub-árvores (ramo da esquerda e ramo da direita), o que é uma característica das árvores binárias.

Um exemplo muitas vezes utilizado para ilustrar a recursividade em árvore é a geração da sequência de Fibonacci, na qual cada número é igual à soma dos dois números anteriores, sendo os dois primeiros, por convenção, 0 e 1. Os primeiros elementos são 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...e a sequência pode ser gerada através de uma definição recursiva.

$$fib(n) = \begin{cases} n & \text{se } n < 2 \\ fib(n-1) + fib(n-2) & \text{nos outros casos} \end{cases}$$

Desta definição deduz-se o procedimento respectivo, que apresenta duas chamadas no passo recursivo.

```

(define fib
  (lambda (n)
    (if (< n 2)
        n
        ; caso base
        (+ (fib (- n 1)) ; caso geral ou passo recursivo
           (fib (- n 2)))))) ; com duas chamadas de fib

> (fib 4)
3
  
```



# SCHEME

## na descoberta da programação



Teste o procedimento fib.

Experimente com um argumento entre 30 e 40, vá depois aumentado esse argumento, de 1 em 1, meça o tempo de resposta e comente os resultados...

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

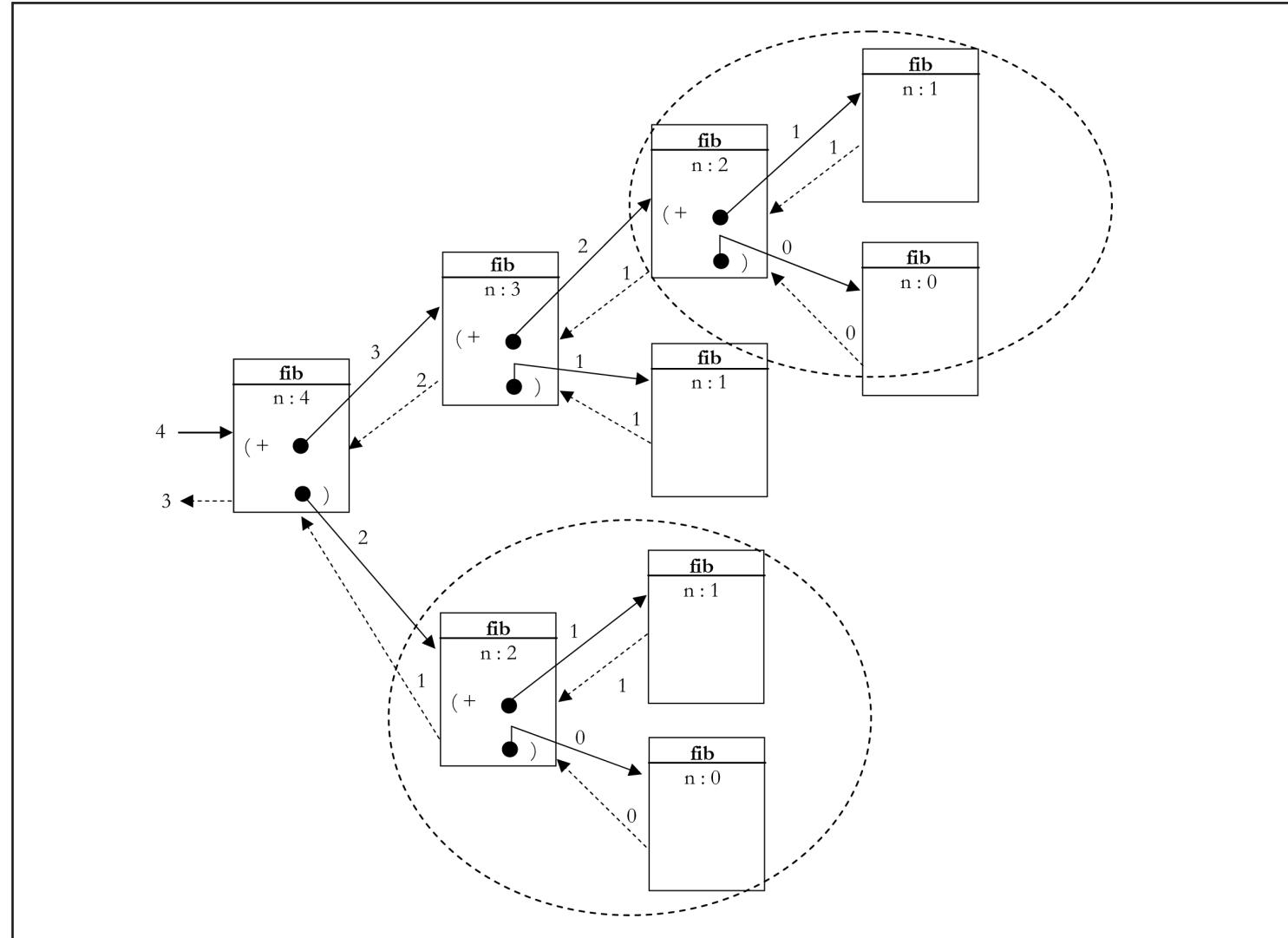
### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Por uma questão de espaço, optou-se por desenhar a árvore rodada. Assim, a árvore tem a raiz em  $(\text{fib } 4)$ , no lado esquerdo, e as folhas em  $(\text{fib } 1)$  e  $(\text{fib } 0)$ , do lado direito. De cada nó (que não é folha) saem 2 ramos, sendo esta por isso uma árvore binária.

Em termos de tempo, o comportamento do processo recursivo em árvore é terrivelmente gastador. Basta verificar na figura que, por exemplo, o cálculo  $(\text{fib } 2)$  é duplicado. O número de chamadas ao procedimento fib é igual ao número de nós da árvore, o que corresponde a um comportamento exponencial. Sendo  $n$  o dado inicial, o número de nós é  $q^n$ , em que  $q$  não chega a ser 2 apesar de saírem 2 ramos de cada nó, uma vez que a árvore não é perfeitamente simétrica.



Prova-se que  $q$ , designado por *golden ratio*, é igual a  $q = (1 + \sqrt{5}) / 2 \approx 1.6180$ , não sendo, propriamente, uma demonstração que nos interesse agora muito.

No entanto, não fica impedido de a procurar.

Em relação ao espaço de memória, verifica-se que o comportamento do processo é apenas linear, conclusão a que se chega seguindo a trajectória indicada pelas setas sobre a árvore. Os cálculos que se mostram nos diferentes ramos da árvore nunca estão todos suspensos ao mesmo tempo, pois vão sendo calculados à medida que se atinge algum dos casos base (as folhas). Por exemplo, o ramo inferior,  $(\text{fib } 2)$ , só será percorrido depois de  $(\text{fib } 3)$  estar totalmente calculado.



Como o Scheme não garante a ordem de cálculo dos operandos de uma expressão,  $(\text{fib } 3)$  poderá ser chamado só depois de  $(\text{fib } 2)$  ser calculado!

Este facto terá alguma influência no que foi dito sobre o comportamento linear do processo?



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

O número máximo de cálculos suspensos, que é de 4, situa-se na zona da árvore com maior profundidade (ramo onde se encontra (fib 3)), profundidade essa que revela uma certa proporcionalidade linear com o número a processar. Variando o número a processar, varia linearmente a profundidade da árvore.

O procedimento recursivo fib gera processos recursivos em árvore com um comportamento exponencial em tempo e linear em memória. Pode sempre tentar um procedimento recursivo que gere um processo iterativo, bastante mais eficiente, quer em tempo (que passa a linear) quer em espaço de memória (que passa a constante), tarefa que se poderá revelar não muito simples.

A ideia a explorar, para a solução iterativa, é manter dois acumuladores, aos quais se associam, em cada iteração, dois números seguidos da sequência de Fibonacci, o número corrente (ac-corrente) e o seguinte (ac-seguinte). Serão inicializados com ac-corrente = fib(0) = 0 e ac-seguinte = fib(1) = 1.

Em cada iteração desenrolar-se-ão, simultaneamente, as seguintes transformações:

- ac-seguinte + ac-corrente vai para ac-seguinte
- ac-seguinte vai para ac-corrente

Para completar a ideia, bastará inicializar um contador com n, o qual, em cada iteração, sofre uma redução de uma unidade. Quando contador atinge zero, a resposta encontra-se em ac-corrente.

```
(define fib-iter
  (lambda (contador ac-corrente ac-seguinte)
    (if (zero? contador)
        ac-corrente
        (fib-iter (sub1 contador)
                  ac-seguinte
                  (+ ac-seguinte ac-corrente)))))
```

Para esconder os parâmetros que não devem ser visíveis pelos utilizadores normais do procedimento, visto que eles foram introduzidos apenas para facilitar a definição de uma ideia de resolução do problema, apresenta-se fib-novo.

```
(define fib-novo
  (lambda (n)
    (fib-iter n 0 1)))
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Teste o procedimento fib-novo.

Experimente fib-novo e fib com os mesmos argumentos, primeiro com valores pequenos, entre 10 e 20, e depois valores maiores, entre 30 e 50, e comente os tempos de resposta.

### Exemplo 4

Ao misturar cores, a partir de um conjunto de cores iniciais, produzem-se novas cores. Supõe-se, neste método de produção, que se utilizam sempre doses iguais de cada cor misturada. Assim, partindo das cores iniciais C1 e C2 conseguem-se as seguintes 3 cores distintas: C1, C2 e C1 + C2 (neste contexto, o sinal + significa mistura de cores em quantidades iguais). Com as cores iniciais C1, C2 e C3 já se conseguem 7 cores: C1, C2, C3, C1 + C2, C1 + C3, C2 + C3 e C1 + C2 + C3.

Estamos interessados em saber quantas cores distintas é possível produzir a partir de 4, 5, ..., ou n cores iniciais.

```
> (n-cores 0)  
0  
> (n-cores 1)  
1  
> (n-cores 2)  
3  
> (n-cores 3)  
7  
> (n-cores 4)  
15  
> (n-cores 5)  
31  
> (n-cores 10)  
??
```

Vamos procurar uma definição recursiva para determinar o número de cores distintas que se conseguem obter com n cores iniciais. Em seguida, escrever em Scheme o procedimento n-cores, com base na definição apresentada, que responda como se indicou.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECUSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A definição recursiva para determinar o número de cores é apresentada, bem como a respectiva justificação.

$$nCores(n) = \begin{cases} n & \text{se } n \leq 1 \\ 1 + 2 * nCores(n-1) & \text{nos outros casos} \end{cases}$$

Com  $n = 0$  cores, não se produz qualquer cor. Com  $n = 1$  cor, só se produz uma cor. Estas duas situações constituem o caso base.

Nos outros casos, considere uma cor qualquer do conjunto inicial de cores.

- essa cor isolada produz 1 cor
- sem essa cor, e apenas com as restantes cores, produzem-se  $nCores(n-1)$  distintas;
- agora se a todas essas  $nCores(n-1)$  se juntar a cor que se isolou, obtém-se mais  $nCores(n-1)$  distintas.

Ao traduzir esta definição recursiva em Scheme, resulta o procedimento `n-cores`.

```
(define n-cores
  (lambda (n)
    (if (<= n 1)
        n ; caso base
        (+ 1 ; caso geral
            (* 2
                (n-cores (sub1 n)))))))
```



Teste o procedimento `n-cores`.



Como desafio, tente identificar o tipo de comportamento da solução apresentada, quer em tempo quer em espaço.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Por uma questão de curiosidade, e também para confirmar os resultados obtidos, vamos aproveitar para relembrar algumas definições de Análise Combinatória.

- Permutações de  $n$  elementos,  $P^n = n!$ , são os agrupamentos formados por esses  $n$  elementos, diferindo uns dos outros apenas pela ordem dos elementos. Permutações de A, B e C,  $3!=6$ , são: ABC, ACB, BAC, BCA, CAB e CBA.
- Arranjos de  $n$  elementos  $k$  a  $k$ ,  $A_k^n = \frac{n!}{(n-k)!}$ , são os agrupamentos de  $k$  elementos que diferem entre si quer pelos elementos que os constituem quer pela ordem. Arranjos de A, B e C 2 a 2,  $3!/1!=6$ , são: AB, BC, AC, CA, BC e CB.
- Combinações de  $n$  elementos  $k$  a  $k$ ,  $C_k^n = \frac{n!}{(n-k)!k!}$ , são os agrupamentos de  $k$  elementos que diferem entre si pelos elementos que os constituem, independentemente da ordem. Combinações de A, B e C 2 a 2,  $3!/(1!2!)=3$ , são AB, AC e BC.

Veja uma tentativa de explicação dos resultados do procedimento `n-cores` baseada em Análise Combinatória.

> `(n-cores 2)`

3

número de cores equivalente a  $C_2^2 + C_1^2 = 1+2 = 3$

> `(n-cores 3)`

7

número de cores equivalente a  $C_3^3 + C_2^3 + C_1^3 = 1+3+3 = 7$



Tente uma alternativa a `n-cores`, baseando-se na definição de combinações de  $n$  elementos  $k$  a  $k$ .

Como desafio, tente identificar o tipo de comportamento da solução a que chegar, quer em relação tempo quer em relação ao espaço. Conseguiu uma solução melhor que a apresentada no Exemplo?

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECUSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Exercício 3

Relativamente à situação do exemplo anterior, suponha agora que, por limitações tecnológicas, terá que haver um limite máximo do número de cores misturadas na obtenção de novas cores.

- Apresente uma definição recursiva para determinar o número de cores distintas que se conseguem obter com  $n$  cores iniciais, sabendo que não se podem misturar mais do que  $k$  cores, em que  $k$  é menor que ou igual a  $n$ .
- Escreva em Scheme o procedimento `n-cores-lim`, com base na definição apresentada, que responda como se mostra mais à frente.
- Apresente a representação gráfica da chamada `(n-cores-lim 3 2)` e verifique se a recursividade é linear ou em árvore.
- Indique se os processos gerados são recursivos ou iterativos e o tipo de comportamento em relação ao tempo e ao espaço.

Observe alguns exemplos da aplicação do procedimento `n-cores-lim` em que o primeiro parâmetro representa o número de cores iniciais e o segundo representa o limite máximo de cores que se podem misturar.

```
> (n-cores-lim 3 0)  
0  
> (n-cores-lim 3 1)  
3  
> (n-cores-lim 3 2)  
6  
> (n-cores-lim 3 3)  
7  
> (n-cores-lim 3 4)  
7
```



Desenvolva e teste o procedimento `n-cores-lim`.



## Exercício 4

Observe com atenção a seguinte pirâmide de números inteiros:

O primeiro e último número de cada linha é sempre 1.

Os números intermédios são iguais à soma dos dois números mais próximos da linha anterior. Por exemplo, o número na 4<sup>a</sup> posição da linha 8, o 35, é igual à soma dos números na 3<sup>a</sup> e 4<sup>a</sup> posições da linha 7, ou seja,  $35 = 15 + 20$ .

O procedimento `ponto-piramide` tem dois parâmetros, `lin` e `col`, e devolve o número que se encontra na linha `lin` e na coluna `col` da pirâmide. Por exemplo, (`ponto-piramide 8 4`) devolve 35.

- Comece por apresentar uma definição recursiva (em notação matemática) para o procedimento ponto-piramide com os parâmetros `lin` e `col` garantidamente inteiros positivos. O procedimento devolve 0 quando `col` for maior que `lin`.
  - Escreva em Scheme o procedimento `ponto-piramide`.
  - Escreva em Scheme o procedimento `linha-piramide`, com o parâmetro `lin`, que visualiza todos os pontos da linha `lin` da pirâmide. Este procedimento pode utilizar o procedimento `ponto-piramide` como auxiliar.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (linha-piramide 4)
linha 4: 1, 3, 3, 1
> (linha-piramide 8)
linha 8: 1, 7, 21, 35, 35, 21, 7, 1
```



Desenvolva e teste os procedimentos ponto-piramide e linha-piramide.



Identifique o tipo de comportamento das soluções a que chegou para os procedimentos ponto-piramide e linha-piramide, quer em relação ao tempo quer em relação ao espaço.

### Avaliação dos recursos computacionais consumidos pelos processos

Neste momento sabe muito bem que os recursos computacionais consumidos pelos processos são o tempo de cálculo e o espaço de memória. Para um mesmo problema, processos resultantes de procedimentos diferentes podem requerer recursos muito diferentes, como já teve ocasião de verificar ao comparar soluções recursivas e iterativas. Uma medida não muito rigorosa dos recursos consumidos, mas suficiente para permitir comparar e escolher, entre várias soluções a mais adequada para um problema, baseia-se na designada ordem de crescimento ou ordem de complexidade (*order of growth* ou *order of complexity*).

Através da ordem de crescimento é possível caracterizar o consumo dos processos em função da dimensão do problema.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Páre um momento e tente imaginar o que será a dimensão de um problema.  
Por exemplo, no factorial, o que será a dimensão do problema?  
E na série de Fibonacci?

A dimensão do problema poderá ser o valor numérico a processar, como acontecia no cálculo do factorial, na determinação da sequência de Fibonacci e nos exemplos das misturas de cores. Em todos estes casos os recursos computacionais gastos pelos processos gerados estavam relacionados com o valor numérico fornecido como argumento. Normalmente, a valores maiores correspondiam recursos gastos também maiores. O objectivo é saber: se a dimensão do problema duplicar, como é que este facto se manifesta nos recursos computacionais consumidos? Também duplicam, variando linearmente? Variam exponencial ou logarítmicamente? Permanecem constantes, ou seja, são independentes do valor do argumento?

Já percebeu que não interessa muito ter uma medida rigorosa, mas apenas uma ideia de como reagirá o processo gerado por um procedimento e qual o seu comportamento, especialmente quando se aumenta (para valores muito elevados) a dimensão do problema.



Antes de avançar pense um pouco na situação que se descreve ou noutra que invente...  
Imagine que vai fazer uma viagem no seu carro. Entra na garagem, abre a porta do carro, instala-se, põe o carro a trabalhar e depois vai fazer o percurso desejado.  
Vai percorrer uns metros, se resolver tomar o pequeno-almoço no café da esquina, ou uns longos quilómetros se planeia ir a Coimbra ou a Lisboa...  
O que representará, neste caso, a dimensão do problema?  
Quais serão, neste caso, os recursos sempre consumidos, qualquer que seja a dimensão do problema, e que são desprezados para efeitos da análise da ordem de crescimento?  
Em relação a estes últimos recursos, em que situações podem ser considerados, de facto, desprezáveis?



Consider

- $n$  o valor que, de alguma forma, reflecte a dimensão do problema;
  - $R(n)$  os recursos consumidos por um processo, para um  $n$  suficientemente elevado, que "dilua" os recursos que são sempre gastos, qualquer que seja a dimensão do problema;
  - $O(f(n))$  a ordem de crescimento de função  $f(n)$ , em que  $f(n)$  pode ser constante, linear ou outra.

Neste contexto,

se  $R(n) \leq K * f(n)$ , para um  $K$  constante e  $n$  suficientemente elevado, então os recursos consumidos apresentam uma ordem de crescimento  $f(n)$ , que se exprime através de  $O(f(n))$ .



Pode parecer estranho, mas uma situação em que  $R(n) \leq 10 * f(n)$  e outra em  $R(n) \leq 1000 * f(n)$ , comparando com  $R(n) \leq K * f(n)$ , vê que terão ambas uma ordem de crescimento igual a  $O(f(n))$ ! Parece-lhe isto aceitável?

**Mas não representarão ambas o mesmo tipo de comportamento quando a dimensão n varia?**

Analise agora algumas situações típicas...

Ordem de complexidade constante

- Se  $R(n)$  tiver um comportamento constante, independente de  $n$ , a ordem de crescimento diz-se constante e representa-se por  $O(1)$ .

Veja a razão desta afirmação: por exemplo, se  $R(n) = 1000$ , podemos encontrar uma constante  $K$  (neste caso,  $K=1000$ ), tal que  $R(n) \leq K * 1$ . Comparando com  $R(n) \leq K * f(n)$ , resulta  $f(n) = 1$ , ou seja,  $O(1)$ .

Aqui poderão começar a surgir algumas dúvidas, pois um processo que gaste constantemente recursos equivalentes a 1 unidade (de tempo ou de memória) e um outro que gaste 1000 unidades do mesmo recurso serão classificados com ordem de crescimento  $O(1)$ . De facto, se existe um  $K_1$  tal que  $1000 \leq K_1 * 1$ , por maioria de razão se encontraria um  $K_2$  em  $1 \leq K_2 * 1$ . Ou seja, em ambos os casos  $f(n)=1$ , logo  $O(1)$ .

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



À primeira vista, esta conclusão não parece nada aceitável, pois em termos absolutos um dos processos gasta 1000 vezes mais recursos do que o outro.

Mas não acha que os dois processos, com  $O(1)$ , reagem da mesma maneira, consumindo recursos de uma forma constante, independente da dimensão do problema?

Dos casos encontrados, apresente pelo menos um com comportamento  $O(1)$  e indique se esse comportamento é relativo ao tempo, ao espaço ou a ambos.



Imagine que tem uma turma com 50 estudantes e que quer saber se o primeiro estudante, aquele que está mais perto de si, pesa 60 quilos.

Entretanto a turma aumentou e agora, em vez de 50, tem 100 estudantes, e continua a querer saber a mesma coisa sobre o primeiro estudante.

Que tipo de comportamento tem este processo? Justifique.

#### Ordem de complexidade linear

- Se  $R(n)$  tiver um crescimento linear com  $n$ , a ordem de crescimento diz-se linear e representa-se por  $O(n)$ . Por exemplo, se  $R(n) = 1000 * n$ , então  $R(n) \leq K * n$ , e  $O(n)$ .

Aplicando o que se acaba de apresentar às soluções recursiva e iterativa do cálculo do factorial, podemos resumir.

- factorial, com processos recursivos: Tempo:  $O(n)$ ; Espaço:  $O(n)$
- factorial-novo, com processos iterativos: Tempo:  $O(n)$ ; Espaço:  $O(1)$ .



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECUSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



É de prever que `factorial` esgotará a memória disponível acima de um valor de  $n$ , situação que nunca ocorrerá com `factorial-novo`. Em termos de tempo, ambos apresentam um comportamento linear. Concorda com o que acaba de se afirmar? Justifique.



Imagine que tem uma turma de estudantes e que quer saber se algum dos estudantes pesa 60 quilos! Na melhor das hipóteses, o primeiro estudante pesa 60 quilos e o problema poder-se-ia considerar resolvido. Mas neste tipo de análise o mais seguro é contar com a pior hipótese. E qual é a pior hipótese? O tipo de comportamento deste processo é  $O(n)$ . Justifique.

### Ordem de complexidade exponencial

- Se  $R(n)$  tiver um crescimento  $M^n$ , a ordem de crescimento diz-se exponencial e representa-se por  $O(M^n)$ . Por exemplo, se  $R(n) = 1000 * M^n$ , então  $R(n) \leq K * M^n$ , e  $O(M^n)$ .

Se, por exemplo,  $R(n) = 2^n$ , então  $O(2^n)$ , caso em que os recursos duplicam quando  $n$  aumenta de um valor unitário. Para  $R(4) = 2^4 = 16$ , para  $R(5) = 2^5 = 32$ , para  $R(6) = 2^6 = 64$ , ...



Verifique o crescimento vertiginoso neste tipo de comportamento, certamente impraticável para valores de  $n$  elevados por conduzir ao esgotamento dos recursos computacionais... Imagine que, para  $n=1$ , um processo  $O(2^n)$  gasta 100 bytes de memória e consome 1 segundo de processador. Calcule em MB e em dias o consumo do mesmo processo para  $n=100$  e  $n=1000$ ...



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Relembrando os procedimento fib e fib-iter da série de Fibonacci:

- fib: Tempo:  $O(q^n)$ , com  $q \approx 1.6180$  (golden ratio); Espaço:  $O(n)$
- fib-iter: Tempo:  $O(n)$ ; Espaço:  $O(1)$ .

Uma solução com um comportamento exponencial será de evitar, pois tende a gastar recursos de uma forma exagerada quando a dimensão aumenta.

### Ordem de complexidade logarítmica

- Se  $R(n)$  tiver um crescimento  $\log_M n$ , a ordem de crescimento diz-se logarítmica e representa-se por  $O(\log_M n)$ . Por exemplo, se  $R(n) = 1000 * \log_M n$ , então  $R(n) \leq K * \log_M n$ , e  $O(\log_M n)$  .

Sendo  $R(n) = \log_2 n$ , então  $O(\log_2 n)$ , caso em que os recursos aumentam de um valor unitário quando n duplica.

Para  $R(8) = \log_2 8 = 3$ ,  $R(16) = \log_2 16 = 4$ ,  $R(32) = \log_2 32 = 5$ , ...



**Confirme, mesmo sem ter visto ainda qualquer exemplo, que  $O(\log M n)$  é melhor que a ordem de crescimento linear, muito melhor que a exponencial e apenas inferior à constante.**

**Pista: Esboce um gráfico com funções dos tipos referidos (exponencial, linear, logarítmica e constante), por exemplo para valores de n iguais a 0, 8, 16 e 32.**

Até agora não encontrámos qualquer situação que possa ilustrar a ordem de complexidade logarítmica  $O(\log_K n)$ . Com o objectivo de encontrar uma dessa situações, vamos por momentos esquecer que o Scheme disponibiliza o procedimento expt, cuja chamada (expt b n) devolve  $b^n$ .

Pretende-se desenvolver o procedimento potencia com 2 parâmetros, b e n, sendo n um inteiro positivo, para o qual a chamada (potencia b n) devolve  $b^n$ .

Surgem de imediato uma definição recursiva e o respectivo procedimento.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECUSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

- Caso base:  $b^0 = 1$
- Caso geral:  $b^n = b \cdot b^{n-1}$

```
(define potencia
  (lambda (b n)
    (if (zero? n)
        1
        (* b (potencia b (sub1 n))))))
```

O processo gerado por `potencia` não é iterativo mas sim recursivo.

Trata-se de um caso em que não existe recursividade em cauda, pois a última operação do procedimento é uma multiplicação, que fica suspensa até ao caso base, definido por  $n=0$ . Este processo, em termos de tempo e espaço, apresenta ordens de crescimento linear,  $O(n)$ .

Uma solução iterativa, `potencia-nova`, é obtida com um procedimento auxiliar, com um terceiro parâmetro, para acumular o resultado das sucessivas multiplicações.

```
(define potencia-nova
  (lambda (b n)
    (potencia-aux b n 1)))

(define potencia-aux
  (lambda (base contador acumulador)
    (if (zero? contador)
        acumulador
        (potencia-aux base (sub1 contador) (* base acumulador)))))
```

Nesta solução é evidente a recursividade em cauda. O processo gerado é iterativo, com  $O(1)$  em termos de espaço e  $O(n)$  em termos de tempo.



**Por que razão é dito que este processo é  $O(1)$  em espaço e  $O(n)$  em tempo?**



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECUSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Neste problema é ainda possível explorar uma outra ideia, em que a redução do expoente acontece muito mais rapidamente.

- Caso base:  $b^0 = 1$
- Casos gerais
  - Se n é par:  $b^n = (b^{n/2})^2$
  - Se n é ímpar:  $b^n = b \cdot b^{n-1}$

O processo gerado por potencia-melhorada é recursivo e, em termos de tempo e espaço, é  $O(\log_2 n)$ . Basta reflectir um pouco para se chegar a esta conclusão. Sempre que o expoente é dividido por 2, passando de n para n/2, o que corresponde a dividir por 2 a dimensão do problema, apenas se utiliza mais uma multiplicação.

```
(define potencia-melhorada
  (lambda (b n)
    (cond ((zero? n) 1)
          ((even? n) (quadrado (potencia-melhorada b (/ n 2))))
          (else
            (* b (potencia-melhorada b (sub1 n)))))))

(define quadrado
  (lambda (x)
    (* x x)))
```

### Exercício 5

Procure uma solução iterativa para potencia-melhorada, a designar por potencia-melhorada-nova, identificando e justificando a ordem de crescimento da solução encontrada.



Desenvolva e teste o procedimento potencia-melhorada-nova.

# **S C H E M E**

## **na descoberta da programação**

### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECUSIVIDADE**

R E 1 2 3

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

Já percebeu que a ordem de crescimento só nos permite caracterizar, de forma aproximada e não absoluta, o consumo de recursos computacionais em função da dimensão do problema. Dá-nos, no entanto, uma ideia suficientemente clara do comportamento dos processos gerados.

Para finalizar este tema, apresenta-se mais um exemplo em que se torna evidente o carácter aproximado deste tipo de medida. Para uma situação hipotética em que os recursos consumidos fossem especificados por  $R(n) = 15 * n^7 + 30 * n^6$ , a ordem de crescimento respectiva seria  $O(n^7)$ .

Sendo  $R(n) = 15 * n^7 + 30 * n^6$ , então será aproximadamente  $R(n) = 15 * n^7$ .

E agora  $R(n) = 15 * n^7 \leq K * n^7$  de onde resulta  $O(n^7)$ .

Como curiosidade, o erro que se comete ao fazer a aproximação  $R(n) = 15 * n^7$  é de cerca de 4% para  $n = 50$ , caindo para 2% para  $n = 100$ .



**Tente confirmar estes erros.**

**Se não foi capaz de confirmar os erros, tem agora uma pista.**

**Considerando apenas o termo de maior grau,  $R(n) = 15 * n^7$ , o erro cometido seria  $(15 * n^7 + 30 * n^6 - 15 * n^7) / 15 * n^7$  ou seja,  $30 * n^6 / 15 * n^7 = 2/n$ .**

**Assim, para  $n = 50$ , o erro seria de  $2/50 = 4\%$  e para  $n = 100$ ,  $2/100 = 2\%$ .**

Uma maneira prática de chegar à ordem de crescimento de uma situação é aplicar, à expressão dos recursos consumidos, as seguintes simplificações:

- ignorar os termos constantes;
- ignorar os termos que se tornam menos importantes quando  $n$  cresce muito.

Se os recursos consumidos forem  $R(n) = n - 1$ , a constante  $-1$  é ignorada e resulta  $O(n)$ .

Mas se forem  $R(n) = 5 * n^7 + 500 * n^6 - 56$ , as constantes 5, 500 e -56 são ignoradas, o termo  $n^6$  torna-se menos importante que  $n^7$  quando  $n$  cresce e resulta  $O(n^7)$ .



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Imagine que tem uma turma de estudantes e que quer saber se algum estudante pesa o mesmo que outro. Na pior hipótese... Testa o primeiro com todos os outros. Depois exclui o primeiro e testa o segundo com os restantes...

O tipo de comportamento deste processo é  $O(n^2)$ .

Trata-se de um comportamento quadrático. Justifique.



Apresente um gráfico com funções de todos os tipos referidos (exponencial, linear, logarítmica e constante), incluindo agora também a função quadrática para, por exemplo, valores de  $n$  iguais a 0, 8, 16 e 32. Inclua também um valor muito mais elevado de  $n$  (por ex., 1024), mesmo que surjam dificuldades ao desenhar o valor de algumas das funções.

Aproveite para verificar no gráfico o que ocorre para os valores de  $n$  muito elevados, aqueles que normalmente nos preocupam em termos da análise da ordem de crescimento...



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Módulo 2.3 - Projecto - Jogo "O computador-adivinho"

Tendo por base um jogo, o computador-adivinho, faz-se uma abordagem de-cima-para-baixo (*top-down*), que vai do geral para o pormenor. As várias fases de um projecto são tidas em consideração, nomeadamente a especificação do problema, a ideia de solução, o algoritmo, a codificação e o teste.

Um projecto, mesmo que medianamente complexo, pode envolver um grande número de procedimentos, a maior parte deles a funcionar como auxiliares de outros. Só gerir os nomes, simplesmente para não os repetir, pode revelar-se complicado. Uma solução passa por esconder os pormenores internos dos procedimentos, constituindo-os como verdadeiros blocos ou caixas-pretas. Escondem-se os nomes dos parâmetros (isso já acontece por omissão), os nomes das variáveis criadas localmente e também os nomes dos procedimentos auxiliares que, para esse efeito, são colocados no interior dos procedimentos que deles necessitam.

Neste módulo encontra duas partes. A primeira parte aborda os mecanismos do Scheme para criar procedimentos como blocos ou caixas-pretas. A segunda parte tem como objectivo o desenvolvimento do projecto de um jogo, o computador-adivinho, em que o computador "adivinha" o número em que pensámos.

### Palavras-Chave

Bloco ou caixa-preta, procedimentos mutuamente recursivos, abordagem de-cima-para-baixo (*top-down*), abordagem de-baixo-para-cima (*bottom-up*), especificação do problema, ideia de solução, algoritmo, codificação, teste.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECUSIVIDADE

R E 1 2 3

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

### Um procedimento é um bloco ou uma caixa-preta

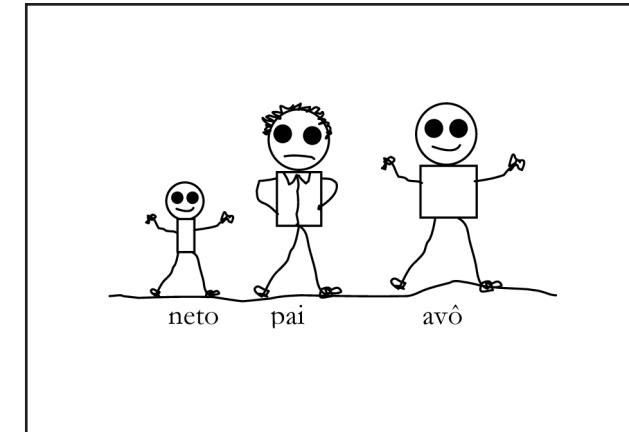
Quando se referem os procedimentos como blocos ou caixas-pretas, o que se pretende é pôr em destaque que estas entidades podem ser utilizadas conhecendo apenas o que fazem, sem ser necessário saber-se como o fazem. O conhecimento do que se passa no interior de cada um desses blocos é, nestes casos, perfeitamente dispensável. De facto, o que interessa conhecer sobre um procedimento para o utilizar é a sua interface, ou seja, o seu nome, os parâmetros que tem e aquilo que devolve como resultado.



Se conhecer e se sentir confortável com a utilização das formas especiais `let`, `letrec` e `let*` do Scheme, e com a abordagem de-cima-para-baixo (top-down), pode saltar esta secção, que representa cerca de metade do presente módulo, e passar de imediato para uma introdução à abordagem de-cima-para-baixo (top-down), ou, até mesmo, para o início do projecto do jogo, o computador-adivinho.

No exemplo que se segue, temos o neto, o pai e o avô, conhecemos os respectivos pesos e queremos saber o peso total dos três e a percentagem do peso de cada um em relação ao peso total. Esta tarefa vai ficar a cargo de um procedimento com o nome `neto-pai-avo`, com 3 parâmetros relativos ao peso de cada elemento da família, neto, pai e avo, por esta ordem.

```
> (neto-pai-avo 25.0 80 95)
peso total: 200.0
neto: 12.5%
pai : 40.0%
avo : 47.5%
> (neto-pai-avo 25.0 85 90)
peso total: 200.0
neto: 12.5%
pai : 42.5%
avo : 45.0%
>
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Apresenta-se uma solução possível para o procedimento neto-pai-avo, na qual é visível a criação de uma variável local com a designação total, que evita a repetição do cálculo deste valor.

```
(define neto-pai-avo
  (lambda (neto pai avo)
    (let ((total (+ neto pai avo)))
      (display "peso total: ")
      (display total)
      (newline)
      (display "neto: ")
      (display (* 100 (/ neto total)))
      (display "%")
      (newline)
      (display "pai : ")
      (display (* 100 (/ pai total)))
      (display "%")
      (newline)
      (display "avo : ")
      (display (* 100 (/ avo total)))
      (display "%"))))
```



O conhecimento da existência de uma variável interna ou de qualquer outro pormenor interno de um procedimento será importante para a sua utilização?

O procedimento neto-pai-avo comporta-se como uma verdadeira caixa-preta, pois não são visíveis a partir do seu exterior nenhum dos nomes nele definidos, nem os parâmetros nem a variável local total. O utilizador não precisa de conhecer mais nada para além do seu nome, parâmetros e o que devolve.

Se tentar estruturar a solução apresentada, identifica certamente uma tarefa que se repete três vezes, e que corresponde ao cálculo e visualização da informação de cada um dos três elementos da família. Aproveitando este facto, apresenta-se uma nova implementação, neto-pai-avo-1, que utiliza o procedimento auxiliar visu-percentagem, para calcular e visualizar a percentagem do peso de um familiar em relação ao peso total dos 3 familiares.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECUSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(define neto-pai-avo-1
  (lambda (neto pai avo)
    (let ((total (+ neto pai avo)))
      (display "peso total: ")
      (display total)
      (visu-percentagem "neto" neto total)
      (visu-percentagem "pai " pai total)
      (visu-percentagem "avo " avo total))))
```

```
(define visu-percentagem
  (lambda (nome peso total)
    (newline)
    (display nome)
    (display ": ")
    (display (* 100 (/ peso total)))
    (display "%")))
```



Teste os procedimentos neto-pai-avo e neto-pai-avo-1.



**Se analisar o procedimento neto-pai-avo-1 concluirá que não se apresenta como uma verdadeira caixa-preta.  
Aponte uma razão para esta afirmação.**

A solução surge com um procedimento auxiliar que lhe é exterior e assim neto-pai-avo-1 deixa de ser uma caixa-preta. Isto impede, por exemplo, que o nome visu-percentagem seja utilizado com outra finalidade, que não a de ser o procedimento auxiliar de neto-pai-avo-1. Esta preocupação é perfeitamente irrelevante na presente situação, mas poderá deixar de o ser no desenvolvimento de grandes programas, onde a gestão cuidada dos nomes utilizados é um ponto fundamental.



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECUSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Em Scheme, é possível definir procedimentos locais dentro de outros procedimentos, e é esta facilidade que agora se explora.

Para além do nome de variáveis, é ainda possível definir com `let` procedimentos não recursivos no corpo de outros procedimentos, os quais terão também um campo de acção limitado ao corpo do `let` onde forem definidos. O procedimento `neto-pai-avo-2` define localmente o procedimento `visu` que substitui `visu-percentagem`.

```
(define neto-pai-avo-2
  (lambda (neto pai avo)
    (let ((total (+ neto pai avo)))
      (let ((visu
            (lambda (nome peso)
              (newline)
              (display nome)
              (display ": ")
              (display (* 100 (/ peso total)))
              (display "%"))))
        ;
        (display "peso total: ")
        (display total)
        (visu "neto" neto)
        (visu "pai " pai)
        (visu "avo " avo)))))
```



Teste o procedimento `neto-pai-avo-2`.



A importância da gestão dos nomes utilizados faz com que muitos dos sistemas de desenvolvimento de programas, como também acontece com o DrScheme, disponibilizem mecanismos que permitem criar módulos ou unidades de código, nos quais é possível definir quais são as entidades visíveis do exterior. É que nem todas as linguagens permitem esconder procedimentos auxiliares dentro de outros procedimentos, prática que, por outro lado, não é muito recomendável por tornar os procedimentos mais longos e de leitura mais difícil.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Uma vantagem do procedimento `neto-pai-avo-2` é o facto de o nome `visu` só ser reconhecido no corpo de `let`, podendo ser utilizado fora deste ambiente para designar outra entidade qualquer.  
Consegue apontar uma situação em que esta vantagem se poderia transformar em desvantagem?

Vejamos ainda outra vantagem deste tipo de construção e que foi também aproveitada em `neto-pai-avo-2`.

Explorando o facto da variável local `total` também ser acessível no corpo de `visu`, não foi necessário passá-la como argumento, como acontecia em `visu-percentagem`. Assim, a chamada de `visu` torna-se menos exigente em termos de recursos computacionais e, torna-se mais eficiente, pois utiliza um menor número de argumentos.

No corpo do procedimento `visu`, quando se utiliza a variável `total`, e sendo que esta não é conhecida localmente, recorre-se ao ambiente imediatamente acima. Este nome não está definido no procedimento `visu` e diz-se, por isso, um nome livre neste ambiente. Se no ambiente acima `total` continuasse a ser um nome livre (o que não acontece), a procura prosseguiria, tendo como limite o ambiente global do Scheme.

A definição de procedimentos dentro de outros procedimentos, através da forma especial `let`, reduz-se a procedimentos não recursivos.



Aponte uma razão para justificar a impossibilidade de definir um procedimento recursivo com a forma `let`.

Com `letrec` esta limitação é ultrapassada. As formas especiais `let` e `letrec` são apresentadas uma ao lado da outra, pondo-se em evidência a grande semelhança de sintaxe entre elas.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Na forma `let`, os identificadores `nome-1`, `nome-2`, ... são apenas reconhecidos no corpo de `let` e associados aos valores das expressões respectivas.

```
(let ( nome-1 expressão-1
      nome-2 expressão-2
      ...
      )
  corpo-de-let )
```

```
(letrec ( nome-1 expressão-1
          nome-2 expressão-2
          ...
          )
  corpo-de-letrec )
```

A forma `letrec`, como acontece com `let`, devolve o valor da última expressão situada no seu corpo. Todavia, na forma `letrec`, os identificadores definidos podem ser associados a procedimentos recursivos.

Para ilustrar a utilização de `letrec`, começamos por um exemplo em que o procedimento pretendido e um procedimento auxiliar são ambos definidos no ambiente global do Scheme.

Pretende-se um procedimento recursivo que toma um argumento `n`, inteiro e positivo, e gera um processo iterativo para calcular  $n + (n-1) + \dots + 1 + 0$ .



Para esta situação identifique o caso base, a operação de redução e o caso geral, antes de analisar a solução que se apresenta de seguida.

```
(define soma-sequencia-iter ; procedimento para esconder
  (lambda (numero) ; os 2 parâmetros de soma-sequencia-aux
    (soma-sequencia-aux numero 0))) ; aqui o acumulador começa em 0

(define soma-sequencia-aux
  (lambda (numero acumulador)
    (if (zero? numero)
        acumulador
        (soma-sequencia-aux (sub1 numero)
                            (+ numero acumulador)))))
```



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

R E 1 2 3

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
> (soma-sequencia-iter 0)
0
> (soma-sequencia-iter 1)
1
> (soma-sequencia-iter 2)
3
>
```



Teste o procedimento soma-sequencia-iter.

Agora, através de letrec, apenas o procedimento soma-sequencia-iter-1 é definido no ambiente global do Scheme, uma vez que o procedimento auxiliar recursivo passa a ser definido no seu interior.

```
(define soma-sequencia-iter-1
  (lambda (numero)
    ;
    ; ----- definições locais de soma-sequencia-iter-1
    ;
    (letrec ((aux
              (lambda (num acumulador)
                (if (zero? num)
                    acumulador
                    (aux (sub1 num) (+ num acumulador))))))
      ;
      ; ----- corpo do procedimento soma-sequencia-iter-1
      ;
      (aux numero 0)))
    )
  )

> (soma-sequencia-iter-1 4)
10
```



Teste o procedimento soma-sequencia-iter-1.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



No procedimento **soma-sequencia-iter-1**, substitua **letrec** por **let**.

Diga e justifique o que acontece nas seguintes situações:

- 1- (**soma-sequencia-iter-1 0**)
- 2- (**soma-sequencia-iter-1 1**)



No procedimento **soma-sequencia-iter-1**, contrariamente ao que aconteceu no procedimento **soma-sequencia-iter**, não houve grande preocupação na escolha do nome do procedimento auxiliar, designado por **aux**.

Aponte uma explicação para esta opção.

Para facilitar a leitura de procedimentos que contenham outros procedimentos definidos localmente, sugere-se que:

- 1- analise os procedimentos auxiliares locais para entender bem a respectiva funcionalidade
- 2- seguidamente, foque a atenção apenas na análise do corpo do procedimento principal, passando por cima das definições locais.

### Exercício 1

O procedimento **piramide** tem um único parâmetro, **base**, que deverá ser um inteiro positivo ímpar, e que representa o comprimento da base de uma "pirâmide".

> (**piramide 5**)

\*  
\*\*\*  
\*\*\*\*\*

> (**piramide 6**)

Nao e' impar!!!



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Escreva em Scheme o procedimento `piramide`, definindo os procedimentos auxiliares no interior dele, e classifique a solução encontrada em termos de ordem de crescimento em relação ao tempo e ao espaço.



Desenvolva e teste o procedimento `piramide`.

### Procedimento que chama outro procedimento e que é por sua vez chamado por este!

Um procedimento que chama outro procedimento e que é por sua vez chamado por este dizem-se mutuamente recursivos.

Vejamos um exemplo. Um procedimento determina se um número é ímpar, baseando-se na seguinte ideia: a pergunta `número é ímpar?` deverá ter a mesma resposta que a pergunta `número - 1 é par?`. A questão, colocada desta forma, reduz o problema, na direcção do caso base, considerando que o caso base é `número = 0`, a que corresponde a resposta `#f`, pois zero não é ímpar.

```
(define numero-impar?
  (lambda (numero)
    (if (zero? numero)
        #f
        (numero-par? (sub1 numero)))))
```

Por um raciocínio análogo, chegar-se-ia ao procedimento `numero-par?`.

```
(define numero-par?
  (lambda (numero)
    (if (zero? numero)
        #t
        (numero-impar? (sub1 numero)))))

> (numero-impar? 3)
#t
> (numero-par? 3)
#f
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECUSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

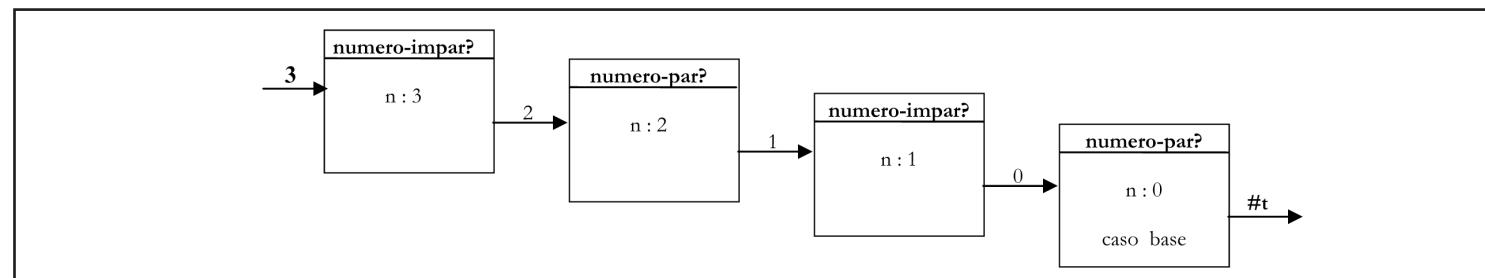
### ANEXOS



O Scheme disponibiliza os procedimentos primitivos `odd?` e `even?`, predicados que indicam se o argumento é ímpar ou par, respectivamente ([Anexo A](#)) - em Processamento Numérico.

Este exemplo, com os procedimentos `numero-impar?` e `numero-par?`, é, por isso, um exercício meramente académico.

Através da chamada `(numero-par? 3)`, sob a forma gráfica, verifique que estes dois procedimentos, mutuamente recursivos, geram processos iterativos (nenhum deles fica suspenso, à espera de qualquer resposta) e apresentam  $O(n)$  em termos de tempo e  $O(1)$  em relação à memória utilizada.



Verifique a seguir que o procedimento `impar?` funciona como uma caixa-preta, baseado nos dois procedimentos mutuamente recursivos, `numero-impar?` e `numero-par?`.

O procedimento `impar?` utiliza dois procedimentos auxiliares, mutuamente recursivos, definidos localmente com `letrec`.



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

R E 1 2 3

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
(define impar?
  (lambda (numero)
  ;
  ; ----- definições locais de impar?
  ;
  (letrec ((numero-impar?
            (lambda (num)
              (if (zero? num)
                  #f
                  (numero-par? (sub1 num)))))

  (numero-par?
    (lambda (num)
      (if (zero? num)
          #t
          (numero-impar? (sub1 num))))))

  ;
  ; ----- corpo do procedimento impar?
  ;
  (numero-impar? numero))))
```

```
> (impar? 40)
#f
> (impar? 41)
#t
```



Teste o procedimento **impar?**.



No procedimento **impar?** substitua **letrec** por **let**.

Diga e justifique o que acontece nas seguintes situações:

- 1- (**impar?** 0)
- 2- (**impar?** 1)



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

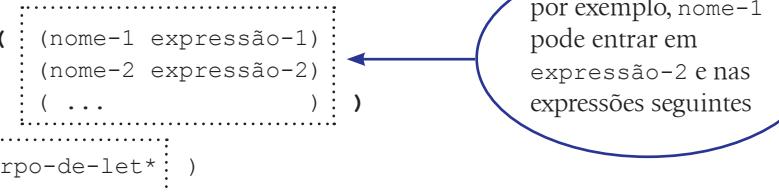
## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

O Scheme disponibiliza uma variante de `let`, a forma especial `let*`, que garante a definição sequencial dos identificadores `nome-1`, `nome-2`, ..., o que não acontece com `let`. Isto permite que um identificador acabado de definir integre a expressão de um identificador a definir.

A forma `let*` é muito semelhante à forma `let`:

```
(let* ( nome-1 expressão-1 )
      ( nome-2 expressão-2 )
      ( ...           ) )
      corpo-de-let* )
```



Parte-se do procedimento `neto-pai-avo` e recorre-se à forma especial `let*` para chegar à versão `neto-pai-avo-3`. Neste procedimento, observe que um identificador acabado de definir, `total`, pode ser imediatamente utilizado ainda no campo de definições de `let*`.

```
(define neto-pai-avo-3
  (lambda (neto pai avo)
    (let* ((total (+ neto pai avo))
           (perc-neto (* 100 (/ neto total)))
           (perc-pai (* 100 (/ pai total)))
           (perc-avo (* 100 (/ avo total))))
      ;
      (display "peso total: ")
      (display total)
      (newline)
      (display "neto: ")
      (display perc-neto)
      (display "%")
      (newline)
      (display "pai : ")
      (display perc-pai)
      (display "%")
      (newline)
      (display "avo : ")
      (display perc-avo)
      (display "%"))))
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Teste o procedimento `neto-pai-avo-3`.

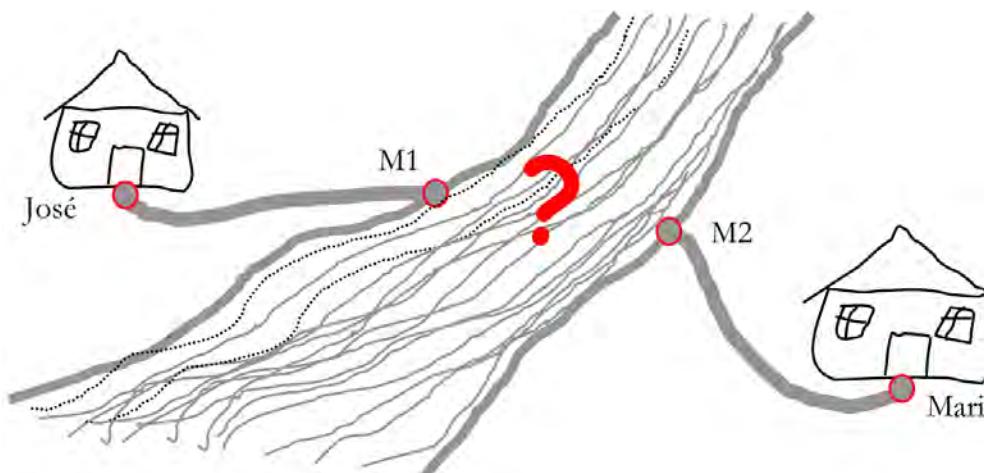
Depois de experimentar o procedimento, tente utilizar `letrec` em vez de `let*` e explique o que acontece.  
Experimente agora utilizar `let` em vez de `let*`. E agora, o que acontece?

### Atacar um problema numa abordagem de-cima-para-baixo

Desenvolver um programa para uma situação de complexidade um pouco superior à requerida pelos exercícios e exemplos apresentados surge como uma tentativa, embora académica, de aproximação a problemas reais que, nem sempre, se apresentam devidamente especificados.

É também uma oportunidade para delinejar uma forma sistemática de programar, designada por abordagem de-cima-para-baixo (*top-down*), em que um problema é dividido em sub-problemas mais simples. Os sub-problemas que ainda apresentem alguma complexidade são abordados isoladamente, também de-cima-para-baixo, e divididos, também eles, em sub-problemas cada vez mais simples.

Mas começemos com um exemplo para ilustrar a abordagem de-cima-para-baixo. Observe com atenção a figura.



### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECUSIVIDADE**

R E 1 2 3

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

O problema a resolver é fazer o percurso da casa do José até à casa da Maria, sendo que há um rio pelo meio. A parte do percurso terrestre é para fazer a pé e a parte do rio, entre as margens M1 e M2, é mais complicada e ainda não se sabe muito bem como a fazer. Talvez a nado, talvez de barco... Mas para já, sem entrar em pormenores, imagine, simplesmente, que tem alguma maneira de atravessar o rio.

#### Problema - Fazer o percurso da casa do José até à casa da Maria

- 1- ir a pé da casa do José até à margem M1
- 2- sub-problema - atravessar o rio de M1 para M2
- 3- ir a pé da margem M2 até à casa da Maria

O problema inicial é suposto estar resolvido e é chegada a altura de focar a atenção apenas no sub-problema da travessia do rio.

Depois de se reflectir um pouco, em vez de se atravessar a nado, optou-se por fazer a travessia de barco, pois até há barcos para alugar junto às margens M1 e M2.

#### Atravessar o rio de M1 para M2

1. alugar um barco
2. entrar no barco em M1
3. atravessar o rio de barco de M1 até M2
4. sair do barco em M2

Se agora juntar tudo, obtém um algoritmo para o problema inicial, no qual são nítidos dois níveis.

#### Problema - Ir da casa do José à casa da Maria

- 1- ir a pé da casa do José até à margem M1
- 2- sub-problema - atravessar o rio de M1 para M2
  1. alugar um barco
  2. entrar no barco em M1
  3. atravessar o rio de barco de M1 até M2
  4. sair do barco em M2
- 3- ir a pé da margem M2 até à casa da Maria



É esta a abordagem **de-cima-para-baixo**.

Justifique o nome desta abordagem.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Numa abordagem, a um problema, de-cima-para-baixo distingue-se normalmente um certo conjunto de fases:

#### 1. Especificação do problema

Procurar entender, com o maior rigor possível, o problema que é colocado, a fim de colmatar lacunas e ultrapassar eventuais dúvidas de uma especificação inicial incompleta.

Pode até acontecer que o programador, em diálogo com o "cliente", apresente propostas que venham completar ou até alterar a especificação inicial.

#### 2. Desenvolvimento manual de situações do problema

Desenvolver manualmente algumas situações do problema, apesar de surgirem casos em que esta fase não se aplique de uma forma simples.

Este esforço, para além de ajudar a entender melhor o problema, também fornece um conjunto de situações que servirão posteriormente para testar o programa.

#### 3. Ideia de solução

Procurar uma ideia de solução para o problema.

É, muito provavelmente, a fase mais criativa de todo o processo.

#### 4. Algoritmo

Escrever um algoritmo em torno da ideia de solução encontrada, descrevendo informalmente, em pseudo-código, os vários passos do programa.

O pseudo-código pode ser uma linguagem rudimentar, normalmente baseada em português ou inglês.

Nesta fase é importante identificar os passos mais complexos e evitar a tentação de os tratar em pormenor. Para isso, basta imaginar que esses passos mais complexos constituem sub-problemas do problema principal, que serão posteriormente resolvidos isoladamente, também segundo uma abordagem de-cima-para-baixo.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

### 5. Codificação em Scheme e Teste

Escrever o algoritmo numa linguagem que o computador entenda e depois testá-lo face aos casos conhecidos, obtidos no passo 2.

Contrariamente ao percurso seguido na fase anterior, é praticamente obrigatório seguir uma abordagem de-baixo-para-cima (*bottom-up*), começando por codificar e testar as partes correspondentes aos sub-problemas que não dependam de outras partes... e assim sucessivamente, de-baixo-para-cima , até se chegar ao topo, ao programa desejado.



Pense um pouco e indique por que razão o teste deve seguir a abordagem de-baixo-para-cima.

### 6. Aplicação da abordagem de-cima-para-baixo aos sub-problemas mais complexos

A cada um desses sub-problemas aplicar os passos de 1 a 6.

Mas agora o melhor é passar à aplicação da abordagem de-cima-para-baixo a um exemplo, o projecto do jogo o computador-adivinho.

### Projecto - o computador-adivinho

Pretende-se adaptar para computador um jogo que utiliza 5 cartões, cada um deles com números impressos.

Cartão 1:	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
Cartão 2:	2	3	6	7	10	11	14	15	18	19	22	23	26	27	30	31
Cartão 3:	4	5	6	7	12	13	14	15	20	21	22	23	28	29	30	31
Cartão 4:	8	9	10	11	12	13	14	15	24	25	26	27	28	29	30	31
Cartão 5:	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

É pedido ao jogador que pense num número de 0 a 31. Seguidamente, é perguntado ao jogador se o número em que pensou está no cartão 1, depois no cartão 2, e assim sucessivamente até ao cartão 5. No final, é dito ao jogador qual o número em que pensou!



INTRODUÇÃO
1 - O ESSENCIAL DO SCHEME
<b>2 - RECURSIVIDADE</b>
R E 1 2 3
3 - ABSTRACÇÃO DE DADOS
4 - PROCEDIMENTOS COMO OBJECTOS DE 1 <sup>a</sup> CLASSE
5 - ABSTRACÇÕES COM DADOS MUTÁVEIS
6 - SCHEME E OUTRAS TECNOLOGIAS
7-EXERCÍCIOS E PROJECTOS
ANEXOS



Mesmo sem cartões, pense num número entre 0 e 31. Pode ser, por exemplo, o 21.

Agora verifique em que cartões está o número em que pensou.

cartão 1 - Sim

cartão 2 - Não

cartão 3 - Sim

cartão 4 - Não

cartão 5 - Sim

Agora, associando os pesos 1, 2, 4, 8 e 16 aos cartões de 1 a 5, adicione os pesos dos cartões com a resposta

Sim.

$$1 + 4 + 16 = 21$$

Sim, é verdade, o resultado é o número em que tinha pensado!!!

Mas tente com outros números...Funciona sempre.

Tem alguma ideia para justificar este truque?

A sequência de 5 perguntas é uma forma subtil de obter do jogador o número em que pensou. Obviamente, não lhe é perguntado directamente o número em que pensou, pois espera-se que o programa se encarregue de o adivinhar.

Para entender bem o que se afirma convém conhecer ou relembrar a codificação de números na base 2. Verifique que a 1<sup>a</sup> pergunta é antecedida pela lista de todos os números, entre 0 e 31, com o bit 0 igual a 1, ou seja, todos os ímpares situados naquela gama de valores.

Com a resposta à 1<sup>a</sup> pergunta, o programa fica a conhecer se o número pensado tem o bit 0 igual a 1 (resposta Sim) ou igual a 0 (resposta Não).

Segue-se a 2<sup>a</sup> pergunta, antecedida pela lista de todos os números, entre 0 e 31, com o bit 1 igual a 1, e assim até à 5<sup>a</sup> pergunta.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Ou seja, no final das 5 perguntas o programa terá apenas de converter um número na base 2 para o valor correspondente na base 10, sabendo que ao bit 0 está associado o peso 1, ao bit 1 o peso 2, ao bit 2 o peso 4, ao bit 3 o peso 8 e ao bit 4 o peso 16.

Assim, se o número pensado tiver sido 13, as respostas serão: sim (pergunta 1 - peso 1), não (pergunta 2 - peso 2), sim (pergunta 3 - peso 4), sim (pergunta 4 - peso 8) e não (pergunta 5 - peso 16). Adicionando os pesos com resposta sim, surge  $1+4+8 = 13$  e o programa está em condições de "adivinar" o número pensado pelo jogador.

Para desenvolver um programa em Scheme que simule o jogo descrito, vamos seguir as fases indicadas para uma abordagem de-cima-para-baixo.

#### Primeiro nível da abordagem de-cima-para-baixo

computador-adivinho - Especificação do problema

Uma sessão possível com o jogo o computador-adivinho é seguidamente apresentada.

> (adivinar)

Pense num numero entre 0 e 31!

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31

P1- O numero pensado esta' no conjunto (1=sim, 0=nao)? 1

2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31

P2- O numero pensado esta' no conjunto (1=sim, 0=nao)? 0

4 5 6 7 12 13 14 15 20 21 22 23 28 29 30 31

P3- O numero pensado esta' no conjunto (1=sim, 0=nao)? 1

8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31

P4- O numero pensado esta' no conjunto (1=sim, 0=nao)? 1

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

P5- O numero pensado esta' no conjunto (1=sim, 0=nao)? 0

O numero pensado foi: 13

>

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Não parece haver dúvidas sobre o modo de funcionamento do jogo e a especificação pode considerar-se completa. Existirão, eventualmente algumas dúvidas sobre como reagirá o programa, quando o jogador é convidado a responder sim/não, através dos valores 1/0 e, intencionalmente ou por engano, responde outra coisa qualquer. Vamos definir que, caso a resposta não se restrinja a uma das duas hipóteses (ou 1 ou 0), o programa repetirá a pergunta.

```
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31  
P1- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 2
```

```
P1- O numero pensado esta' no conjunto (1=sim, 0=nao) ?
```

computador-adivinho - Desenvolvimento manual de situações do problema

Esta fase é dispensável, pois pode ser aproveitado o que já foi adiantado na fase anterior de especificação.

computador-adivinho - Ideia de solução para o problema

Sem cair na tentação de entrar em pormenores, o problema decompõe-se:

- 1- na visualização da mensagem inicial;
- 2- na sequência de 5 perguntas que, de uma forma subtil, acaba por obter do jogador o número em que ele pensou;
- 3- na visualização da mensagem final que inclui o número "adivinhado".

computador-adivinho - Algoritmo para o problema

A ideia sugere um algoritmo de 3 passos.

- 1- Mensagem inicial;
- 2- Sequência de perguntas e cálculo do número;
- 3- Mensagem com a resposta.

Dos 3 passos do algoritmo, podemos avançar, sem grandes dúvidas, que o passo 2 é de longe o mais complexo, exigindo assim um aprofundamento adicional.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECUSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Para o problema global, podemos imaginar a existência de um procedimento principal, sem parâmetros, com a designação adivinhar. Este procedimento:

- 1- visualiza a mensagem inicial;
- 2- recorre ao procedimento perguntas, sem parâmetros, que faz a sequência de cinco perguntas e processa as respostas para calcular o número;
- 3- visualiza a mensagem com a resposta.

computador-adivinho - Codificação e Teste

Face ao que já foi apresentado, é possível desde já codificar em Scheme o procedimento adivinhar.

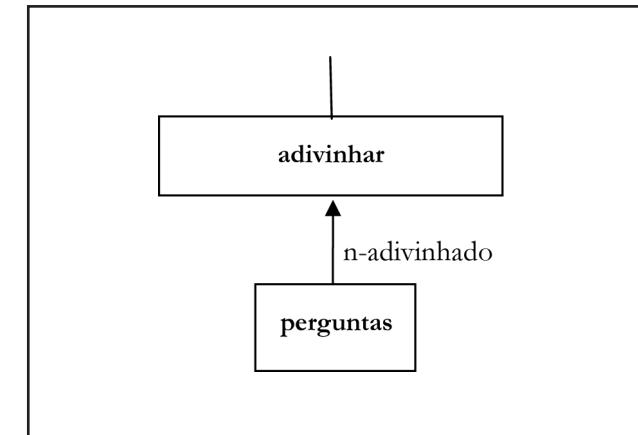
```
(define adivinhar
  (lambda ()
    ; --- visualização da mensagem inicial
    (newline)
    (display "Pense num numero entre 0 e 31!")
    (newline)

    ; --- perguntas e determinação do número pensado
    (let ((numero-adivinhado (perguntas)))

      ; --- visualização da resposta
      (newline)
      (display "O numero pensado foi: ")
      (display numero-adivinhado))))
```



Este seria o momento adequado para testar o procedimento adivinhar.  
Por que razão não é possível fazê-lo já?



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

O teste do procedimento `adivinar` fica impedido por não haver, para já, solução para o procedimento `perguntas`, com o qual se resolve o passo 2 do algoritmo do problema computador-adivinho.



Mas se quiser testar a parte do procedimento `adivinar` já concluída, equivalente à visualização das mensagens inicial e final, pode escrever uma versão provisória do procedimento `perguntas`, que apenas devolve um número, sem fazer qualquer pergunta.

```
; procedimento apenas para efeitos de teste
(define perguntas
  (lambda () ; devolve sempre o valor 7!!!
    7))
```

Teste (parcialmente) o procedimento `adivinar`.



**O que obteve ao experimentar `adivinar` desta maneira simplificada?  
Vê alguma utilidade nesta experiência?**

A partir deste momento, o teste do procedimento `adivinar` fica suspenso para podermos focar a atenção apenas no que passará a ser o nosso problema, sequência de perguntas e cálculo do número, como se tratasse de um problema novo...



**Exagerando um pouco a importância do assunto, este novo problema até poderia ser desenvolvido por outra equipa de programadores, à qual teria que passar a sua especificação.  
Mais tarde, a solução que viesse a receber seria integrada no seu código, neste caso, no procedimento `adivinar`.  
Precisa então, de especificar este novo problema designado por sequência de perguntas e cálculo do número.**



### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECURSIVIDADE**

R E 1 2 3

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

## **Segundo nível da abordagem de-cima-para-baixo**

sequência de perguntas e cálculo do número - Especificação do problema

Da sessão com o jogo computador-adivinho foi isolada a parte associada à sequência de perguntas e cálculo do número.

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31  
P1- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 1

2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31  
P2- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 0

4 5 6 7 12 13 14 15 20 21 22 23 28 29 30 31  
P3- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 1

8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31  
P4- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 1

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31  
P5- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 0

Trata-se de uma sequência de 5 perguntas e outras tantas respostas. Com a sequência de respostas verificada será devolvido o valor 13.

sequência de perguntas e cálculo do número - Desenvolvimento manual de situações do problema  
Esta fase é dispensável, pois pode ser aproveitado o que foi adiantado na fase anterior de especificação.

sequência de perguntas e cálculo do número - Ideia de solução para o problema

O problema sequência de perguntas e cálculo do número pode ser decomposto num ciclo de 5 perguntas, resultando de cada uma delas uma resposta 1 ou 0. Estas respostas deverão ser sucessivamente multiplicadas pelos pesos 1, 2, 4, 8 e 16 e depois adicionadas.

sequência de perguntas e cálculo do número - Algoritmo para o problema  
A ideia sugere um algoritmo de 5 passos.

- 1- Faz pergunta 1 e recolhe a respectiva resposta
- 2- Faz pergunta 2 e recolhe a respectiva resposta

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

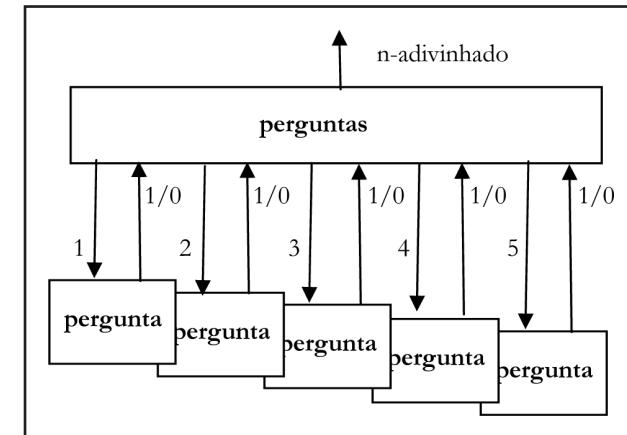
## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

- 3- Faz pergunta 3 e recolhe a respectiva resposta
- 4- Faz pergunta 4 e recolhe a respectiva resposta
- 5- Faz pergunta 5 e recolhe a respectiva resposta

Para este problema foi imaginado um procedimento com o nome perguntas, sem parâmetros. Este procedimento recorre ao procedimento pergunta, com um parâmetro que é o número da pergunta (de 1 a 5), para visualizar a lista dos números relativos à pergunta e recolher a resposta, e esta é certamente a parte mais complexa do problema agora em questão.



Não esquecer que só se aceitará esta resposta se ela for 1 ou 0.

sequência de perguntas e cálculo do número - Codificação e Teste

Com a ajuda do procedimento auxiliar pergunta já é possível escrever o procedimento perguntas.

```
(define perguntas
  (lambda ()
    (letrec ((ciclo
              (lambda (n-p)
                (if (> n-p 5) ; multiplica as respostas 0 ou 1
                    0           ; pelos pesos 1, 2, 4, 8 e 16 ...
                    (+ (* (pergunta n-p)           ; resposta 0 ou 1
                           (expt 2 (sub1 n-p))) ; peso da pergunta (1, 2, 4, 8 e 16)
                           (ciclo (add1 n-p))))))
                  ; --- corpo do procedimento
                  (ciclo 1))))
```

O teste do procedimento perguntas fica por agora impedido por não haver solução para o procedimento pergunta.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Imagine uma solução para testar a parte do procedimento perguntas já concluída, que poderia ser a seguinte versão provisória do procedimento perguntas:

```
; procedimento apenas para efeitos de teste
(define perguntas
  (lambda (n) ; devolve sempre o valor 1!!!
    1))
```

Teste (parcialmente) o procedimento perguntas.



O que obteve ao experimentar o procedimento perguntas desta maneira simplificada?

A partir deste momento, o problema sequência de perguntas e cálculo do número fica em suspenso para podermos focar a atenção apenas no que passará a ser o problema, faz pergunta, como se tratasse de um problema novo...

### Terceiro nível da abordagem de-cima-para-baixo

faz pergunta - Especificação do problema

Uma pergunta é antecedida pela visualização dos números associados a um determinado cartão. Segue-se a leitura da resposta que, necessariamente, só será aceite quando for 1 ou 0.

```
> (perguntas 1)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31
P1- O numero pensado esta' no conjunto (1=sim, 0=nao)? 2

P1- O numero pensado esta' no conjunto (1=sim, 0=nao)?
```



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECUSIVIDADE

R E 1 2 3

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

Os cartões são 5 e as perguntas também. No exemplo estamos perante o diálogo relativo ao cartão 1.

faz pergunta - Desenvolvimento manual de situações do problema

Esta fase é dispensável tendo em consideração o que já foi adiantado na fase anterior de especificação.

faz pergunta - Ideia de solução para o problema

O problema faz pergunta pode ser decomposto numa sequência de passos, começando por 1- visualização dos números de um cartão; seguido de 2- colocação da pergunta; 3- leitura e verificação da validade da resposta, com eventual repetição da pergunta e, finalmente, 4- devolução da resposta com a indicação 0 ou 1, significando ausência ou presença do número no cartão.

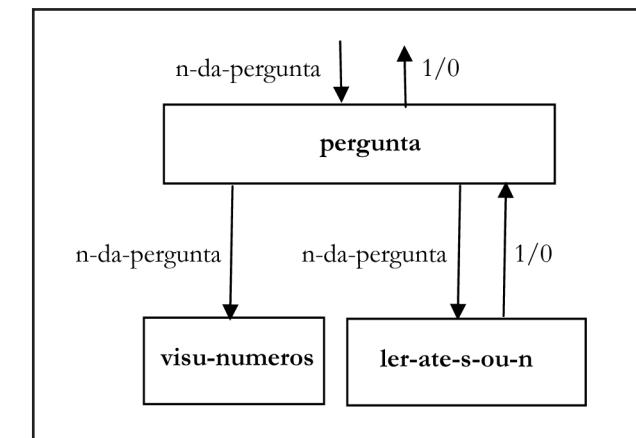
Como a expressão desta ideia parece bastante pormenorizada, opta-se por uma descrição mais compacta, do tipo:  
1- visualização dos números de um cartão (entre 5) e 2- leitura de uma resposta até à obtenção de um sim ou de um não.

faz pergunta - Algoritmo para o problema

Face a um número de pergunta (de 1 a 5), para o problema faz pergunta a ideia de solução sugere um algoritmo com 2 passos.

- 1- Visualiza os números do cartão ligado à pergunta especificada por um número entre 1 e 5
- 2- Lê até obter uma resposta sim ou não

No procedimento pergunta, com um parâmetro correspondente ao número da pergunta (de 1 a 5), identificam-se 2 procedimentos auxiliares, visu-numeros e ler-ate-s-ou-n, também eles com um parâmetro correspondente ao número da pergunta. O primeiro visualiza os números associados à pergunta e o segundo coloca a pergunta e lê até que a resposta seja sim (1) ou não (0).



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

R E 1 2 3

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

faz pergunta - Codificação e Teste

Com a ajuda dos procedimentos auxiliares visu-numeros e ler-ate-s-ou-n, já é possível escrever o procedimento pergunta.

```
(define pergunta
  (lambda (n-pergunta)
    (visu-numeros n-pergunta) ; visualiza os números da pergunta.
    (ler-ate-s-ou-n n-pergunta))) ; lê até obter uma resposta 1 ou 0.
```

O procedimento auxiliar ler-ate-s-ou-n é codificado e testado de imediato, o mesmo não acontecendo com visu-numeros, que parece exigir mais um nível de pormenorização.

Segue-se a codificação e teste do procedimento ler-ate-s-ou-n, que só termina com a leitura de 0 ou 1.

```
(define ler-ate-s-ou-n
  (lambda (n-perg)
    (display "P")
    (display n-perg)
    (display "- O numero pensado esta' no conjunto (1=sim, 0=nao)? ")
    (let ((num-lido (read)))
      (if (or (equal? num-lido 1)
              (equal? num-lido 0))
          num-lido ; caso em que a resposta ou é 1 ou 0.
          (begin
            (newline) ; caso em que a resposta nem é 1 nem 0.
            (ler-ate-s-ou-n n-perg))))))
```

```
> (ler-ate-s-ou-n 2)
P2- O numero pensado esta' no conjunto (1=sim, 0=nao)? 3 ←
P2- O numero pensado esta' no conjunto (1=sim, 0=nao)? 1
1
> (ler-ate-s-ou-n 2)
P2- O numero pensado esta' no conjunto (1=sim, 0=nao)? 0
0
```

resposta a não  
considerar

O teste do procedimento pergunta fica impedido por não haver, para já, solução para o procedimento visu-numeros.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Teste o procedimento `ler-ate-s-ou-n`.

Teste (parcialmente) o procedimento `pergunta`, imaginando e desenvolvendo uma versão provisória do procedimento `visu-numeros`.

A partir deste momento, deixamos em suspenso o problema `faz pergunta` para focarmos a nossa atenção no que passará a ser o problema, visualiza números do cartão, como se tratasse de um problema novo...

### Quarto nível da abordagem de-cima-para-baixo

visualiza números do cartão - Especificação do problema

Como exemplo, para os cartões 1 e 2, os números a visualizar são os indicados.

```
> (visu-numeros 1)  
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31  
> (visu-numeros 2)  
2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31
```

visualiza números do cartão - Desenvolvimento manual de situações do problema

Esta fase é dispensável tendo em consideração o que já foi adiantado na fase anterior de especificação.

visualiza números do cartão - Ideia de solução para o problema

Conforme o número de um cartão (de 1 a 5)...

- com o cartão 1 são visualizados todos os números de 0 a 31 com o bit 0 igual a 1.
- com o cartão 2 são visualizados todos os números de 0 a 31 com o bit 1 igual a 1.
- com o cartão 3 são visualizados todos os números de 0 a 31 com o bit 2 igual a 1.
- com o cartão 4 são visualizados todos os números de 0 a 31 com o bit 3 igual a 1.
- com o cartão 5 são visualizados todos os números de 0 a 31 com o bit 4 igual a 1.

Assim, dado o número  $x$  de um cartão, percorrem-se os números de 0 a 31, visualizando os que tiverem o bit  $x-1$  igual 1.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

R E 1 2 3

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

visualiza números do cartão - Algoritmo para o problema

Face a um número de pergunta ou cartão (de 1 a 5), para o problema visualiza números do cartão identifica-se o algoritmo que agora se indica.

Para o cartão x,

- 1- Percorrer os números de 0 a 31

Se o número corrente tiver o bit  $x-1$  igual a 1 será visualizado

visualiza números do cartão - Codificação e Teste

```
(define visu-numeros
  (lambda (nperg)
    (letrec ((tem-bit-a-1      ; numero tem o bit em análise igual a 1?
              (lambda (numero ordem-do-bit)
                (odd? (quotient numero (expt 2 ordem-do-bit)))))

              (ciclo      ; percorre os números de 0 a 31...
              (lambda (numero)           (ciclo (add1 numero))))))
        (cond ((> numero 31)
               (newline))
              ((tem-bit-a-1 numero (sub1 nperg))
               (display numero)
               (display " "))
               (ciclo (add1 numero)))
              (else

               (newline)
               (ciclo 0)))))

  > (visu-numeros 1)

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31
> (visu-numeros 2)

2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31
```



Teste o procedimento visu-numeros.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECUSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

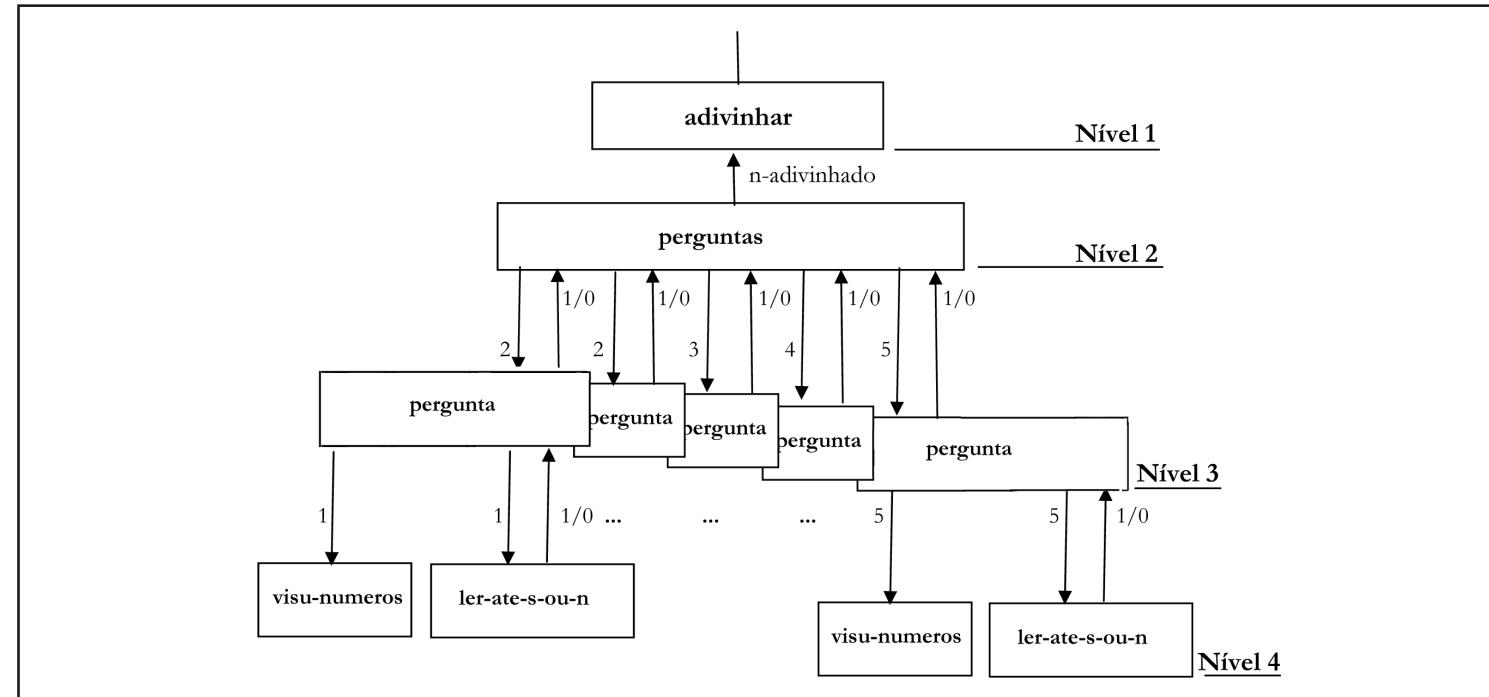
### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A partir deste momento, e numa abordagem de-baixo-para-cima, testam-se todos os procedimentos que ficaram por testar. Através da figura, verifica-se que ficaram por testar, no nível 3 - pergunta, no nível 2 - perguntas e no nível 1 - adivinhar.



### Teste no nível 3 - pergunta

```

> (pergunta 1)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31
P1- O numero pensado esta' no conjunto (1=sim, 0=nao)? 3
P1- O numero pensado esta' no conjunto (1=sim, 0=nao)? 1
  
```

```

1
> (pergunta 2)
  
```

```

2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31
P2- O numero pensado esta' no conjunto (1=sim, 0=nao)? 0
  
```

```

0
>
  
```



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

R E 1 2 3

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

## Teste no nível 2 - perguntas

> (perguntas)

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31

P1- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 1

2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31

P2- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 0

4 5 6 7 12 13 14 15 20 21 22 23 28 29 30 31

P3- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 1

8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31

P4- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 1

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

P5- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 0

13

>

## Teste no nível 1 - adivinhar

> (adivinhar)

Pense num numero entre 0 e 31!

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31

P1- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 1

2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31

P2- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 0

4 5 6 7 12 13 14 15 20 21 22 23 28 29 30 31

P3- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 1

8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31

P4- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 1

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

P5- O numero pensado esta' no conjunto (1=sim, 0=nao) ? 0

O numero pensado foi: 13

>



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

R E 1 2 3

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Testado o procedimento principal adivinhar, está agora em condições de demonstrar o jogo junto dos seus amigos.



Teste o jogo o computador-adivinho.

### Exercício 2

Com 5 cartões pode escolher um número entre 0 e 31, com 6 cartões, o número escolhido situa-se entre 0 e 63, já com 7 cartões a gama de valores vai de 0 a 127.

Tendo por base o projecto anterior, imagine agora que o procedimento adivinhar tem um parâmetro, numero-de-cartoes, que define com quantos cartões se pretende jogar. Faça todas as adaptações necessárias ao programa fornecido naquele projecto.



Adapte e teste o jogo o computador-adivinho.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Resumo dos módulos 3.1 a 3.4

Resumo dos assuntos abordados nos módulos

Módulo 3.1 - **Abstracção de dados para captar as características dos problemas**

Módulo 3.2 - **Colocar objectos em sequência**

Módulo 3.3 - **Um jogo para testar a memória**

Módulo 3.4 - **Utilizar e construir abstracções para reutilizar código**



### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

R E 1 2 3 4

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

## **Resumo dos assuntos abordados nos módulos 3.1 a 3.4**

Neste conjunto de módulos é mostrado o que deve ser feito quando os tipos de dados e respectivas operações básicas, oferecidos pelas linguagens de programação, não se adaptam convenientemente aos problemas que pretende resolver. Em situações deste tipo, recomenda-se que componha novos tipos de dados, criando o que se designa por abstracção de dados, a partir da qual procurará captar as características do problema a resolver.

### **Módulo 3.1**

Os tipos de dados fornecidos pelas linguagens de programação estão longe de se adequarem a todas as situações que nos surjam, o que implica a criação de novos tipos de dados de acordo com as características de cada problema. Numa situação que envolva formas gráficas a duas dimensões (2D), é perfeitamente justificável a criação de um novo tipo de dados, o ponto-2D, constituído por dois valores numéricos que representam as coordenadas x e y de um ponto no plano.

Um novo tipo de dados que se ajuste às características de um certo problema, acompanhado das operações básicas respectivas, constitui uma espécie de barreira sob a qual se encontra o que é essencial para trabalhar com os dados desse problema. É a isto que se chama abstracção de dados.

O par é uma entidade do Scheme que permite criar abstracções de dados. O par é acompanhado de operadores do tipo construtor, para criar objectos novos, e selector, para dar acesso às partes de um objecto.

Na abstracção tartaruga, os objectos que se podem criar e manipular são tartarugas. Sobre esta abstracção, enquanto disponibilizada num directório do DrScheme, é proposto um projecto de um jogo com alguma complexidade, designado por a tartaruga escondida.

### **Módulo 3.2**

Em relação aos dados, nem sempre as situações que encontramos se resolvem com dados simples. Em certos casos é necessário colocar objectos em sequência, o que requer dados que são compostos por outros dados. O par do Scheme permite compor dados com qualquer nível de complexidade, mas a lista é a forma privilegiada do Scheme para implementar sequências. A lista pode ser vista como um par, herdando por isso toda a funcionalidade deste, quer em termos de construção de objectos quer em termos de selecção dos seus componentes.



### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

R E 1 2 3 4

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

Neste módulo, a introdução à funcionalidade da lista é feita ao mesmo tempo que se estuda uma forma de implementar essa mesma funcionalidade e se aproveita assim para ir treinando a programação com listas.

Para além dos números, booleanos, pares e listas, aparecem agora os símbolos, como sendo um novo tipo de dados muito adequado em determinadas situações. Por exemplo, representar o mês de Janeiro pelo número 1 ou pelo símbolo `janeiro` não é bem a mesma coisa, a legibilidade do símbolo é muito maior.

A sequência de dados ou de objectos pode envolver elementos que são, eles próprios, sequências, e assim aparece a lista de listas. Surge, desta forma, a necessidade de processar a lista em profundidade, ou seja, a necessidade de percorrer a lista e explorar a estrutura de cada um dos componentes.

### **Módulo 3.3**

O desenvolvimento de um programa que funciona como teste à memória do utilizador surge como um exemplo de aplicação da abstracção conjunto. Já depois deste programa estar desenvolvido, introduzem-se alterações à abstracção conjunto, tendo em vista a obtenção de uma versão com melhor desempenho. A aplicação desta última versão no programa já desenvolvido, sem exigir qualquer alteração a este, demonstra bem que a abstracção funciona como uma barreira que esconde perfeitamente as suas características internas. Isto significa que mantendo a interface de uma abstracção, ou seja, a sua funcionalidade, é possível submetê-la a melhoramentos sem inviabilizar as aplicações que entretanto tenham sido desenvolvidas sobre ela.

### **Módulo 3.4**

A reutilização de um código desenvolvido pelo próprio ou por outros é uma prática comum e importante, especialmente se aquele código for digno de confiança. Neste módulo mostra-se como se pode reutilizar código que foi desenvolvido e disponibilizado sob a forma de uma abstracção. Neste contexto, são apresentados exemplos de utilização das abstracções janela gráfica e tabuleiro e a criação da abstracção eixos, também esta acompanhada de exemplos de utilização.

Com a abstracção janela gráfica é possível controlar uma caneta que desenha, pinta e escreve texto, e tudo isto sem exigir qualquer conhecimento dos procedimentos gráficos do DrScheme, que assim ficarão completamente escondidos por esta barreira da janela gráfica. Sobre a abstracção janela gráfica criaram-se as abstracções eixos e tabuleiro, a primeira para visualizar sistemas de eixos ortogonais e a segunda para visualizar e manipular tabuleiros semelhantes aos tabuleiros de damas.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Exercícios e Exemplos

São apresentados alguns exercícios e exemplos para consolidação da matéria tratada nos módulos 3.1 a 3.4.

Para estes exercícios e exemplos recomenda-se a consulta do Anexo A (principais procedimentos do Scheme), em particular a secção correspondente ao processamento de listas. Para reutilização de abstracções previamente desenvolvidas ou criação de novas, sugere-se o estudo do Anexo C (reutilização de código) e a consulta dos Anexo B (abstracção janela gráfica) e Anexo D (abstracção tabuleiro) sempre que achar necessário.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 1

Pretende-se criar uma abstracção de dados relacionada com entidades do tipo ponto-2D, compostas por dois valores numéricos que representam as coordenadas de pontos num plano.

Escreva em Scheme os seguintes procedimentos:

- 1- O construtor `faz-ponto` recebe dois valores numéricos, respectivamente as coordenadas `x` e `y` de um ponto, e devolve uma entidade do tipo `ponto-2D`. Os selectores `x-coord` e `y-coord` recebem um `ponto-2D` e devolvem, respectivamente, as coordenadas `x` e `y` desse ponto.
- 2- Um rectângulo, com os lados paralelos aos eixos de coordenadas, pode ser definido por dois vértices opostos. O construtor `faz-rectangulo` recebe duas entidades do tipo `ponto-2D`, que representam dois vértices opostos que definem um rectângulo, e devolve uma entidade do tipo `rectangulo`. O procedimento `area-rectangulo` recebe um `rectangulo` e devolve a área desse rectângulo.
- 3- O procedimento `distancia` recebe duas entidades `ponto-2D` como argumentos e devolve a distância entre elas.
- 4- O procedimento `area-de-interseccao` recebe duas entidades `rectangulo` e devolve a área de intersecção desses 2 rectângulos.



Desenvolva e teste os procedimentos `faz-ponto`, `x-coord`, `y-coord`, `faz-rectangulo`, `area-rectangulo`, `distancia` e `area-de-interseccao`.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Exercício 2

Escreva em Scheme os seguintes procedimentos:

- 1- O procedimento substitui-primeira-ocorrecia tem 3 parâmetros, lista, novo, e velho, e devolve uma lista equivalente a lista depois de substituir a primeira ocorrência de velho por novo.

```
> (substitui-primeira-ocorrecia '(o meu cao e o teu cao) 'gato 'cao)
(o meu gato e o teu cao)
> (substitui-primeira-ocorrecia '() 'gato 'cao)
()
```

- 2- O procedimento substitui-todas-ocorrecias tem 3 parâmetros, lista, novo, e velho, e devolve uma lista equivalente a lista depois de substituir todas as ocorrências de velho por novo.

```
> (substitui-todas-ocorrecias '(o meu cao e o teu cao) 'gato 'cao)
(o meu gato e o teu gato)
> (substitui-todas-ocorrecias '() 'gato 'cao)
()
```



Desenvolva e teste os procedimentos substitui-primeira-ocorrecia e substitui-todas-ocorrecias.

## Exercício 3

Escreva em Scheme os seguintes procedimentos:

- 1- O procedimento elimina-primeira-ocorrecia tem 2 parâmetros, lista e elemento, e devolve uma lista equivalente a lista depois de eliminar a primeira ocorrência de elemento.

```
> (elimina-primeira-ocorrecia '(o meu cao e' esperto) 'cao)
(o meu e' esperto)
> (elimina-primeira-ocorrecia '() 'cao)
()
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

2- O procedimento `elimina-todas-ocorrencias` tem 2 parâmetros, `lista` e `elemento`, e devolve uma lista equivalente a `lista` depois de eliminar todas as ocorrências de `elemento`.

3- O procedimento `elimina-ultima-ocorrencia` tem 2 parâmetros, `lista` e `elemento`, e devolve uma lista equivalente a `lista` depois de eliminar a última ocorrência de `elemento`.



Desenvolva e teste os procedimentos `elimina-primeira-ocorrencia`, `elimina-todas-ocorrencias` e `elimina-ultima-ocorrencia`.

### Exercício 4

Escreva em Scheme os seguintes procedimentos:

1- O procedimento `junta-ordenado` tem dois parâmetros, `crescente-1` e `crescente-2`, que são duas listas com elementos numéricos ordenados do menor para o maior. Sabe-se que `junta-ordenado` devolve uma lista cujos elementos são os elementos das listas dadas, em ordem crescente.

```
> (junta-ordenado '(2 3 40) '(1 3 34))  
(1 2 3 3 34 40)  
> (junta-ordenado '() '(1 3 34))  
(1 3 34)  
> (junta-ordenado '(2 3 40) '())  
(2 3 40)
```

2- O procedimento `junta-ordenado-sem-repetidos`, com os mesmos parâmetros de `junta-ordenado`, não inclui elementos repetidos na lista que devolve.

```
> (junta-ordenado-sem-repetidos '(2 3 40) '(1 3 34))  
(1 2 3 34 40)  
> (junta-ordenado-sem-repetidos '() '(1 3 34))  
(1 3 34)  
> (junta-ordenado-sem-repetidos '(2 3 40) '())  
(2 3 40)
```



INTRODUÇÃO
1 - O ESSENCIAL DO SCHEME
2 - RECURSIVIDADE
3 - ABSTRACÇÃO DE DADOS
R E 1 2 3 4
4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE
5 - ABSTRACÇÕES COM DADOS MUTÁVEIS
6 - SCHEME E OUTRAS TECNOLOGIAS
7-EXERCÍCIOS E PROJECTOS
ANEXOS



Desenvolva e teste os procedimentos junta-ordenado e junta-ordenado-sem-repetidos.

## Exercício 5

Escreva em Scheme os seguintes procedimentos:

- 1- Antes de escrever o procedimento inverter-posicao-de-todos, analise as duas chamadas que se seguem.

```
> (reverse '(a (b c) (d (e f))))
((d (e f)) (b c) a)
> (inverter-posicao-de-todos '(a (b c) (d (e f))))
(((f e) d) (c b) a)
```

- 2- Antes de escrever o procedimento substituir-todos, analise as três chamadas que se seguem.

```
> (substitui-todos '(a (b (a c)) (a (d a))) 'z 'a)
(z (b (z c) (z (d z))))
> (substitui-todos '(((1) (0)) 0 '(1)))
((0 (0)))
> (substitui-todos '() 'cao 'gato)
()
```

- 3- O procedimento planificador, com o parâmetro lista, devolve uma lista formada por todos os elementos de lista que não sejam pares.

```
> (planificador '(a (b c d) ((e f) g)))
(a b c d e f g)
```



Desenvolva e teste os procedimentos inverte-posicao-de-todos, substitui-todos e planificador.



### Exemplo 1

## Projecto - Caminho

Vamos imaginar um caminho traçado sobre um tabuleiro composto por 25 células, organizadas numa matriz  $5 \times 5$ . Numa das células do tabuleiro é colocada uma tartaruga que apenas se desloca na horizontal, para a célula imediatamente ao lado (nossa lado direito), ou na vertical, para a célula imediatamente abaixo. Chegando à coluna 5, a tartaruga não se pode deslocar mais para a direita. Também, quando atinge a linha do fundo, a tartaruga não se pode deslocar mais na vertical, para baixo. O objectivo da tartaruga é atingir a célula 25.

O caminho pode complicar-se, pois é possível colocar obstáculos intransponíveis em várias células. Por exemplo, colocando obstáculos nas células 2, 9, 12, 14, 15, 17 e 23, o tabuleiro toma o aspecto indicado na figura.

A tartaruga segue sempre um caminho de acordo com uma estratégia muito simples: desloca-se prioritariamente na horizontal, para a direita, e só quando fica bloqueada neste movimento (ou encontrou um obstáculo ou alcançou a coluna 5) é que tenta o deslocamento na vertical, para baixo.

Para representar computacionalmente um tabuleiro, escolhemos uma lista contendo os índices das células com obstáculos. O tabuleiro, com obstáculos indicados, terá a seguinte representação: a lista (2 9 12 14 15 17 23).

O procedimento caminho, que se pretende desenvolver, tem dois parâmetros: o ponto de partida, onde se coloca a tartaruga, e uma lista que representa o tabuleiro e vai indicando o caminho percorrido por esta de acordo com a estratégia já indicada.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	<del>2</del>	3	4	5
6	7	8	<del>9</del>	10
11	<del>12</del>	13	<del>14</del>	<del>15</del>
16	<del>17</del>	18	19	20
21	22	<del>23</del>	24	25

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (define tabuleiro '(2 9 12 14 15 17 23))
> (caminho 1 tabuleiro)
1; 6; 7; 8; 13; 18; 19; 20; 25; consegui
> (caminho 11 tabuleiro)
11; 16; 21; 22; falhei
> (caminho 2 tabuleiro)
partida errada!!!
> (caminho 3 tabuleiro)
3; 4; 5; 10; falhei
```

O procedimento caminho verifica se o ponto de partida, fornecido como argumento, coincide com uma célula ocupada por um obstáculo. Se assim for, a mensagem `partida errada!!!` é imediatamente visualizada e o processo termina. Tratando-se de uma célula livre, inicia-se a caminhada, ou seja, é chamado o procedimento andar, que é o núcleo central da resolução do problema, como verá.

```
(define caminho
  (lambda (partida obstaculos)
    (if (em-obstaculo? partida obstaculos) ; partida sobre um obstáculo?
        (display "partida errada!!!") ; sim
        (andar partida obstaculos)))) ; não
```

O procedimento caminho utiliza o procedimento `em-obstaculo?` que indica se uma célula contém ou não um obstáculo.

```
(define em-obstaculo?
  (lambda (elem lista)
    (member elem lista)))
```



No procedimento `em-obstaculo?`, utiliza-se um pequeno artifício que complica um pouco a sua legibilidade. Estamos a falar da utilização de `member`.

Explique a sua utilização nesta situação.

Apresente uma solução alternativa para que o procedimento se apresente mais legível.



O procedimento andar é recursivo e tem dois casos de terminação:

- Quando posicao (variável que modela a posição da tartaruga) atinge a célula 25, que significa objectivo atingido;
  - Quando a tartaruga fica bloqueada (não consegue deslocar-se nem na horizontal nem na vertical).

O procedimento andar tem também dois casos gerais

- Estando a tartaruga bloqueada na horizontal, avança para a linha seguinte:

(andar (+ 5 posicao)) ; avançar para a célula imediatamente  
; abaixo (na linha seguinte) corresponde  
; a somar 5 à posição actual da tartaruga.

- Não estando bloqueada na horizontal, avança sobre essa linha.

(andar (add1 posicao)) ; avançar para a célula imediatamente  
; ao lado (na mesma linha) corresponde  
; a somar 1 à posição actual da tartaruga.

Pelos casos gerais do procedimento `andar`, conclui-se que a operação de redução, utilizada no passo recursivo, se traduz em adicionar um certo valor a `posicao`, tentando assim chegar a 25. Num dos casos, adicionando-lhe 1 (movimento na horizontal para a direita), no outro, adicionando-lhe 5 (movimento na vertical para baixo).

```
(define andar
  (lambda (posicao obstaculos)
    (display posicao) ; visualiza posicao
    (display "; ")
    (cond
      ((= posicao 25) ; atingiu a célula 25
       (display "consegui")
       (newline))
      ((bloqueado? posicao obstaculos)
       (display "falhei") ; ficou bloqueado
       (newline))
      ((bloqueado-horiz? posicao obstaculos) ; bloqueado na horizontal?
       (andar (+ 5 posicao) obstaculos)) ; Sim. Avança na vertical
      (else
        (andar (add1 posicao) obstaculos)))))) ; Não. Avanca na horizontal
```



## Explique o funcionamento do procedimento andar.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



O procedimento andar utiliza o procedimento bloqueado?, que determina se a tartaruga está ou não bloqueada na horizontal ou na vertical.

```
(define bloqueado?
  (lambda (posi obs) ; bloqueado significa...
    (and (bloqueado-horiz? posi obs) ; bloqueado na horizontal
         (bloqueado-vert? posi obs)))) ; e bloqueado na vertical

(define bloqueado-horiz?
  (lambda (posi obs)
    (or ; bloqueado na horizontal significa estar...
        (em-coluna-direita? posi) ; na coluna 5
        (em-obstaculo? (add1 posi) obs)))) ; ou ter um obstáculo à direita

(define bloqueado-vert?
  (lambda (posi obs)
    (or ; bloqueado na vertical significa estar...
        (em-linha-fundo? posi) ; na linha 5, ou ter um
        (em-obstaculo? (+ posi 5) obs)))) ; obstáculo por baixo

(define em-coluna-direita? ; está encostado à direita?
  (lambda (posicao)
    (zero? (remainder posicao 5)))

(define em-linha-fundo? ; está encostado ao fundo?
  (lambda (posicao)
    (> posicao 20)))
```

Acabámos de apresentar um exemplo em que a tarefa a tratar é relativamente complexa e exige uma abordagem de-cima-para-baixo, enquanto a abstracção de dados passa quase despercebida. O tabuleiro com os obstáculos foi representado através de uma lista perfeitamente normal. Quanto à tartaruga, um inteiro chegou para representar a célula que ocupa no tabuleiro.

INTRODUÇÃO
1 - O ESSENCIAL DO SCHEME
2 - RECURSIVIDADE
3 - ABSTRACÇÃO DE DADOS
R E 1 2 3 4
4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE
5 - ABSTRACÇÕES COM DADOS MUTÁVEIS
6 - SCHEME E OUTRAS TECNOLOGIAS
7-EXERCÍCIOS E PROJECTOS
ANEXOS



Teste o procedimento `caminho`, acrescentando ou retirando obstáculos e experimentando vários pontos de partida.

## Exercício 6

No contexto do exemplo anterior, introduza as alterações necessárias à solução apresentada, para chegar ao procedimento `caminho-no-tabuleiro`, que passa a responder como se indica.

```
> (define obs (list 2 9 12 14 15 17 23))  
> (caminho-no-tabuleiro 1 obs)
```

```
1  x  .  .  .  
6  7  8  x  .  
.  x 13  x  x  
.  x 18 19 20  
.  .  x  . 25
```

```
> (caminho-no-tabuleiro 2 obs)  
partida errada!!!
```

```
> (caminho-no-tabuleiro 3 obs)
```

```
.  x  3  4  5  
.  .  .  x 10  
.  x  .  x  x  
.  x  .  .  .  
.  .  x  .  .
```



Desenvolva e teste o procedimento `caminho-no-tabuleiro`, acrescentando ou retirando obstáculos e experimentando vários pontos de partida.



## Exercício 7

No contexto do exemplo anterior, introduza as alterações necessárias à solução apresentada para chegar ao procedimento caminho-no-tabuleiro-graf que, utilizando a janela corrente e a abstracção tabuleiro, passa a responder como se indica. Este procedimento tem apenas um parâmetro que define a posição inicial da tartaruga. Para incluir a abstracção tabuleiro, fazer

```
(require (lib "tabuleiro.scm" "user-feup"))
```

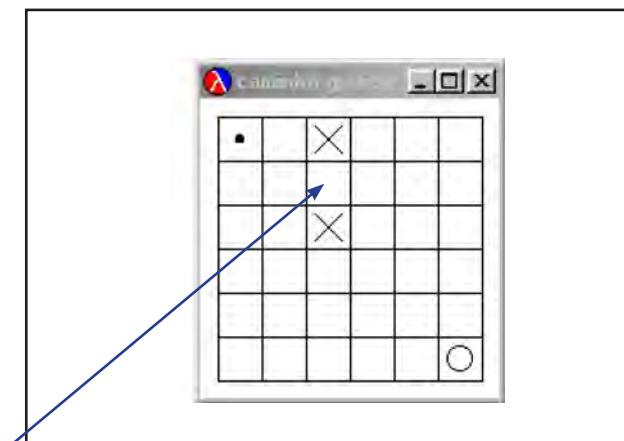
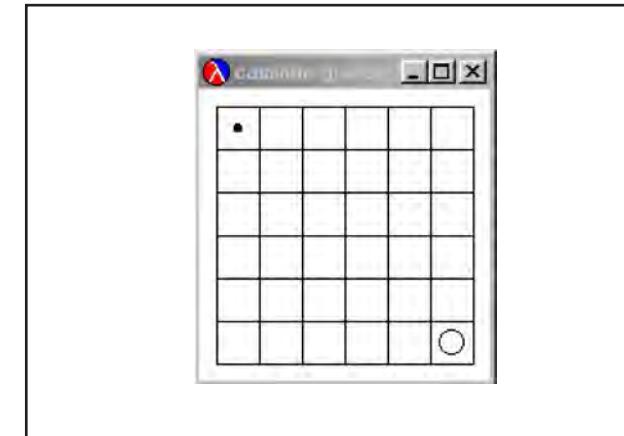
> (caminho 1)  
Lado do tabuleiro em celulas : 6  
Lado da celula em pixels: 25

1- obstaculo      2- posicao partida  
8- partida      9- limpar trajectoria  
10- fim  
1  
Coluna (de 0 a 5): 2  
Linha (de 0 a 5): 0

1- obstaculo      2- posicao partida  
8- partida      9- limpar trajectoria  
10- fim  
1  
Coluna (de 0 a 5): 2  
Linha (de 0 a 5): 1

1- obstaculo      2- posicao partida  
8- partida      9- limpar trajectoria  
10- fim  
1  
Coluna (de 0 a 5): 2  
Linha (de 0 a 5): 2

1- obstaculo      2- posicao partida  
8- partida      9- limpar trajectoria  
10- fim  
1  
Coluna (de 0 a 5): 2  
Linha (de 0 a 5): 1



a escolha de célula já preenchida elimina obstáculo

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

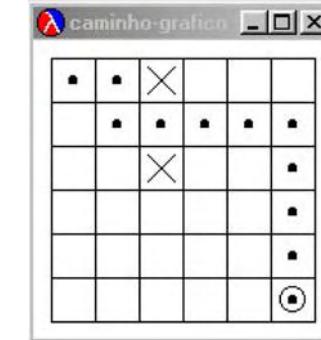
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
1- obstaculo    2- posicao partida
8- partida      9- limpar trajectoria
10- fim
8
```

```
1- obstaculo    2- posicao partida
8- partida      9- limpar trajectoria
10- fim
10
```

acabou



Desenvolva e teste o procedimento caminho-no-tabuleiro-graf.

### Exemplo 2

#### Projecto - Adivinha-número

Um programa designado por adivinha-numero não tem parâmetros, escolhe aleatoriamente um número entre 0 e 31 e convida o utilizador a adivinhar esse número, permitindo-lhe, previamente, fazer 4 enumerações, que serão "comentadas" pelo computador. Vejamos um exemplo de utilização.

```
> (adivinha-numero)
Estou a escolher um numero entre 0 e 31
... e tu vais tentar adivinha'-lo com 4 sugestoes...
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
Indicar lista de numeros entre 0 e 31:  
(1 3 5 6 7 10 12 14 16 23 31 24)  
O numero ESTA' nos indicados  
Indicar lista de numeros entre 0 e 31:  
(1 3 5 6 7 10)  
O numero NAO ESTA' nos indicados  
Indicar lista de numeros entre 0 e 31:  
(12 14 16)  
O numero NAO ESTA' nos indicados  
Indicar lista de numeros entre 0 e 31:  
(23 31)  
O numero ESTA' nos indicados  
A tua vez de adivinhar o numero escolhido: 23  
Falhaste. O numero certo seria: 31  
  
> (adivinha-numero)
```

Estou a escolher um numero entre 0 e 31  
... e tu vais tentar adivinha'-lo com 4 sugestoes...

```
Indicar lista de numeros entre 0 e 31:  
(1 2 3 4 5 6 7 8 9 10)  
O numero ESTA' nos indicados  
Indicar lista de numeros entre 0 e 31:  
(1 2 3 4 5)  
O numero NAO ESTA' nos indicados  
Indicar lista de numeros entre 0 e 31:  
(6 7 8)  
O numero NAO ESTA' nos indicados  
Indicar lista de numeros entre 0 e 31:  
(9)  
O numero NAO ESTA' nos indicados  
A tua vez de adivinhar o numero escolhido: 10  
Parabens... Acertaste.
```

Em relação a este exemplo considerou-se não ser necessário gastar muito tempo com a definição de uma abstracção de dados específica, pois manifestou-se ser perfeitamente razoável a utilização de uma lista de números e alguns procedimentos do Scheme para a manipulação de listas. Quanto aos aspectos funcionais do jogo, optou-se por uma abordagem de-cima-para-baixo, para identificar as tarefas principais do programa adivinha-numero.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

- gerar um número aleatório entre 0 e 31;
- pedir quatro vezes ao utilizador uma lista de números entre 0 e 31 e comentar as respectivas respostas;
- convidar o utilizador a adivinhar o número seleccionado aleatoriamente;
- apresentar a mensagem de parabéns ou de lamentação por não ter acertado no número.

A segunda tarefa, devido ao seu carácter repetitivo, justificou o procedimento auxiliar `faz-perguntas`, procedimento recursivo em que o caso base corresponde ao contador `vezes` atingir o valor zero. As tarefas de `faz-perguntas` são:

- visualizar uma mensagem pedindo a indicação de uma lista de números entre 0 e 31;
- verificar se o número gerado aleatoriamente se encontra ou não na lista indicada e informar o utilizador desse facto;
- repetir as tarefas anteriores até atingir o caso base.

```
; para aceder ao procedimento roleta-1-n contido em varios.scm
; (ver Anexo C)
(require (lib "varios.scm" "user-feup"))

;
(define adivinha-numero
  (lambda ()
    ;
    ; definição local do procedimento faz-perguntas
    ;
    (letrec
      ((faz-perguntas
        (lambda (vezes num-selec)
          (if (positive? vezes)
              (begin
                (display
                  "Indicar lista de numeros entre 0 e 31:")
                (newline)
                (if (member num-selec (read))
                    (display
                      "O numero ESTA' nos indicados")
                    (display
                      "O numero NAO ESTA' nos indicados"))
                (newline)
                (faz-perguntas (sub1 vezes) num-selec))))))
      ;
      ----- o corpo principal do programa adivinha-numero -----
      (newline)
      (newline)))
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(display "Estou a escolher um numero entre 0 e 31")
(newline)
(display "... e tu vais tentar adivinha'-lo com 4 sugestoes...")
(newline)
(newline)
(let ((num-sel (sub1 (roleta-1-n 32))))
  (faz-perguntas 4 num-sel)
  (display "A tua vez de adivinhar o numero escolhido: ")
  (newline)
  (if (= num-sel (read))
      (begin
        (newline)
        (display "Parabens... Acertaste."))
      (begin
        (display "Falhaste. O numero certo seria: ")
        (display num-sel)))
  (newline)))))
```



Teste o programa adivinha-numero.



No corpo do programa **adivinha-numero**, identifique as suas tarefas principais.  
No corpo do procedimento local **faz-perguntas**, identifique as suas tarefas principais

**Exemplo 3** - avançar e recuar... a técnica de *backtracking*

Num tabuleiro de xadrez uma rainha ataca outra rainha se ambas estiverem na mesma linha horizontal, vertical ou diagonal.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

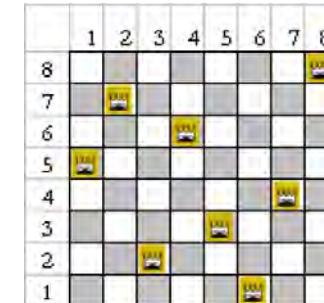
## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

O problema que se pretende resolver consiste em posicionar 8 rainhas num tabuleiro de xadrez, de modo a que nenhuma delas possa atacar outra. No tabuleiro indicado ao lado é apresentada uma dessas posições, mas existem ainda mais 91... e o desafio é descobri-las com a ajuda do computador.

Uma forma de codificar a posição com 8 rainhas, indicada no tabuleiro, começando na rainha mais à direita e terminando na que está mais à esquerda, é a lista (5 7 2 6 3 1 4 8) de comprimento 8.



**Explique como funciona a codificação que levou à lista (5 7 2 6 3 1 4 8).**

**Esta lista representa uma posição legal, pois nenhuma rainha ataca outra.**

**Das posições que se seguem, embora não sendo de comprimento 8, descubra manualmente as que são legais e ilegais:**

**(1 4 8), (8 4 8), (5), (6 4 8) e (7 1 4 8)**

Em vez de verificar manualmente se uma posição é ou não legal, pode utilizar, como se fosse uma caixa-preta, o predicado `tentativa-legal?`.

Por exemplo, para verificar se (3 7 1 4 8) é ou não legal bastará fazer executar (`tentativa-legal? 3 (list 7 1 4 8)`), supondo que (7 1 4 8) é legal, como pode ver na figura.



**Observando a figura, acha que (3 7 1 4 8) é uma posição legal? Justifique.**



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

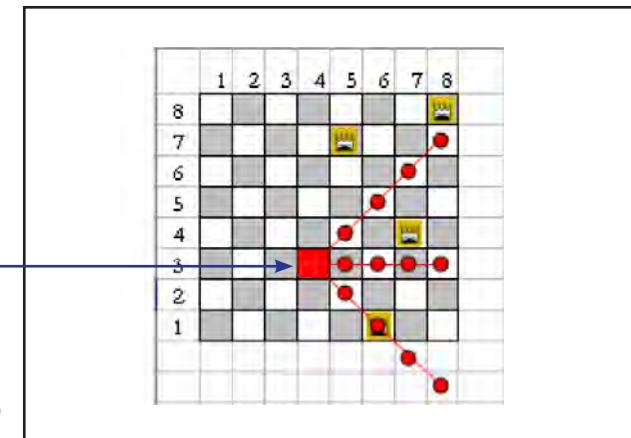
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



; verifique se, ao juntar um novo elemento a uma posição legal, resulta uma posição legal com o comprimento incrementado de 1

```
(define tentativa-legal?
  (lambda (novo pos-i-legal)
    (letrec ((tentativa-aux?
              ;
              (lambda (p-legal distancia)
                (if (null? p-legal)
                    #t
                    (let ((prox (car p-legal)))
                      (and
                        (not (= prox novo))
                        (not (= prox (+ novo distancia))))
                        (not (= prox (- novo distancia)))
                        (tentativa-aux? (cdr p-legal)
                                         (add1 distancia)))))))
                  (tentativa-aux? pos-i-legal 1)))))
```



Teste o procedimento `tentativa-legal?`.



Apresente uma explicação para o modo de funcionamento do predicado `tentativa-legal?`  
De acordo com a explicação apresentada, comente este procedimento.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Imagine que se encontra na posição legal correspondente a (4 1 7 5 3 8), indicada no tabuleiro, imediatamente antes de colocar a rainha na coluna 2 e na linha 6.

No que respeita à coluna 2, nas linhas 8 e 7 não seria possível colocar uma rainha, foi por isso que se escolheu a linha 6.

A linha 6 é a primeira hipótese possível. No entanto, a posição obtida, (6 4 1 7 5 3 8), apesar de legal, impede que se alcance uma posição legal de comprimento 8.

	1	2	3	4	5	6	7	8
8								Q
7								Q
6								Q
5								Q
4								Q
3								Q
2								Q
1								Q



Verifique, a começar pela linha 8 e tentando até à linha 1, a impossibilidade de colocar uma rainha na coluna 1 a partir da posição legal (6 4 1 7 5 3 8).

Que solução propõe, então?



Se continuasse a recuar, em que situação poderia concluir que não haveria solução?



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

A ideia é partir de uma posição vazia e fazer (`faz-solucao '()`)

```
(define tentativa-inicial '())
(define comprimento-solucao 8)
(define faz-solucao
  (lambda (posi-legal)
    (if (solucao? posi-legal)
        posi-legal
        (avanca tentativa-inicial posi-legal))))
```

e com avanços e recuos

```
(define avanca
  (lambda (novo posi-legal)
    (cond
      ((zero? novo) (recua posi-legal))
      ((tentativa-legal? novo posi-legal)
       (faz-solucao (cons novo posi-legal)))
      (else (avanca (sub1 novo) posi-legal)))))

(define recua
  (lambda (posi-legal)
    (if (null? posi-legal)
        '()
        (list)
        (avanca (sub1 (car posi-legal))
                (cdr posi-legal)))))
```

chegar certamente a uma solução como, por exemplo, `(5 7 2 6 3 1 4 8)`.

```
(define solucao?
  (lambda (posi-legal)
    (= comprimento-solucao (length posi-legal))))
```



Teste o procedimento `faz-solucao` e os procedimentos auxiliares.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

A partir de uma lista vazia chegou-se à solução (5 7 2 6 3 1 4 8). Para encontrar outra solução, a ideia é recuar sobre a solução obtida, imaginando que é uma "falha" e tentar construir outra solução a partir dessa situação de "falha". Por exemplo, para obter 3 soluções,

```
> (let*
  ((sol1 (faz-solucao '())))
  (sol2 (faz-solucao (recua sol1)))
  (sol3 (faz-solucao (recua sol2))))
  (list sol1 sol2 sol3))

((5 7 2 6 3 1 4 8) (4 7 5 2 6 1 3 8) (6 4 7 1 3 5 2 8))
>
```



Verifique manualmente que estas 3 soluções são, de facto, posições legais de 8 damas num tabuleiro de xadrez.

E agora para encontrar todas as soluções possíveis.

```
(define faz-todas-as-solucoes
  (lambda ()
    (letrec
      ((ciclo
        (lambda (solucao)
          (if (null? solucao)
              '()
              (cons solucao
                (ciclo (recua solucao)))))))
      (ciclo (faz-solucao (list)))))

> (length (faz-todas-as-solucoes))
92
> (faz-todas-as-solucoes)
((5 7 2 6 3 1 4 8)
 (4 7 5 2 6 1 3 8)
 (6 4 7 1 3 5 2 8)
 ...
 (3 5 2 8 6 4 7 1)
 (5 2 4 7 3 8 6 1)
 (4 2 7 3 6 8 5 1))
>
```





Teste o procedimento `faz-todas-as-solucoes`.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

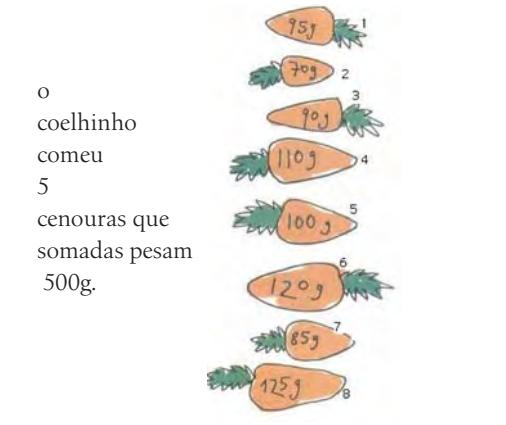


Apresente uma justificação para o caso de terminação do procedimento auxiliar designado por `ciclo` no procedimento `faz-todas-as-solucoes`.

### Exercício 8 - um desafio com alguma dificuldade...

Uma solução para este problema poderá ser procurada no exemplo anterior, sobre a colocação de 8 rainhas num tabuleiro de xadrez com a técnica de *backtracking*.

Como pode verificar, a base do enunciado é um problema que normalmente se coloca no ensino básico! No entanto, o problema surge agora com uma formulação mais geral, pretendendo-se uma solução automática e não manual como acontece no ensino básico.



- 1- comece por analisar o problema proposto e procure, manualmente, a solução pedida.
- 2- seguidamente, imagine que uma lista é composta pelo peso de cada uma das cenouras.
- 3- pretende-se que desenvolva um procedimento em Scheme que possa receber uma "lista de cenouras" e um peso e devolva uma lista com as cenouras cujos pesos somados equivalem ao peso dado.

Veja uma hipótese de diálogo, com o procedimento pretendido.

```
> (define lista-cenouras-1 (list 5 7 6 2))  
> (as-cenouras lista-cenouras-1 9)
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
(7 2)
> (as-cenouras lista-cenouras-1 10)
()
```

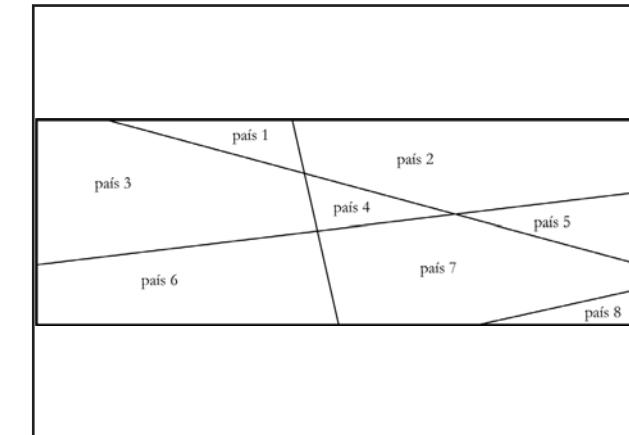


Desenvolva e teste o procedimento `as-cenouras`.

**Exercício 9** - mais um desafio com alguma dificuldade... uma aplicação da técnica de *backtracking*?  
Projecto - Colorir mapas

Pretende-se colorir mapas de países, com um número mínimo de cores e sem que cores iguais sejam utilizadas em países vizinhos. São considerados países vizinhos os que tiverem, pelo menos, uma linha de fronteira comum (caso dos países 1 e 2). O mapa é modelado através de uma lista, existindo um elemento por cada país do mapa. O elemento correspondente a um país representa os países vizinhos desse país. Por exemplo, o país 1, sendo o primeiro elemento da lista, tem como vizinhos os países 2 e 3.

```
(define mapa1
  ' ((2 3)          ; país 1
    (1 4 5)         ; país 2
    (1 6 4)         ; país 3
    (7 2 3)         ; país 4
    (7 2)           ; país 5
    (3 7)           ; país 6
    (4 5 6 8)       ; país 7
    (7) ))          ; país 8
```



A modelação das cores a utilizar pode também ser feita através de uma lista.

```
(define paleta-cores
  ' (c1 c2 c3 c4))           ; neste caso, utilizam-se 4 cores
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



A atribuição de cores aos vários países é conseguida com chamadas a `colorir-mapa`, procedimento que recebe dois argumentos, mais especificamente duas listas, uma que representa o mapa e outra que representa as cores utilizadas para colorir o mapa. Este procedimento devolve uma lista com a indicação das cores a atribuir a cada país, a começar pelo país 1.

```
> (colorir-mapa mapal paleta-cores)
(c1 c2 c2 c1 c1 c1 c2 c1)
```



**Daqui se concluiria que, para colorir o mapa, bastariam apenas duas cores.  
Concorda?**

De uma tentativa para colorir o mesmo mapa, com um conjunto de apenas uma cor, não resultaria solução possível.

```
> (colorir-mapa mapal '(azul))      (paleta apenas com a cor azul)
Poucas cores. Nao ha' solucao...
```

- 1- Faça uma abordagem de-cima-para-baixo ao problema proposto.
- 2- Escreva em Scheme o procedimento `colorir-mapa`, com base nos resultados da abordagem anterior.



Desenvolva e teste o programa `colorir-mapa`.

Convém ter em consideração que está perante um problema de média/elevada complexidade e que, por isso, não deverá desanimoar se uma solução não surgir de imediato. Este é o tipo de problema que o pode acompanhar durante dias, levando-o a exercitar a sua capacidade criativa e os seus conhecimentos de programação.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



#### Pista

Provavelmente a dispensar se já trabalhou o exemplo das 8 rainhas ou o exercício das cenouras, indicados imediatamente antes.

A ideia que se pretende explorar consiste em atribuir a primeira cor da paleta ao primeiro país da lista. Assim, a lista de cores, com a solução a procurar, inicia-se com a cor atribuída ao primeiro país. Vamos agora passar ao país seguinte e procurar uma cor para ele. Por tentativas, começaremos pela primeira cor da paleta. Mas será que esta é uma boa cor para colorir o segundo país? Só o saberemos comparando-a com as cores de todos os países vizinhos, entretanto já coloridos. De facto, nesta altura, só o primeiro país está colorido e, ainda por cima, com uma cor igual, o que significa que, se o primeiro país faz parte da lista dos vizinhos do segundo país, esta cor não é boa. Por ordem, tenta-se uma nova cor da paleta, até encontrar uma boa cor para o segundo país. E, de forma idêntica, passaríamos ao terceiro país, ao quarto, até não haver mais países para colorir. Nessa altura, existirá uma solução para colorir todos os países.

A parte mais delicada do problema corresponde à situação em que, ao tentar encontrar uma boa cor para um país, começando, como sempre, pela primeira cor da paleta, é alcançada e experimentada a última cor da paleta sem conseguir esse objectivo. Isto poderá ocorrer por uma de duas razões. Ou a paleta tem poucas cores e não existe uma solução para colorir os países sem que a mesma cor se encontre em dois países vizinhos, ou ocorreu alguma má escolha de cores para países anteriores. Este último caso, obriga a recuar para o país anterior, para, a seguir à cor que lhe foi anteriormente atribuída, procurar outra boa cor. Se esta for encontrada, avançamos novamente para o país seguinte. Se não for encontrada, recuamos para o país anterior. Se os recuos forem de tal ordem que se alcance o primeiro país, então não vale a pena fazer outras tentativas, pois já se pode concluir que as cores da paleta não chegam para colorir o mapa. Facilmente se entende que, recuando até ao primeiro país, que havia sido preenchido com a primeira cor da paleta, é inútil procurar outra boa cor como se fez nos outros países. De facto, seria pura perda de tempo, uma vez que para este tipo de problema, qualquer cor da paleta é igualmente boa para o primeiro país.

Concorda com esta conclusão?





Reflectindo um pouco mais sobre a ideia exposta, podemos então considerar que a tarefa `colorir-mapa` se desenvolve sobre uma tarefa a designar por `procurar-solucao`, cujos parâmetros estariam associados ao país que, naquele momento, se vai colorir (inteiro com valores de 1 a  $p$ , sendo este o número de países do mapa), à cor por onde se inicia a procura de uma boa cor (inteiro com valores de 1 a  $c$ , sendo este o número de cores na paleta) e à solução encontrada até ao momento (lista de inteiros com valores de 1 a  $c$ , sendo este o número de cores na paleta que representam os índices das cores atribuídas aos países entretanto coloridos). Procuremos agora identificar as sub-tarefas de `procurar-solucao`.

- Enquanto houver países no mapa para colorir, tomar a primeira cor da paleta e verificar se é uma boa cor para o país a colorir. Uma cor será boa para atribuir a um país se nenhum dos países vizinhos, já com cor atribuída, a utiliza. Sendo uma boa cor, retoma-se novamente a tarefa `procurar-solucao` com o país seguinte, a primeira cor da paleta e a solução a que se acrescentou a cor encontrada. Sendo uma má cor, também se retoma `procurar-solucao`, mas com o mesmo país, a cor seguinte da paleta e a solução sem qualquer alteração.
- No entanto, pode atingir-se o fim da paleta sem se encontrar uma boa cor. Ou, de facto, não há solução possível, ou alguma das escolhas não foi bem feita. Assim, recua-se para a solução anterior, ou seja, retoma-se `procurar-solucao` com o país anterior, com a cor a partir da cor na altura escolhida, e com a solução a que se retirou a última cor. É claro que se tiver de recuar tanto que alcance o primeiro país, é sinal de que não há mesmo solução possível.
- Quando não há mais países para colorir, a tarefa `procurar-solucao` devolve a solução encontrada, convertendo os índices das cores nas designações indicadas na paleta.

Desta descrição ainda se identificam algumas sub-tarefas:

- `cor-boa?` para um país, que pode necessitar de `cores-dos-vizinhos-ja-coloridos`
- `cores-do-mapa` para converter uma lista de índices de cores numa lista com as designações da paleta.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS





# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Localizacao do mineiro: col. 5, lin. 3  
Situacao do mineiro: 99

Indicar pista para a proxima mina

N/S/E/O/numero? E

Distancia calculada pelo computador: 2

-----  
Numero restante de jogadas: 9

Terreno de minas:

Col. 1 2 3 4 5 6 7 8

Lin.

1	45	34	-55	2	-33	80	-78	4
2	37	5	34	-45	6	-67	-28	-7
3	-6	-54	2	34	99	90	78	-89
4	23	45	-78	95	-4	35	-67	49
5	1	45	2	-7	67	44	-90	34
6	80	-87	45	-38	58	52	-95	-73
7	77	91	-36	62	-5	9	-23	74
8	-35	74	95	-82	5	-7	9	44

N

O -|- E

S

Localizacao do mineiro: col. 7, lin. 3

Situacao do mineiro: 177

Indicar pista para a proxima mina

N/S/E/O/numero? 1

Orientacao calculada pelo computador: N

-----  
Numero restante de jogadas: 8

Terreno de minas:

Col. 1 2 3 4 5 6 7 8

Lin.

1	45	34	-55	2	-33	80	-78	4
2	37	5	34	-45	6	-67	-28	-7
3	-6	-54	2	34	99	90	78	-89
4	23	45	-78	95	-4	35	-67	49
5	1	45	2	-7	67	44	-90	34
6	80	-87	45	-38	58	52	-95	-73
7	77	91	-36	62	-5	9	-23	74
8	-35	74	95	-82	5	-7	9	44





## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

### Exercício 11

#### Projecto - Lançamento de projécteis

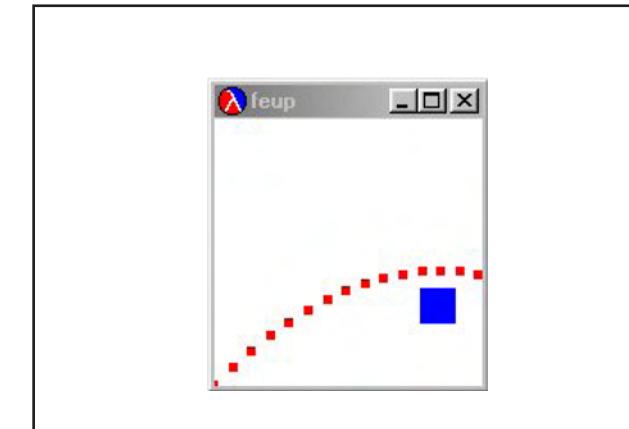
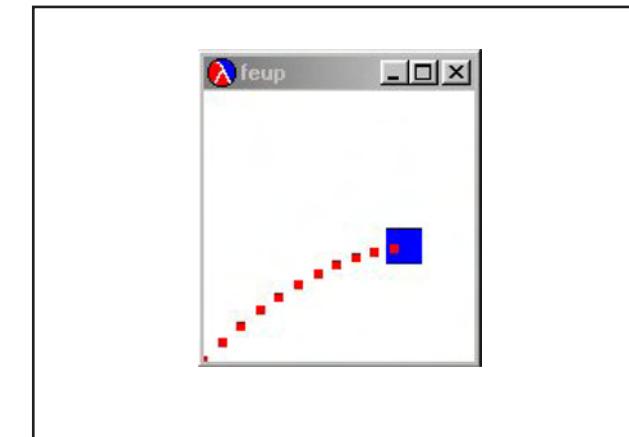
A simulação do lançamento de projécteis retoma um exercício já considerado num outro módulo, mas agora num ambiente de interacção com saída gráfica. A simulação do lançamento de projécteis inicia-se chamando o procedimento `projectil`, em que os dois primeiros parâmetros constituem, respectivamente, a largura e a altura da janela onde se visualizará o movimento do projéctil. O terceiro argumento, uma cadeia de caracteres, corresponde ao título da janela. Uma janela gráfica é imediatamente criada, seguindo-se uma pergunta sobre se se pretende ou não lançar um projéctil. À resposta positiva corresponderá, na janela criada, a visualização de um objecto-alvo numa posição determinada aleatoriamente. O utilizador do simulador é então interrogado sobre os valores iniciais do ângulo e velocidade de um projéctil, colocado no canto inferior esquerdo da janela, para tentar alcançar o objecto-alvo. A trajectória do projéctil começa a ser visualizada na janela até atingir o objecto-alvo, ou então até atingir o solo sem lhe acertar.

Para melhor se entender a especificação do simulador, deve analisar o diálogo e as figuras que se seguem.

```
> (projectil 150 150 "feup")
pretende lançar projectil (n = não; outro- sim):
s
Angulo de partida (0 < ang <= 90):
45
Velocidade inicial (> 0)):
50
Boa-pontaria...
```

```
pretende lançar projectil (n = não; outro- sim):
s
Angulo de partida (0 < ang <= 90):
45
Velocidade inicial (> 0)):
50
Ma' -pontaria...
```

```
pretende lançar projectil (n = não; outro- sim):
n
A simulação vai terminar
```



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



- 1- Desenvolva em Scheme o simulador descrito, tendo em conta que as suas principais tarefas poderão ser:
  - criar uma janela gráfica de dimensão L x A (L e A são representados pelos dois primeiros parâmetros);
  - perguntar se quer ou não lançar o projéctil (se não quer, termina o programa);
  - limpar a janela gráfica corrente (não seria necessário da primeira vez, mas é mais fácil considerá-lo sempre);
  - visualizar o objecto-alvo, num ponto da janela determinado aleatoriamente;
  - pedir o ângulo ( $0 < \text{ang} \leq 90$ ) e a velocidade ( $\text{vel} > 0$ ) iniciais;
  - lançar e visualizar o projéctil a partir do canto inferior-esquerdo e, ao mesmo tempo, verificar se o objecto-alvo é atingido. Esta tarefa termina se o alvo for atingido ou se o projéctil cair no solo sem o atingir;
  - visualizar a mensagem de acordo com o facto de ter atingido ou não o objecto-alvo;
  - retomar o ponto em que pergunta se quer ou não lançar o projéctil.



Desenvolva e teste o projecto lançamento de projécteis.  
Siga uma aproximação de-cima-para-baixo.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

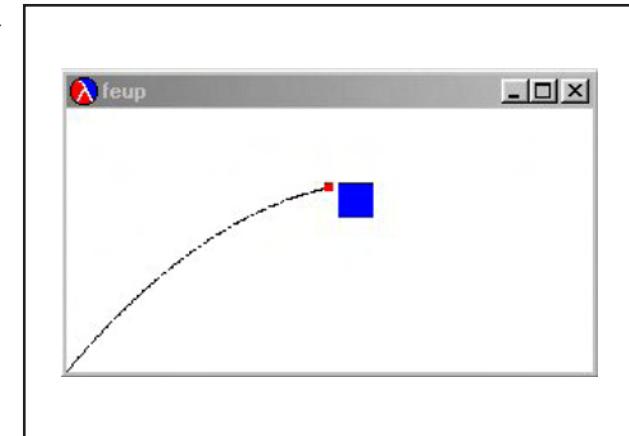
## ANEXOS

2- Numa segunda versão do programa, deve melhorar o simulador apresentado, introduzindo as seguintes alterações:

- a trajectória termina se o projétil ultrapassar o lado direito da janela;
- a deslocação do projétil não se faz por saltos, mas de uma forma mais contínua, como se indica na figura.

Assim, repetidamente:

- O projétil é visualizado numa posição e, passado um curto intervalo de tempo, é apagado;
- Nesta posição, é visualizado um ponto que define o rastro do projétil;
- O projétil avança para uma nova posição da trajectória, muitíssimo perto da posição anterior;



### Pista

Para obter o efeito de dinamismo, visualize o projétil com a cor 'temp' (cor temporária - ver Anexo B)



Desenvolva e teste o projecto lançamento de projécteis numa versão melhorada, tendo já por base a solução da versão anterior.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Módulo 3.1 - Abstracção de dados para captar as características dos problemas

Os tipos de dados fornecidos pelas linguagens de programação estão longe de se adequarem a todas as situações que nos surgem, o que implica a criação de novos tipos de dados de acordo com as características de cada problema. Numa situação que envolva formas gráficas a duas dimensões (2D), é perfeitamente justificável a criação de um novo tipo de dados, o ponto-2D, constituído por dois valores numéricos que representam as coordenadas x e y de um ponto no plano. A partir desse momento, as extremidades de um segmento de recta ou os vértices de um triângulo podem ser vistos como um objecto do tipo ponto-2D e não como dois valores numéricos correspondentes às suas coordenadas.

Um novo tipo de dados que se ajuste às características de um certo problema, acompanhado das operações básicas respectivas, constitui uma espécie de barreira sob a qual se encontra o que é essencial para trabalhar com os dados desse problema. É a isto que se chama abstracção de dados.

Uma barreira esconde os pormenores de implementação da abstracção de dados. Ou seja, alterações à implementação da abstracção não devem ter repercussões sobre outros códigos que tenham sido desenvolvidos sobre ela.

O par é uma entidade do Scheme que permite criar abstracções de dados. O par é acompanhado de operadores do tipo construtor, para criar objectos novos, e selector, para dar acesso às partes de um objecto.

A abstracção racional permite criar e manipular fracções e é proposta como um exercício cujo objectivo fundamental é ajudar a consolidar este conceito importante que é a abstracção de dados.

Na abstracção tartaruga, os objectos que se podem criar e manipular são tartarugas. Sobre esta abstracção, entre tanto disponibilizada num directório do DrScheme, é proposto um projecto de um jogo com alguma complexidade, designado por a tartaruga escondida.

### Palavras-Chave

Abstracção de dados, barreira da abstracção, par, construtor, selector.



# **S C H E M E**

## **na descoberta da programação**

### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

R E 1 2 3 4

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

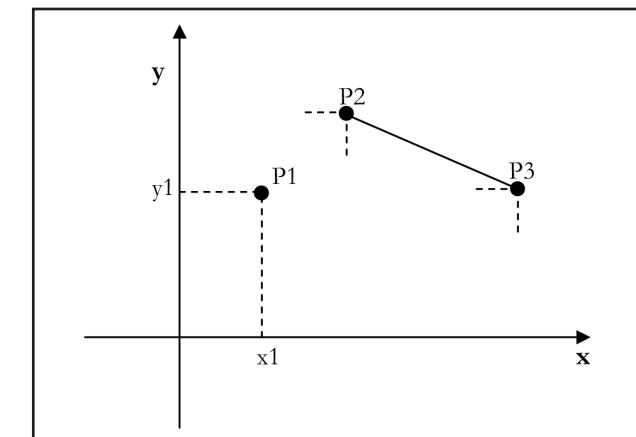
## **Um ponto no plano pode ser representado por um par!**

Os números e os booleanos surgem no Scheme de uma forma natural e estes dois tipos de dados adaptam-se a muitas situações. Estamos a falar de dados simples. Com um objecto numérico, um número, podemos representar, por exemplo, um contador (quantas pessoas entraram numa sala? 30 pessoas? retirar ou acrescentar 1 ao número de pessoas na sala), com um booleano podemos representar o estado de um objecto (a porta está aberta? Sim ou não, verdadeiro ou falso e, na linguagem Scheme, #t ou #f).

Não será difícil imaginar situações em que não é prático utilizar dados simples e que obrigam à criação de dados compostos, dados constituídos por várias partes. Estas várias partes poderão ser dados simples, como números e booleanos, mas também outros dados compostos entretanto definidos. Perante as características de um problema, é conveniente compor os dados mais adequados a cada caso. Por exemplo, num problema de características gráficas a duas dimensões (2D), é perfeitamente justificável a criação de um novo tipo de dados, o ponto-2D, constituído por dois valores numéricos que representam as coordenadas x e y de um ponto no plano.

É o que acontece na figura com os pontos P1, P2 e P3, qualquer um deles composto por 2 valores numéricos, que representam as coordenadas x e y.

P1 não é visto como dois números, mas sim como um objecto único com duas partes, as suas coordenadas x e y, representadas cada uma delas por um valor numérico. Ainda na figura pode identificar outro tipo de objecto, segmento-de-recta, também um objecto único composto por duas partes, os seus vértices, representadas cada uma delas por um dado do tipo ponto-2D.



**Ainda no contexto dos gráficos, a manipulação de áreas triangulares colocadas num plano, poderia levar à criação do tipo de dados triangulo.**

**Qual seria a composição deste novo tipo de dados?**





Um outro exemplo que pode considerar são as fracções, às quais pode associar o tipo de dados `fraccao`. Qual seria a composição deste novo tipo de dados?

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

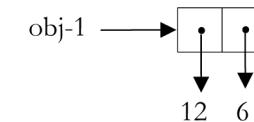
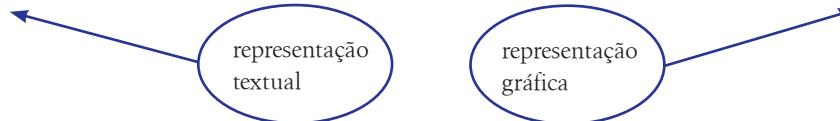


Um novo tipo de dados que se ajuste às características de um certo problema, acompanhado das operações básicas respectivas, constitui uma espécie de barreira sob a qual se encontra o que é essencial para trabalhar com os dados desse problema. É a isto que se chama abstracção de dados. Por exemplo, se a operação de adição de duas fracções fizer parte da abstracção `fraccao`, bastar-lhe-á saber que esta operação recebe dois objectos do tipo `fraccao` e devolve um objecto do mesmo tipo que corresponde à soma daqueles dois. Não precisa de saber como estão organizadas as partes constituintes de uma fração ou como são somadas duas frações, pois tem acesso à abstracção `fraccao` que reconhece objectos e as operações associadas ao tipo `fraccao`.

### O par em Scheme para criar abstracções de dados

Com o Scheme pode criar um par, um objecto constituído por duas partes, com o procedimento primitivo `cons`.

```
> (define obj-1 (cons 12 6))
> obj-1
(12 . 6)
```



O objecto `obj-1` surge em duas representações, uma textual e outra gráfica, esta última designada por caixa-e-apontador. Em ambas é notório o agrupamento de dois valores, constituindo uma entidade única. O objecto `obj-1` é um dado composto.

O procedimento `cons` é um construtor do Scheme, pois aceita as componentes que vão formar um objecto e depois constrói e devolve o objecto construído, sob a forma de um par.

INTRODUÇÃO
1 - O ESSENCIAL DO SCHEME
2 - RECURSIVIDADE
<b>3 - ABSTRACÇÃO DE DADOS</b>
R E 1 2 3 4
4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE
5 - ABSTRACÇÕES COM DADOS MUTÁVEIS
6 - SCHEME E OUTRAS TECNOLOGIAS
7-EXERCÍCIOS E PROJECTOS
ANEXOS



E tendo um par, como aceder a cada uma das suas partes?

A parte da esquerda e a parte da direita?

Os procedimentos primitivos car e cdr são selectores, pois aceitam um par e seleccionam uma das partes desse par.

```
> (car obj-1)  
12  
> (cdr obj-1)  
6
```



Teste os procedimentos primitivos cons, car e cdr.

Experimente a criação de alguns pares e o acesso às partes respectivas.

Faça o mesmo tipo de experiências com pares que tenham outros pares na sua constituição.

As regras de cálculo destes três procedimentos primitivos relacionados com o par do Scheme são agora apresentadas.

- (cons obj1 obj2)  
Aceita 2 objectos como argumentos e devolve o par constituído por eles. Trata-se de um construtor.
- (car par)  
Aceita um par como argumento e devolve a parte esquerda do par. Trata-se de um selector.
- (cdr par)  
Aceita um par como argumento e devolve a parte direita do par. Trata-se de um selector.



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



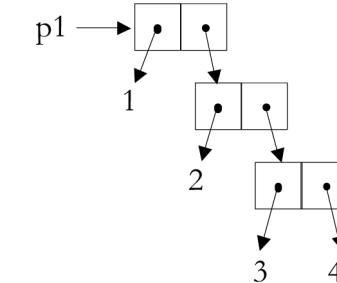
Se p1 for definido como sendo

```
(cons 1 (cons 2 (cons 3 4)))
```

a sua representação caixa-e-apontador é equivalente ao indicado na figura.

Analise a expressão do interior para o exterior e encontrará o par constituído por 3 e 4, ou seja  $(3 . 4)$ , e a seguir o par cuja parte esquerda é 2 e a parte direita é  $(3 . 4)$ , ou seja  $(2 . (3 . 4))$ .

Finalmente, verifique que o par p1 é constituído por 1 e pelo par  $(2 . (3 . 4))$ , o que equivale a  $(1 . (2 . (3 . 4)))$ .



```
> (define p1 (cons 1 (cons 2 (cons 3 4))))  
> p1  
(1 2 3 . 4)
```



**Surpreendido com a representação textual de p1?  
Que resultado esperava?**

O Scheme elimina o ponto e o parêntesis respectivo, quando o elemento do lado direito de um par é também um par. Por este motivo, a representação textual de p1 será, sucessivamente,

```
(1 . (2 . (3 . 4)))    -->    (1 2 . (3 . 4))    -->    (1 2 3 . 4)
```

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



Verifique, sem a ajuda do Scheme, se as respostas indicadas estão ou não correctas.

```
> (define p1 (cons 1 (cons 2 (cons 3 4))))  
> p1  
(1 2 3 . 4)  
> (car p1)  
1  
> (cdr p1)  
(2 3 . 4)  
> (car (cdr p1))  
2  
> (cdr (cdr p1))  
(3 . 4)  
> (car (cdr (cdr p1)))  
3  
> (cdr (cdr (cdr p1)))  
4
```



Verifique no Scheme a correcção das suas respostas.



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

### Exercício 1

Se  $p_2$  for definido como sendo `(cons 35 (cons 5 7))`, comece por desenhar a sua representação caixa-e-apontador e determine o resultado das expressões que se seguem.

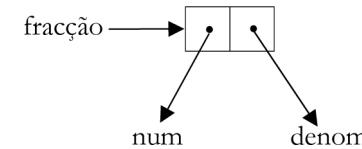
```
> p2
??
> (car p2)
??
> (car (cdr p2))
??
> (cdr (cdr p2))
??
```



Verifique no Scheme a correcção das suas respostas.

**Exercício 2** - Um exercício para consolidar o conceito  
Abstracção de dados

Vamos abordar um exercício para consolidar o conceito de abstracção de dados, sobre o tema números racionais ou fracções. O objectivo é definir e desenvolver a abstracção racional que permita criar e manipular fracções. Fracções com o seu numerador e denominador, que possam ser visualizadas no formato numerador/denominador.



A modelação computacional de uma fração pode ser um par.  
Concorda?  
Vê alguma alternativa?



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

As operações desejadas para a abstracção racional são seguidamente apresentadas.

- (`raccons num denom`)  
com dois parâmetros inteiros `num` e `denom`, devolve uma fracção cujo numerador é `num` e o denominador é `denom`. Trata-se de um construtor.
- (`racnum rac`)  
com um parâmetro do tipo racional, devolve um inteiro correspondente ao seu numerador. Trata-se de um selector.
- (`racdenom rac`)  
com um parâmetro do tipo racional, devolve um inteiro correspondente ao seu denominador. Trata-se de um selector.
- (`racvisu rac`)  
com um parâmetro do tipo racional, visualiza a fracção `rac` no formato numerador/denominador. Trata-se de um selector.
- (`racreduz rac`)  
com um parâmetro do tipo racional, devolve uma fracção equivalente a `rac` mas na forma reduzida. Por exemplo, se receber como argumento a fracção  $2/4$ , devolve uma fracção  $1/2$ . No entanto, se receber a fracção  $1/2$  devolve uma fracção  $1/2$ . Trata-se também de um construtor.

Implemente em Scheme a abstracção racional.



Desenvolva a abstracção racional.

Com a abstracção racional desenvolvida, crie fracções, visualize-as e experimente reduzi-las.

Complete a abstracção racional com as operações que se seguem.

Deve começar por especificá-las melhor, pois apenas se fornece o respectivo nome.

`rac+`, `rac-`, `rac*`, `rac/`, `rac=`, `rac>`, `racmax`, e `racpositive?`.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## A abstracção tartaruga

Com um exemplo, vamos continuar a explorar a capacidade dos procedimentos primitivos `cons`, `car` e `cdr` para criar e utilizar dados compostos, qualquer que seja a sua complexidade.

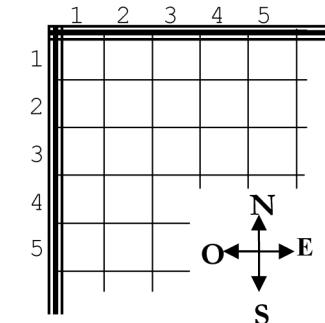
Vamos imaginar umas tartarugas coloridas que se movem num tabuleiro, num dos quatro sentidos: Norte, Sul, Este e Oeste. Notar que o tabuleiro, sem limites à direita e em baixo, tem limites à esquerda, a coluna 1, e em cima, a linha 1.

Pretende-se simular a movimentação de tartarugas no tabuleiro, nos quatro sentidos.

Sabe-se que na movimentação para Oeste, a tartaruga não ultrapassa a coluna 1 e, analogamente, na movimentação para Norte, não ultrapassa a linha 1.

Nesta altura, terá pela frente algumas dificuldades, nomeadamente, saber como criar ou modelar computacionalmente entidades do tipo tartaruga, especificando as suas características de cor e posição no tabuleiro e, posteriormente, saber o valor de cada uma dessas características. Será também interessante saber como visualizar no ecrã, de uma forma clara, todas as características de uma tartaruga. Com tudo isto em mente, pode imaginar um conjunto de procedimentos para criar e processar entidades do tipo tartaruga.

- (`faz-tartaruga col lin cor`)  
com três parâmetros, inteiros positivos, `col`, `lin` e `cor`, devolve uma tartaruga situada na intersecção da coluna `col` com a linha `lin`, colorida com `cor`. Trata-se de um constructor.
- (`linha-tar tar`)  
com um parâmetro do tipo tartaruga, devolve um inteiro positivo que identifica a linha onde se encontra a tartaruga `tar`. Trata-se de um selector.
- (`coluna-tar tar`)  
com um parâmetro do tipo tartaruga, devolve um inteiro positivo que identifica a coluna onde se encontra a tartaruga `tar`. Trata-se de um selector.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

- (cor-tar tar)  
com um parâmetro do tipo tartaruga, devolve um inteiro positivo que identifica a cor da tartaruga tar. Trata-se de um selector.
- (visu-tar tar)  
com um parâmetro do tipo tartaruga, visualiza no ecrã as suas características, coluna, linha e cor. De uma certa maneira, trata-se também de um selector.

De facto, acabámos de especificar um novo tipo de dados, para além dos que o Scheme disponibiliza (inteiros, booleanos e outros), designado por tartaruga, ao qual está associado um conjunto de operações.



**Este novo tipo, tartaruga, é um exemplo de uma abstracção de dados.  
Depois de implementada, em que tipo de problemas pode ser útil? Justifique.**

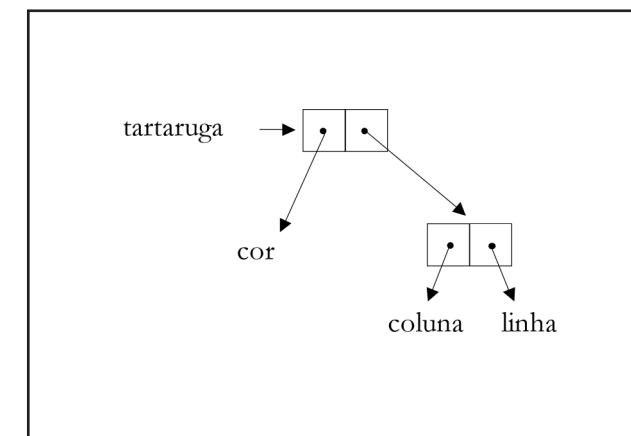
Para tornar utilizável a abstracção tartaruga é necessário implementá-la, o que vai ser feito em Scheme.

Em primeiro lugar, vamos decidir como modelar os dados que representam uma tartaruga. Entre várias hipóteses, optou-se por utilizar um par, em que a parte da esquerda é a cor da tartaruga e a parte da direita é, ela própria, também um par, onde se encontram os dados correspondentes à posição da tartaruga no tabuleiro, coluna e linha.

Com esta decisão tomada, já é possível definir os procedimentos associados à abstracção tartaruga:

O construtor faz-tartaruga e os selectores linha-tar, coluna-tar, cor-tar e visu-tar.

```
(define faz-tartaruga      ; 3 parâmetros, inteiros positivos
  (lambda (col lin cor)    ; devolve uma tartaruga situada em col
    (cons cor (cons col lin)))) ; e lin, com a cor dada. É um construtor.
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(define linha-tar ; parâmetro do tipo tartaruga
  (lambda (tart) ; devolve inteiro positivo que identifica a linha
    (cdr (cdr tart)))) ; onde se encontra a tartaruga. É um selector.
```



Desenvolva os selectores que faltam.

Depois de desenvolvida a abstracção tartaruga, experimente criar tartarugas e aceder às suas características utilizando os procedimentos da abstracção.

Um pequeno teste com a abstracção tartaruga.

```
> (define tartaruga-teste (faz-tartaruga 1 30 51))
> tartaruga-teste
(51 1 . 30)
> (linha-tar tartaruga-teste)
30
> (coluna-tar tartaruga-teste)
1
> (cor-tar tartaruga-teste)
51
```

forma pouco interessante  
de visualizar uma tartaruga

Quanto ao procedimento visu-tar, com um parâmetro do tipo tartaruga, visualiza a respectiva posição e cor.

```
> (visu-tar tartaruga-teste)
tartaruga em col: 1, lin: 30, cor: 51

(define visu-tar
  (lambda (tart)
    (display "tartaruga em col: ")
    (display (coluna-tar tart))
    (display ", lin: ")
    (display (linha-tar tart))
    (display ", cor: ")
    (display (cor-tar tart))
    (newline)))
```

Notar no procedimento visu-tar, a utilização dos selectores associados às entidades do tipo tartaruga.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Utilizar a abstracção tartaruga para simular o movimento de tartarugas

Nesta altura, já tem na mão a abstracção tartaruga que lhe permite criar objectos do tipo tartaruga, aceder às suas características e visualizá-las no ecrã. Esta abstracção pode ser vista como uma espécie de barreira que esconde os pormenores de implementação, pois para a utilizar basta-lhe conhecer as operações que lhe estão associadas. E é isso que vamos passar a fazer, para desenvolver um conjunto de operações que simulam o movimento de tartarugas de n posições em qualquer uma das quatro direcções, com as limitações impostas pela coluna 1 e linha 1, conforme foi referido.



**Uma questão importante, mas com uma resposta nada simples...**

**Na abstracção tartaruga, para além dos selectores, que nada alteram, resta apenas o construtor faz-tartaruga que também nada altera mas cria tartarugas novas. Assim, por exemplo, a simulação do movimento de uma tartaruga x células na direcção Norte será traduzido pela criação de uma nova tartaruga x células mais à frente, na direcção Norte, esquecendo a tartaruga original.**

**Criar um novo objecto para simular um objecto que se modifica pode ser extremamente penalizador.**

**Porquê? Como evitar este problema?**

Comecemos por criar a tartaruga tar-10.

```
>(define tar-10 (faz-tartaruga 5 7 10))  
>(visu-tar tar-10)  
tartaruga em col: 5, lin: 7, cor: 10
```

No exemplo que se segue, é criada uma tartaruga a partir de tar-10, movimentando-a 3 posições para Norte.

```
>(define tar-20 (vai-para-norte tar-10 3))  
>(visu-tar tar-20)  
tartaruga em col: 5, lin: 4, cor: 10
```

A ideia para conseguir movimentar uma tartaruga será criar outra tartaruga com a cor da tartaruga inicial, mas posicionada de acordo com o movimento pedido. De facto, assim deverá ser pois não dispomos, para já, de qualquer operação para modificar uma entidade anteriormente criada.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
(define vai-para-norte ; ----- deslocamento para Norte de n posições.  
  (lambda (tart n)  
    (let ((destino (- (linha-tar tart)  
                      n))) ; de facto, o que acontece é definir  
      (faz-tartaruga (coluna-tar tart)) ; uma nova tartaruga com a mesma cor  
      (if (< destino 1) ; e na mesma coluna e numa linha mais  
          1 ; acima, não ultrapassando a linha 1  
          destino)  
        (cor-tar tart))))  
  
(define vai-para-sul ; ----- deslocamento para Sul de n posições.  
  (lambda (tart n)  
    (faz-tartaruga (coluna-tar tart) ; neste caso, não há limitações ao  
    (+ (linha-tar tart) n) ; deslocamento.  
    (cor-tar tart))))
```



Estes dois procedimentos, **vai-para-norte** e **vai-para-sul**, são selectores ou construtores? Justifique.



Desenvolva **vai-para-oeste** e **vai-para-este**.

Experimente criar tartarugas, simular movimentá-las nas várias direcções e visualizar os resultados.

É importante verificar que os quatro procedimentos que simulam a movimentação de tartarugas são todos construtores. Todos eles terminam com uma chamada ao construtor **faz-tartaruga**, que devolve uma tartaruga, com características da tartaruga dada, mas situada noutro local.

### Utilizar a abstracção tartaruga para calcular distâncias entre tartarugas

Ainda sobre a abstracção tartaruga vai ser agora desenvolvido um conjunto de operações que calculam as distâncias entre duas tartarugas, distâncias que são medidas em número de células entre elas.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

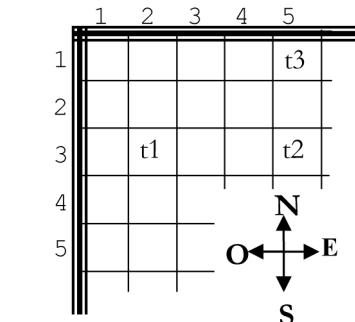
## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Na figura, entre as tartarugas t1 e t3, a distância-horizontal é 3, a distância-vertical é 2 e a distância é 5, esta última correspondente à soma daquelas duas.

```
> (define t1 (faz-tartaruga 2 3 10))
> (define t2 (vai-para-este t1 3))
> (visu-tar t1)
tartaruga em col: 2, lin: 3, cor: 10
> (visu-tar t2)
tartaruga em col: 5, lin: 3, cor: 10
> (define t3 (vai-para-norte t2 6))
> (visu-tar t3)
tartaruga em col: 5, lin: 1, cor: 10
> (distancia t1 t3)
5
```



Na definição dos procedimentos que se seguem, utilizam-se selectores da abstracção tartaruga.

```
(define distancia-horizontal
  (lambda (tart1 tart2) ; distância horizontal entre tart1 e tart2
    (abs (- (coluna-tar tart1) ; dada pelo valor absoluto da diferença
             (coluna-tar tart2))))) ; entre as colunas de tart1 e tar2
```



Desenvolva `distancia-vertical` e `distancia`.

Experimente criar tartarugas, simular movimentá-las nas várias direcções e calcular as distâncias entre elas.

**Especialmente para quem tiver dúvidas que uma abstracção funciona como uma barreira**

Este tema é apresentado sob a forma de um exercício. Cabe-lhe a si a sua exploração.

# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

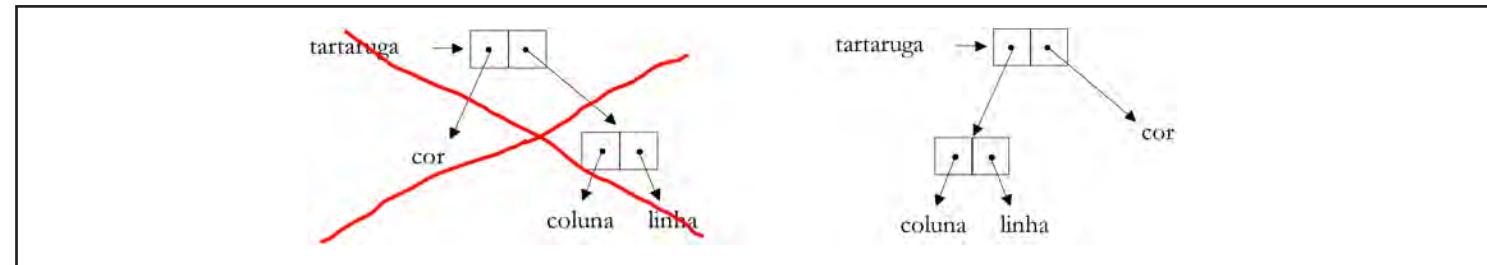
7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



### Exercício 3

Pretende-se alterar a estrutura de dados do tipo tartaruga, como se indica na figura. Do lado esquerdo está indicada a modelação utilizada até agora e do lado direito a que se pretende utilizar.



Desenvolva as novas versões dos procedimentos básicos para manipulação de objectos do tipo tartaruga: `faz-tartaruga`, `linha-tar`, `coluna-tar`, `cor-tart`, `visu-tar`.

Experimente visualizar tartarugas criadas com o construtor `faz-tartaruga` e também com os procedimentos de deslocação.



No contexto da nova implementação da abstracção tartaruga...

- 1 - Verifique que um dos procedimentos básicos da abstracção tartaruga não sofre qualquer alteração!  
Identifique-o e apresente uma justificação para esse facto.
- 2 - Verifique que os procedimentos que simulam os movimentos de tartarugas num tabuleiro continuam completamente funcionais, não requerendo qualquer alteração. Apresente também uma justificação para esse facto.
- 3 - E o que acontece com os procedimentos para cálculo das distâncias, serão também eles independentes da nova modelação da tartaruga?

A abstracção tartaruga comportou-se, de facto, como uma barreira, pois escondeu a sua implementação. Esta implementação foi alterada, mas, ao manter a sua interface, garantiu a validade do código que a utilizava.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

### Exercício 4

Sobre a abstracção tartaruga, escreva em Scheme os predicados (`cima? tar1 tar2`), (`baixo? tar1 tar2`), (`direita? tar1 tar2`) e (`esquerda? tar1 tar2`), todos eles com dois parâmetros do tipo `tartaruga` e que indicam, respectivamente, se `tar1` está acima, abaixo, à direita ou à esquerda de `tar2`.

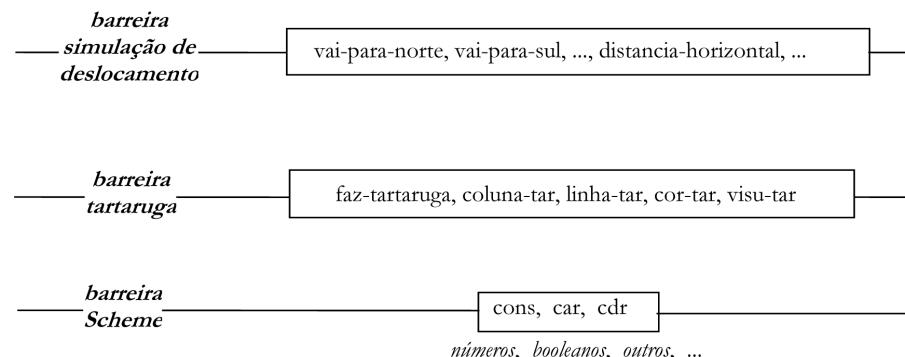


Desenvolva os predicados pedidos.

Experimente criar tartarugas e verifique as suas posições relativas.

São várias as vantagens que surgem quando se usa uma abstracção de dados, vantagens que são bem visíveis mesmo num exemplo tão simples como o que acaba de se apresentado.

1. Com o construtor e os selectores da abstracção tartaruga, desenvolveram-se os procedimentos que simulavam o deslocamento das tartarugas num tabuleiro, sem ser necessário saber como é que uma tartaruga era por dentro. Bastou apenas conhecer as características do problema e usar a respectiva linguagem, neste caso, saber que uma tartaruga está numa linha, numa coluna e que tem uma cor.
2. Viu que a alteração da estrutura de dados dos objectos do tipo `tartaruga` apenas teve repercussão sobre os procedimentos básicos, construtor e selectores. Todos os outros mantiveram-se funcionais, graças às barreiras que se criaram entre os vários níveis de abstracção.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A barreira colocada ao nível mais baixo, barreira Scheme, corresponde ao que nos é disponibilizado pelo Scheme. Sobre ela definiu-se a abstracção tartaruga a que corresponde a barreira tartaruga.

Sobre esta abstracção, construiu-se a abstracção simulação de deslocamento, onde se simulam os deslocamentos das tartarugas e se calculam as distâncias entre elas.

**Exercício 5** - desafio já com alguma dificuldade...

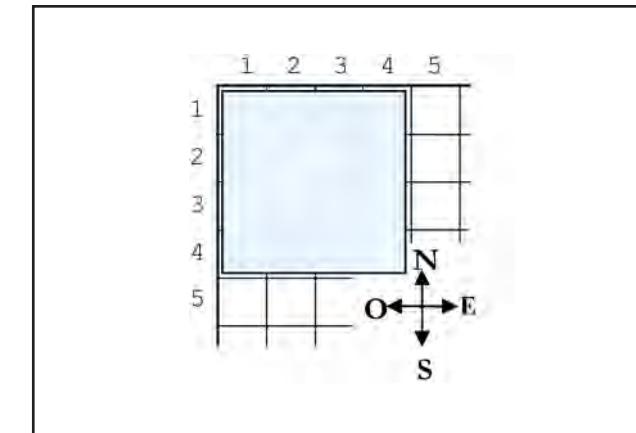
Projecto - a tartaruga escondida

Tendo por base as abstracções tartaruga e simulação de deslocamento apresentadas anteriormente, desenvolva um pequeno programa que reveste a forma de um jogo, conforme a descrição que se segue.

O computador esconde uma tartaruga numa célula de um tabuleiro e coloca, também nesse tabuleiro, uma outra tartaruga, que irá procurar a que está escondida. As células em que as duas tartarugas são colocadas são escolhidas aleatoriamente. Depois desta operação, o computador indica a posição de uma das tartarugas e a distância a que se encontrada da tartaruga escondida. Por seu lado, o jogador vai encaminhando a tartaruga na procura da tartaruga escondida, utilizando os movimentos nas 4 direcções possíveis.

A cada movimento, o computador volta a indicar a nova posição da tartaruga que procura e a distância entre as duas tartarugas. E tudo isto se repete até que a distância se anule.

O procedimento principal, em que se estrutura o programa pedido, designado por `tartaruga-escondida`, tem apenas um parâmetro que representa o lado de uma porção quadrada do tabuleiro, medido em células, cujo vértice superior-esquerdo localiza-se na célula definida pela coluna 1 e linha 1. É nesse quadrado onde o computador coloca aleatoriamente as duas tartarugas. A figura põe em relevo o quadrado, de lado 4.



O jogador poderá conduzir a tartaruga para além das fronteiras da zona quadrada, segundo as regras definidas para a movimentação das tartarugas sobre o tabuleiro, mas será tempo perdido, pois nessas tentativas não encontra a tartaruga escondida.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

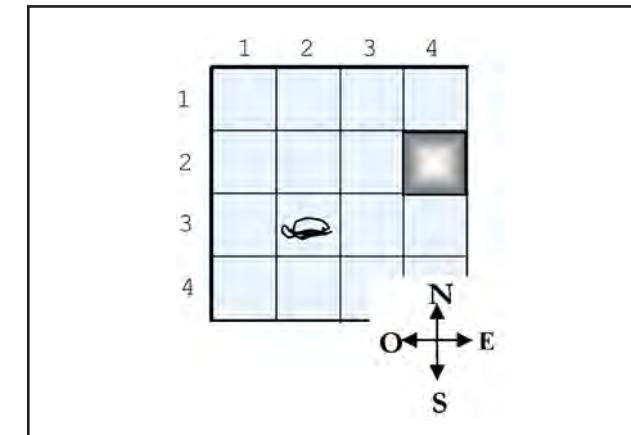
Uma possível sessão deste jogo é seguidamente apresentada, na qual a chamada de tartaruga-escondida vai acompanhada do argumento 4. Vamos imaginar que o computador colocou aleatoriamente a tartaruga que procura na célula (coluna 2, linha 3) (informação divulgada no início da sessão) e a outra na célula (coluna 4, linha 2) (informação que se mantém secreta).

```
> (tartaruga-escondida 4)
tartaruga em col: 2, lin: 3, cor: 10
distancia entre as tartarugas e': 3
```

```
jogada 1
direccao (0-N, 1-S, 2-E, 3-O): 0
deslocamento: 2
tartaruga em col: 2 e lin: 1, cor: 10
distancia entre as tartarugas e': 3
```

```
jogada 2
direccao (0-N, 1-S, 2-E, 3-O): 2
deslocamento: 2
tartaruga em col: 4 e lin: 1, cor: 10
distancia entre as tartarugas e': 1
```

```
jogada 3
direccao (0-N, 1-S, 2-E, 3-O): 1
deslocamento: 1
tartaruga em col: 4 e lin: 2, cor: 10
distancia entre as tartarugas e': 0
Terminou a procura...
```



Este problema surge como um desafio de complexidade média, sugerindo uma abordagem do tipo de-cima-para-baixo, abordagem em que o problema é decomposto sucessivamente em sub-problemas cada vez mais simples. Recorda-se que nesta abordagem são identificadas várias fases: especificação do problema, desenvolvimento manual de situações do problema, ideia de solução, algoritmo com identificação dos passos (ou sub-problemas) mais complexos, codificação numa linguagem de programação (em Scheme) e teste.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



A abstracção tartaruga e a abstracção simulação de deslocamento ficam acessíveis no seu programa com a integração de uma cópia de código Scheme apresentado no decorrer deste módulo. Todavia recomenda-se uma outra forma de tratar situações semelhantes, em que existe código reutilizável. De facto, uma abstracção é um código com grande probabilidade de vir a ser reutilizado na resolução de vários problemas. Este tema é tratado no [Anexo C - Reutilização de código](#), no entanto, a consulta deste anexo é, neste momento dispensável, se aceitar a explicação que se segue.

A reutilização de código requer a colocação de um ficheiro com esse código num directório a criar em `PLT\collects\` do Scheme. Para os nossos trabalhos, o directório criado chama-se `user-feup` onde, entre outros, se encontram os ficheiros `tartaruga.scm` e `simula-descola.scm` com as abstracções referidas. Em vez de copiar o código Scheme para ter acesso a estas abstracções, bastará escrever no início do seu programa

```
(require (lib "tartaruga.scm" "user-feup"))
```

para usufruir das operações

```
faz-tartaruga, linha-tar, coluna-tar, cor-tar, visu-tar
```

e escrever

```
(require (lib "simula-desloca.scm" "user-feup"))
```

para usufruir das operações

```
vai-para-norte, vai-para-sul, vai-para-oeste, vai-para-este,  
distancia-horizontal, distancia-vertical, distancia.
```



Desenvolva e teste o projecto a tartaruga escondida, tendo já disponíveis as abstracções tartaruga e simulação de deslocamento.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Módulo 3.2 - Colocar objectos em sequência

Em relação aos dados, nem sempre as situações que encontraremos se resolvem com dados simples. Em certos casos é necessário colocar objectos em sequência, o que requer dados que são compostos por outros dados. O par do Scheme permite compor dados com qualquer nível de complexidade, mas a lista é a forma que o Scheme privilegia para implementar sequências. A lista pode ser vista como um par, herdando por isso toda a funcionalidade deste, quer em termos de construção de objectos quer em termos de selecção dos seus componentes. Mas a lista aparece com uma funcionalidade muito mais rica que a do par. Neste módulo, a introdução à funcionalidade da lista é feita ao mesmo tempo que se estuda uma forma de a implementar e aproveitar, assim, para ir treinando a programação com listas.

Para além dos números, booleanos, pares e listas, aparecem agora os símbolos como um novo tipo de dados muito adequado em determinadas situações. Por exemplo, representar o mês de Janeiro pelo número 1 ou pelo símbolo `janeiro` não é bem a mesma coisa, a legibilidade do símbolo é muito maior.

A sequência de dados ou de objectos pode envolver elementos que são, eles próprios, sequências e é assim que aparece a lista de listas. Surge, desta forma, a necessidade de processar a lista em profundidade, ou seja, a necessidade de percorrer a lista e explorar a estrutura de cada um dos seus componentes.

### Palavras-Chave

Sequência de dados ou de objectos, par, lista, construtores de listas, selectores de listas.



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

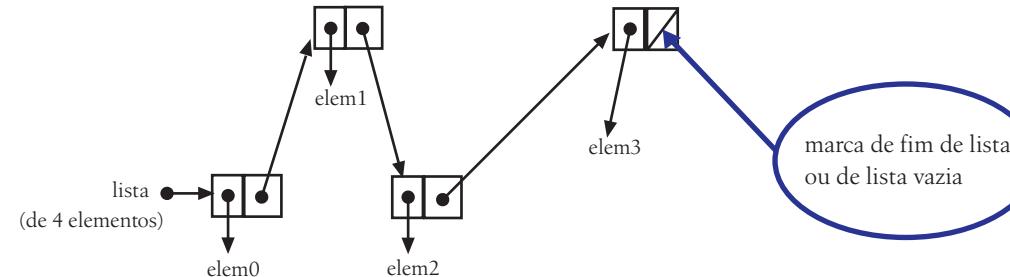
6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

Consegue imaginar uma sequência de objectos como se fosse um par?

A lista é uma das formas de representar sequências de objectos, sejam eles números, booleanos, pares ou outro tipo de dados.



A figura representa uma lista de 4 elementos, em que o primeiro foi designado por elem0. A lista dá acesso privilegiado ao primeiro elemento, bastando para tal fazer (car lista). Justifique esta expressão de acesso ao primeiro elemento da lista.

O primeiro elemento da lista é a parte esquerda do primeiro par.

Repare que a parte direita deste primeiro par, por seu lado, é uma lista.

Qual é a expressão que dá acesso a esta lista? Justifique.

Quantos elementos tem esta lista? Justifique.

E qual é a expressão que dá acesso ao segundo elemento da lista representada na figura?

"Consegue imaginar uma sequência de objectos como sendo um par?"

Isto constitui o título desta secção!

Apresente agora, ou procure nas próximas páginas, uma ideia que sustente a este título.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

As listas constituem um meio poderoso e flexível para compor e manipular dados de qualquer nível complexidade, aliás como já acontecia com os pares. No entanto, com as listas, estas operações tornam-se mais naturais e mais fáceis de utilizar. Bastará dizer que o Scheme deriva da linguagem Lisp, e que este nome vem de *List Programming*, para poder imediatamente antever a relevância das listas em tudo o que se seguirá.

### Junte um elemento a uma lista existente para criar uma nova lista

A expressão correspondente a uma lista vazia é (). No entanto, se o Scheme que utiliza não aceitar esta expressão, tente outra parecida ' () , cujo significado veremos um pouco mais à frente.

```
> (define lista-vazia ())  
> lista-vazia  
()
```

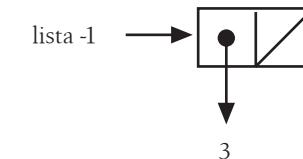


Um par de parêntesis, sem qualquer elemento entre eles, é uma expressão que representa uma lista vazia.  
Imagine como será a expressão que representa uma lista com os elementos 1, 2, 3 e 4.

Para criar uma lista de um elemento, junte esse elemento a uma lista vazia. A partir da lista obtida pode criar uma lista de dois elementos juntando-lhe mais um elemento. De uma forma geral, pode criar uma lista de  $m+1$  elementos com o procedimento primitivo `cons`, juntando um elemento a uma lista de  $m$  elementos.

```
> (define lista-1 (cons 3 lista-vazia))  
> lista-1  
(3)
```

Ao juntar o número 3 a uma lista vazia criou a lista `lista-1`, a que corresponde a representação textual (3).



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

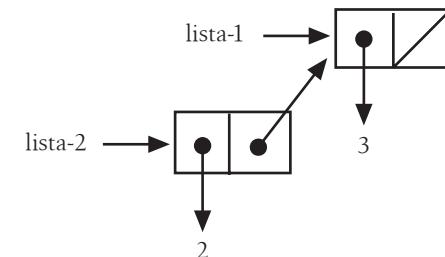
## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

A representação caixa-apontador de lista-1 mostra um par, cuja parte esquerda é o número 3 e cuja parte direita mostra uma diagonal que significa fim-de-lista.

À lista designada por lista-1 junte-lhe agora o valor numérico 2, dando origem a lista-2.

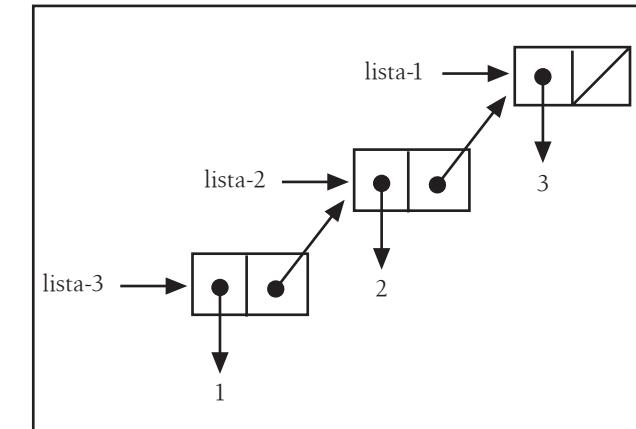
```
> (define lista-2 (cons 2 lista-1))  
> lista-2  
(2 3)
```



Verifique que esta lista-2 pode ser vista como um par, em que a parte esquerda é o número 2 e a parte direita é lista-1.

Mas, de uma forma até mais natural, lista-2 pode ser vista como uma sequência de elementos em que o primeiro é 2 e o segundo e último é 3.

Ao juntar mais um elemento a lista-2 vai criar lista-3.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



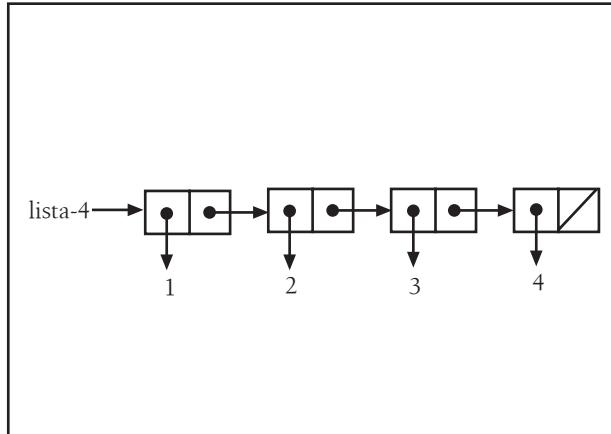
Experimente a criação e visualização de listas.

Crie a lista `lista-3` e visualize-a.

Visualize também `lista-1` e `lista-2`. Que conclusão retira do resultado visualizado?

O procedimento primitivo `list` é uma forma simples de criar listas a partir dos seus elementos. A lista é criada de uma só vez, o que não acontece com `cons`. No caso de `cons`, acrescenta-se apenas um elemento de cada vez e a nova lista vai sendo progressivamente constituída, como teve oportunidade de verificar.

```
> (define lista-4 (list 1 2 3 4))  
> lista-4  
(1 2 3 4)
```



Verifique que `(list 1 2 3 4)` é equivalente a `(cons 1 (cons 2 (cons 3 (cons 4 ()))))`



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Junte a uma lista novos elementos que são listas

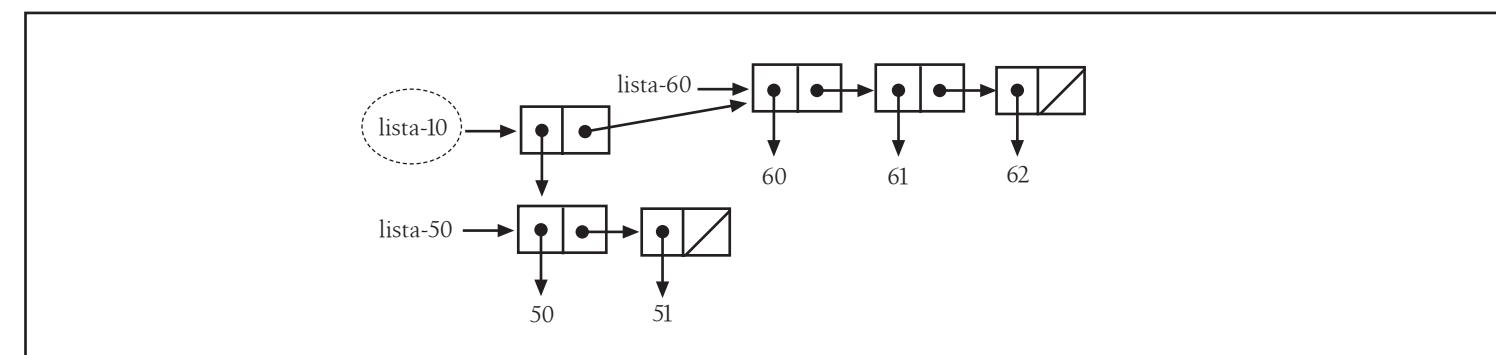
Qualquer entidade Scheme pode ser elemento de uma lista, o que significa que as próprias listas podem ser elementos de outras listas.

```
> (define lista-50 (list 50 51))  
> (define lista-60 (list 60 61 62))
```



Analise o que resulta da agregação destas duas listas com o construtor `cons`.

```
> (define lista-10 (cons lista-50 lista-60))
```



Com o construtor `cons` é criada lista-10.

Quantos são os elementos que compõem esta lista? Indique quais são esses elementos.



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
> lista-10  
((50 51) 60 61 62)
```

A lista lista-10, criada com o construtor cons, é composta por 4 elementos, uma vez que se junta um elemento, lista-50, a uma lista de 3 elementos, lista-60. Também se verifica pela representação caixa-apontador que lista-10 é uma sequência de 4 elementos, respectivamente lista-50, 60, 61 e 62.



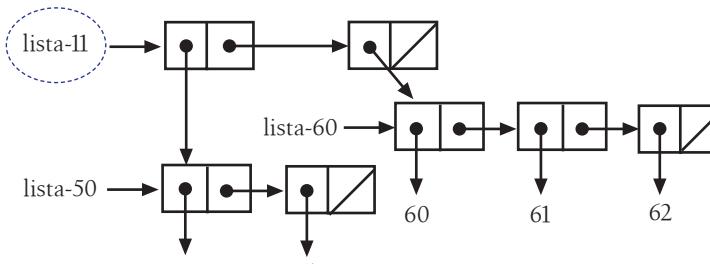
Indique quantos e quais os elementos que teria lista-10 se a definisse de outra maneira

```
(define lista-10 (cons lista-60 lista-50))
```

ou seja, se tivesse trocado entre si a posição dos argumentos de cons.

Veja agora o que resulta quando agrupa lista-50 e lista-60 com o construtor list.

```
> (define lista-11 (list lista-50 lista-60))
```



Com o construtor list é criada lista-11.

Quantos são os elementos que compõem esta lista?

Indique quais são esses elementos.



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

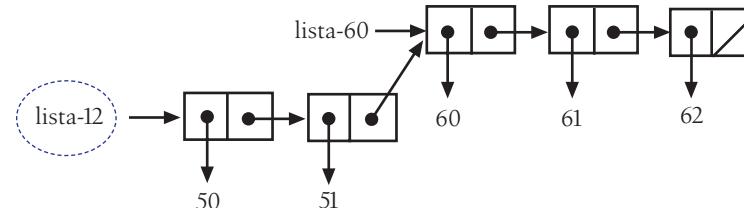
ANEXOS

```
> lista-11  
((50 51) (60 61 62))
```

Com `list`, foi criada uma nova lista, `lista-11`, composta por 2 elementos. Trata-se da sequência composta por `lista-50` e `lista-60`.

Para além dos procedimentos primitivos `cons` e `list`, utilizados na criação de listas, o Scheme disponibiliza também o procedimento `append`. Este procedimento aceita listas como argumentos e devolve uma nova lista composta por todos os elementos das listas fornecidas como argumentos.

```
> (define lista-12 (append lista-50 lista-60))
```



Com o construtor `append` é criada `lista-12`.  
Quantos são os elementos que compõem esta lista?  
Indique quais são esses elementos.

```
> lista-12  
(50 51 60 61 62)
```

De facto, o que `append` faz é copiar os elementos das várias listas para uma nova lista, excepto no caso da última lista que é partilhada com a lista criada. A representação caixa-apontador de `list-12` esclarece o que se acaba de afirmar.



A lista `lista-12` é composta por cinco elementos, dois copiados de `lista-50` e os três finais partilhados com `lista-60`.

De forma resumida, apresentam-se as várias maneiras de criar listas com os procedimentos primitivos, que são, por este facto, construtores.

- (cons obj lista)

Sendo o parâmetro `lista` uma lista de  $m$  elementos ( $m$  pode ser zero), a lista resultante é de  $m+1$  elementos, constituída por `obj` e pelos próprios elementos de `lista`. A criação de uma nova lista com `cons` é feita elemento a elemento, pois o procedimento `cons` apenas acrescenta um elemento a uma lista existente.

- (list obj-1 obj-2 ... obj-m)

Sendo  $m$  parâmetros, a lista resultante é de  $m$  elementos, constituída por  $\text{obj-1}$ ,  $\text{obj-2}$ , ...,  $\text{obj-m}$ . Com `list` cria-se uma lista de uma só vez, a partir de todos os seus elementos.

- (append lista-1 lista-2 ... lista-m)

Vamos utilizar o `append` apenas com parâmetros que sejam listas, apesar do Scheme admitir que o último seja outro tipo de objecto diferente de lista.

A lista resultante tem um número de elementos que é igual à soma do número de elementos das listas que são os parâmetros e é constituída por uma cópia dos elementos de `lista-1`, seguida por uma cópia dos elementos de `lista-2`... e partilhando os elementos de `lista-m`. Com `append` cria-se uma lista de uma só vez a partir dos elementos de várias listas.



Experimente os construtores de listas `cons`, `list` e `append`.

## Exercício 1

Sendo lista-2 equivalente a (list 7 (list 8 9)) e lista-3 equivalente a (list 1 2 3), apresente as representações textual e caixa-apontador dos objectos a seguir definidos, bem como o número de elementos de cada um deles.

```
(define objeto-1 (cons lista-2 lista-3))  
(define objeto-2 (list lista-2 lista-3))  
(define objeto-3 (append lista-2 lista-3))
```

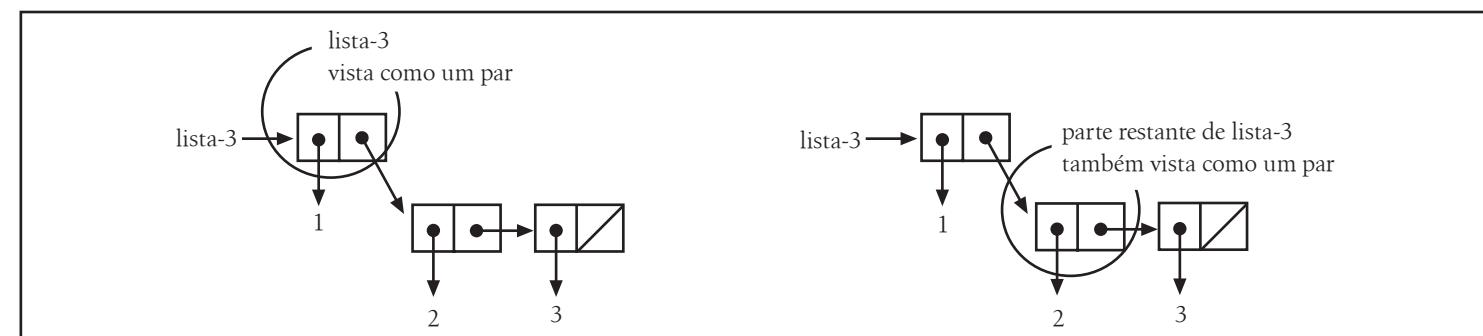


Verifique a correcção das respostas que vier a apresentar. Para isso, comece por criar lista-2 e lista-3 e depois objecto-1, objecto2, objecto-3.

**Não é só criar listas, é fundamental poder aceder aos seus elementos**

Para aceder aos elementos de uma lista, utilizam-se os selectores usados com os pares, `car` e `cdr`, o que não é para admirar, pois uma lista pode ser vista também como um par cujo elemento da esquerda é o primeiro elemento da lista e o elemento da direita é toda a lista, excluindo o seu primeiro elemento.

Tomando como exemplo a lista `lista-3`, pode vê-la como se fosse um par, em que a parte da esquerda desse par é o número 1 e a parte da direita é a lista `(2 3)`. Assim, `car` aplicado a `lista-3` devolverá 1 e `cdr` aplicado à mesma lista devolverá `(2 3)`.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



## Exercício 2

Sendo lista-3 equivalente a (list 1 2 3), complete o diálogo que se segue.

```
> (car (cdr lista-3))  
??  
> (cdr (cdr lista-3))  
??  
> (car (cdr (cdr lista-3)))  
??
```



Verifique a correcção das respostas que vier a apresentar.

## Exercício 3

Analise a definição que se segue das listas lista-1 e lista-2 e represente-as sob a forma de caixa-apontador.  
Seguidamente complete o diálogo.



O Anexo A, no que se refere a processamento de listas, indica 28 formas possíveis de combinar car e cdr.  
Por exemplo, (car (cdr lista)) é equivalente a (cadr lista).

```
> (define lista-1 (list 13 (list 45 32) (cons 1 2) 19)
> lista-1
??
> (cadr lista-1)
??
> (caaddr lista-1)
??
> (caddr lista-1)
??
> (caddr lista-1)
??
> (define lista-2 (cons (list 45 32) (cons 1 2)))
> lista-2
??
> (caar lista-2)
??
> (cdr lista-2)
??
> (cadr lista-2)
??
> (cadr lista-2)
```



Verifique a correcção das respostas que vier a apresentar.

**Vai agora conhecer os procedimentos que o Scheme oferece para processar as listas**

O Scheme disponibiliza um conjunto muito útil de procedimentos para processamento de listas, ou não fossem as listas a estrutura de dados privilegiada desta linguagem de programação. No **Anexo A**, em Processamento de Pares e Listas, encontra um resumo com estes procedimentos. Através de uma sequência de exemplos e exercícios, para além do treino de programação com listas que se proporciona, pretende-se ainda fazer uma introdução a grande parte desses procedimentos.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Em primeiro lugar, são definidas três listas, uma delas vazia.

```
> (define lista-1 (list 1 2 3 4 5 6))
> (define lista-2 (list 10 11 12))
> (define lista-vazia (list))
> lista-vazia
()
```

#### Exemplo 1 - null? - lista vazia?

Escreva em Scheme o procedimento soma-elementos-da-lista que recebe uma lista de números como argumento e devolve a soma de todos os seus elementos.

Se a lista for vazia, o resultado é conhecido e igual a zero. Trata-se do caso base. Os outros casos correspondem à lista não vazia e podem ser tratados recursivamente somando o primeiro elemento da lista, (`(car lista)`), à soma dos restantes, sabendo que os restantes constituem a lista que se obtém a partir da original retirando-lhe o primeiro elemento, (`(cdr lista)`). O procedimento primitivo `cdr` é, neste caso, a operação de redução, pois vai na direcção do caso base, ou seja, da lista vazia.

E assim, tem tudo o que é necessário para escrever o procedimento pedido, uma vez que o Scheme disponibiliza o predicado `null?` que aceita uma lista como argumento e devolve `#t` se a lista for vazia.

```
> (null? lista-1)
#f
> (null? lista-vazia)
#t

(define soma-elementos-da-lista
  (lambda (lis)
    (if (null? lis)
        0
        (+ (car lis)
            (soma-elementos-da-lista (cdr lis))))))

> (soma-elementos-da-lista lista-2)
33
> (soma-elementos-da-lista lista-vazia)
0
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Teste o procedimento soma-elementos-da-lista.

### Exercício 4

O procedimento soma-elementos-da-lista gera um processo recursivo, com  $O(n)$  em tempo e espaço.

Em alternativa, escreva uma versão que crie um processo iterativo e identifique a respectiva ordem de crescimento em relação ao tempo e espaço.



Desenvolva e teste o procedimento soma-elementos-da-lista que gera processos iterativos.

### Exercício 5 - length - qual o comprimento da lista?

O procedimento comprimento-da-lista tem como parâmetro uma lista e devolve o seu comprimento, ou seja, o número dos seus elementos.

```
> (comprimento-da-lista lista-1)  
6
```



Em relação ao procedimento comprimento-da-lista, pense um pouco e identifique o caso base, o caso geral e a operação de redução, antes de analisar a solução que se apresenta de seguida.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
(define comprimento-da-lista
  (lambda (lis)
    (if (null? lis)
        0
        (add1 (comprimento-da-lista (cdr lis))))))
```

O Scheme disponibiliza o procedimento `length` que faz o mesmo que `comprimento-da-lista`.

```
> (length lista-2)
3
> (length lista-vazia)
0
```



Teste os procedimentos `comprimento-da-lista` e `length`.

**Exemplo 2** - `list-ref` - o elemento da lista que se encontra na posição  $n$

A questão que agora se coloca relaciona-se com a selecção do elemento  $n$  de uma lista. Para este efeito, considera-se que o primeiro elemento é o elemento na posição 0, o segundo é o elemento na posição 1, e assim sucessivamente. O procedimento que se pretende é designado por `o-elemento-n-da-lista`, aceita uma lista e um inteiro positivo como argumentos e responde da seguinte maneira:

```
> (o-elemento-n-da-lista lista-1 2)
3
> (o-elemento-n-da-lista lista-1 0)
1
```



**O caso base corresponde ao argumento inteiro com um certo valor.**

**Qual é o valor desse argumento?**

**Qual a expressão a usar no caso base?**

**O caso geral resume-se a uma chamada recursiva do procedimento com redução dos dois argumentos.**

**Quais são as duas operações de redução?**



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
(define o-elemento-n-da-lista
  (lambda (lis n)
    (if (zero? n)
        (car lis)
        (o-elemento-n-da-lista (cdr lis) (sub1 n)))))
```



O que acontece se n for igual ao comprimento de lis?

O que acontece se n for maior que o comprimento de lis?

O Scheme disponibiliza o procedimento `list-ref`, que faz o mesmo que `o-elemento-n-da-lista`.

```
> (list-ref lista-1 2)  
3
```



Teste os procedimentos `o-elemento-n-da-lista` e `list-ref`.

Relativamente ao procedimento `o-elemento-n-da-lista`, indique a ordem de crescimento dos processos que gera, em relação ao tempo e espaço.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

### Exemplo 3

Temos falado essencialmente no acesso ao primeiro elemento das listas e também ao elemento da posição n. E se pretender aceder ao último elemento de uma lista?

A questão não faz sentido se a lista for vazia, pois, neste caso, não existe o último nem qualquer outro elemento. Se a lista tiver um único elemento, o primeiro e o último são o mesmo elemento e a resposta é conhecida: trata-se do caso base, que se identifica fazendo (`null? (cdr lista)`).



Indique uma alternativa equivalente a (`null? (cdr lista)`) que passe pela utilização de `length`.



Analise a solução que se apresenta e justifique a necessidade do procedimento recursivo local `aux`.

```
(define o-ultimo-elemento-da-lista
  (lambda (lis)
    (letrec ((aux
              (lambda(lis-aux)
                (if (null? (cdr lis-aux))
                    (car lis-aux)
                    (aux (cdr lis-aux)))))))
      ;
      (if (null? lis)
          (display "lista vazia")
          (aux lis)))))

> (o-ultimo-elemento-da-lista '())
lista vazia
> (o-ultimo-elemento-da-lista (list 3 56 3 78))
78
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Pretende-se agora encontrar uma solução baseada num procedimento não recursivo para aceder ao último elemento de uma lista que faça uso dos procedimentos primitivos `length` e `list-ref`.

```
(define o-ultimo-elemento-da-lista-v1
  (lambda(lis)
    (let ((comprimento (length lis)))
      (if (zero? comprimento)
          (display "lista vazia")
          (list-ref lis (sub1 comprimento))))))
```



O que aconteceria se em vez de `(list-ref lis (sub1 comprimento))` escrevesse `(list-ref lis comprimento)`?



Não sabe como `list-ref` foi implementado mas sabe como uma lista é constituída. Assim, tente imaginar a ordem de crescimento dos processos gerados por `o-ultimo-elemento-da-lista-v1`, em relação ao tempo e espaço. Justifique.

**Exemplo 4** - `reverse` - uma lista gerada a partir de outra lista mas com os elementos na ordem inversa.

Apresenta-se mais um exemplo de um procedimento não recursivo para aceder ao último elemento de uma lista, que utiliza o procedimento primitivo `reverse`. Este procedimento primitivo toma uma lista como argumento e devolve outra lista constituída por elementos iguais aos da lista dada, mas na ordem inversa.

```
> lista-2
(10 11 12)
> (reverse lista-2)
(12 11 10)
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
(define o-ultimo-elemento-da-lista-v2
  (lambda(lis)
    (if (zero? (length lis))
        (display "lista vazia")
        (car (reverse lis)))))

> (o-ultimo-elemento-da-lista-v2 lista-2)
12
```



Teste os procedimentos o-ultimo-elemento-da-lista nas 3 versões indicadas.



**Não sabe como `reverse` foi implementado mas sabe como uma lista é constituída.**

**Assim, tente imaginar a ordem de crescimento dos processos gerados por `o-ultimo-elemento-da-lista-v2`, em relação ao tempo e ao espaço. Justifique.**

Nos exemplos que se seguem, os procedimentos actuam sobre listas e devolvem listas, o que não acontecia com os procedimentos anteriores.

**Exemplo 5** - `list-tail` - uma lista que é a cauda de outra lista

Escreva em Scheme o procedimento `remove-primeiros-n` que tem como parâmetros uma lista e um inteiro positivo,  $n$ , e que devolve uma sublist da lista dada que não considera os primeiros  $n$  elementos.

# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

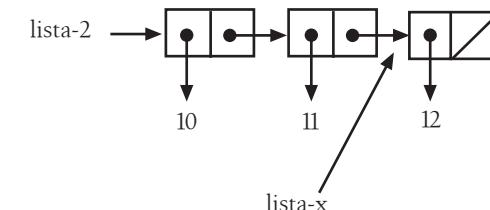
5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
> lista-2
(10 11 12)
> (define lista-x (remove-primeiros-n lista-2 2))
> lista-x
(12)
> lista-2
(10 11 12)
```



O caso base corresponde ao argumento inteiro com um certo valor.  
Qual é o valor desse argumento? Qual a expressão a usar no caso base?  
O caso geral resume-se a uma chamada recursiva do procedimento com redução dos dois argumentos.  
Quais são as duas operações de redução?

```
(define remove-primeiros-n
  (lambda(lis n)
    (if (zero? n)
        lis
        (remove-primeiros-n (cdr lis) (sub1 n)))))
```

O Scheme disponibiliza o procedimento `list-tail`, com uma funcionalidade semelhante a `remove-primeiros-n`.

```
> (list-tail lista-2 2)
(12)
```



Teste os procedimentos `remove-primeiros-n` e `list-tail`.



# S C H E M E

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

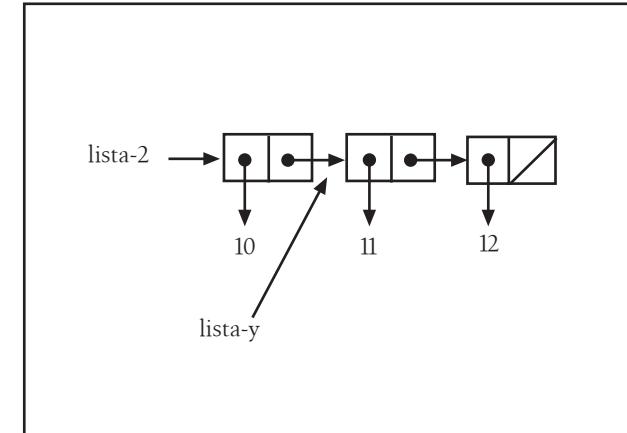
ANEXOS



**Exemplo 6** - `member` - outra forma de exprimir uma lista que é a cauda de outra lista

Escreva em Scheme o procedimento `remove-ate-elem` que espera dois argumentos: um eventual elemento de uma lista e essa lista. Este procedimento devolve uma sublist da lista dada, esquecendo os primeiros elementos, até encontrar um elemento que seja igual ao elemento fornecido, que já não será esquecido.

```
> lista-2
(10 11 12)
> (define lista-y (remove-ate-elem 11 lista-2))
> lista-y
(11 12)
> (remove-ate-elem 30 lista-2)
#f
> lista-2
(10 11 12)
```



Como responde o procedimento quando o primeiro argumento não faz parte da lista dada?

Parece-lhe bem? Justifique.

Quando encontrar o procedimento primitivo `member` do Scheme, vai entender esta resposta de `remove-ate-elem`...

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Neste procedimento há 2 casos base.

Quais serão esses casos base?

Quais as expressões respectivas?

O caso geral resume-se a uma chamada recursiva do procedimento com redução de um dos argumentos.

Qual é a operação de redução utilizada?

```
(define remove-ate-elem
  (lambda (elem lis)
    (cond ((null? lis) #f)
          ((equal? (car lis) elem)           ; equal?... !!!
           lis)
          (else
            (remove-ate-elem elem (cdr lis)))))))
```

A novidade que se verifica neste procedimento é o predicado `equal?`. De facto, poderia ser o operador `=`, num contexto limitado a valores numéricos. Mas `equal?` é muito mais geral e o espectro dos seus operandos é muito mais largo, pois compara qualquer tipo de objectos.

```
> (equal? (list 1 2) (list 1 2))
#t
> (= (list 1 2) (list 1 3))
=: expects type <number> as 2nd argument, given: (1 3); other arguments were: (1 2)
```



Entre `equal?` (o menos restritivo, aplicável a tudo) e `=` (o mais restritivo, só aplicável a números), o Scheme ainda disponibiliza mais dois comparadores, `eq?` e `eqv?`, cujos campos de aplicação não são fáceis de distinguir e que têm uma especificação dependente, algumas vezes, da implementação do próprio Scheme utilizado. Por este motivo, não se lhes dedica uma grande atenção e apenas se utiliza `=` em contextos numéricos simples (não compostos) e `equal?` nas restantes situações.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

O Scheme disponibiliza o procedimento `member` com uma funcionalidade semelhante à de `remove-at-elem`.

```
> (member 4 lista-1)
(4 5 6)
```



Teste os procedimentos `remove-at-elem` e `member`.



Para além de `member`, que utiliza o `equal?`, o Scheme ainda disponibiliza `memv` e `memq` que utilizam, respetivamente, `eqv?` e `eq?`, e que têm por este motivo, campos de aplicação igualmente não muito fáceis de distinguir, como se refere em nota anterior.

**Exemplo 7** - `append` - uma lista criada pelos elementos das listas dadas como argumento

Escreva em Scheme o procedimento `junta-duas-listas` que aceita duas listas como argumentos e devolve uma lista com uma cópia de todos os elementos daquelas listas.

```
> lista-2
(10 11 12)
> lista-1
(1 2 3 4 5 6)
> (junta-duas-listas lista-1 lista-2)
(1 2 3 4 5 6 10 11 12)
```

Trata-se de um caso particular de `append`, pois este procedimento primitivo do Scheme aceita um número não fixo de listas.

```
> (append lista-1 lista-2)
(1 2 3 4 5 6 10 11 12)
> (append lista-1 lista-2 lista-1)
(1 2 3 4 5 6 10 11 12 1 2 3 4 5 6)
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (junta-duas-listas lista-1 (junta-duas-listas lista-2 lista-1))
(1 2 3 4 5 6 10 11 12 1 2 3 4 5 6)
> lista-2
(10 11 12)
> lista-1
(1 2 3 4 5 6)

(define junta-duas-listas
  (lambda (lis-1 lis-2)
    (if (null? lis-1) ; caso base, se lis-1 é lista vazia,
        lis-2           ; então é devolvida lis-2
        (cons (car lis-1) ; caso geral: juntar o primeiro elemento de
              (junta-duas-listas (cdr lis-1) ; lis-1 ao resultado de juntar
                                 lis-2)))) ; (cdr lis-1) a lis-2
```



Teste os procedimentos `junta-duas-listas` e `append`.



Explique a ideia em que se baseia o procedimento `junta-duas-listas`.

### Exercício 6

Tendo em consideração o procedimento `junta-duas-listas` do exemplo anterior, se `lis-1` for a lista `(1 2 3)` e `lis-2` a lista `(10 11 12 13)`, complete a sequência de operações correspondentes à chamada `(junta-duas-listas lis-1 lis-2)`.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(junta-duas-listas lis-1 lis-2) =  
= (cons (car lis-1)  
        (junta-duas-listas (cdr lis-1) lis-2)) =  
= (cons 1  
        (junta-duas-listas (2 3) lis-2)) =  
= ...
```

Indique a ordem de crescimento, em relação tempo e ao espaço, desta solução.

### Nem sempre números, booleanos, pares, listas... agora, símbolos

Até aqui utilizámos essencialmente números para representar as entidades a processar, mas, em certas situações, torna-se muito mais cómodo utilizar símbolos em vez de números. Quando se trabalha com os meses do ano, não será muito mais intuitivo usar `janeiro` ou `fevereiro` em vez de uma codificação numérica qualquer? Em vez de representar uma cor através de um código numérico, não será muito mais significativo dizer que se trata, por exemplo, do azul?

Considera a seguinte situação, em que o António se dirige ao José em voz alta:

Hipótese 1- Diz 'o teu nome!'

Hipótese 2- Diz o teu nome.

A resposta provável será ouvirmos a palavra José, ou seja, o nome da pessoa a quem António se dirigiu. Todavia, se António em vez de comunicar oralmente o tivesse feito por escrito, as respostas às duas hipóteses teriam sido diferentes, provavelmente.

Hipótese 1- o teu nome.

Hipótese 2- José.

De onde vem a diferença? Na hipótese 1, uma parte da frase vem delimitada por plicas, significando que essa parte não deverá ser processada, mas mantida integralmente. Na hipótese 2, a frase deverá ser completamente processada.

Em Scheme, uma plica tem um significado parecido com o que acabámos de apresentar, possibilitando delimitação expressões ou parte de expressões que não deverão ser processadas, mas sim mantidas integralmente.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Veja alguns exemplos:

```
> (define lista-vazia '())
> lista-vazia
()
> (define lista-123 '(1 2 3))
> lista-123
(1 2 3)
> (define nome 'ana)
> nome
ana
> (define lista-nomes '(ana jose manuel))
> lista-nomes
(ana jose manuel)
```

Nestes exemplos, ao identificador `lista-vazia` fica associado um par de parêntesis, ou seja, uma lista vazia, a `lista-123` fica associada a lista `(1 2 3)`, a `nome` o símbolo `ana` e a `lista-nomes` uma lista de símbolos, `(ana jose manuel)`.



**Se, como alternativa ao último exemplo, tentasse:**

```
> (define lista-nomes (list ana jose manuel))
reference to undefined identifier: ana
```

**Justifique a resposta do Scheme.**

A expressão `(list ana jose manuel)` seria processada para determinação do valor a associar ao identificador `lista-nomes`, e então os identificadores `ana`, `jose`, e `manuel` seriam entendidos como nomes de variáveis, originando a mensagem de erro respectiva pelo facto de não terem sido previamente definidas. Uma alternativa ao último exemplo, mas que funciona, seria:

```
> (define lista-nomes (list 'ana 'jose 'manuel))
> lista-nomes
(ana jose manuel)
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Entende-se que, neste caso, cada um dos três argumentos de `list` não seria calculado, mas tomado tal e qual é, não havendo lugar a uma mensagem de erro.

Em vez de plica, o Scheme disponibiliza o procedimento `quote`, com uma funcionalidade semelhante.

```
> (quote (1 d 2 f))  
(1 d 2 f)
```

### Exercício 7

Analise as várias situações que se apresentam e indique os respectivos resultados.

```
> (define a 10)  
> (car '(a b c d))  
???  
> (car (list a))  
???  
> (car (list 'a))  
???  
> (car '(list a))  
???  
> (list (a 'a))  
???
```



Verifique as respostas às situações apresentadas.

### Exemplo 8

Numa certa aplicação são utilizadas cores, que são referidas directamente pelos seus nomes e não por códigos numéricos. Os nomes das cores em jogo e os respectivos códigos são: vermelho - código 1, verde - código 2, azul - código 3 e amarelo - código 4.

O procedimento `num->cor` recebe como argumento um número, possivelmente um código de uma cor, e devolve um símbolo correspondente ao nome dessa cor.

# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
> (num->cor 1)
vermelho
> (num->cor 6)
codigo nao previsto

(define num->cor
  (lambda (num)
    (cond ((= num 1) 'vermelho)
          ((= num 2) 'verde)
          ((= num 3) 'azul)
          ((= num 4) 'amarelo)
          (else (display "codigo nao previsto")))))
```

Uma alternativa a esta solução.

```
(define num->cor
  (lambda (num)
    (let ((cores '(vermelho verde azul amarelo)))
      (if (> num 4)
          (display "codigo nao previsto")
          (list-ref cores (sub1 num))))))
```



Das 2 versões do procedimento num->cor, qual prefere? Apresente algumas razões.

A segunda solução passa pela definição local de uma lista com os símbolos referentes às designações das cores, o que resulta num procedimento cujo tamanho é independente do número de cores em jogo.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 8

Escreva em Scheme o procedimento `cor->num` que responde como se indica:

```
> (cor->num 'vermelho)  
1  
> (cor->num 'amarelo)  
4  
> (cor->num 'rosa)  
cor nao prevista
```



Teste o procedimento `num->cor` e desenvolva e teste o procedimento `cor->num`.

### Exercício 9

Escreva em Scheme o procedimento `da-carta` que simula a escolha aleatória de uma carta de um baralho de 52 cartas de jogar e a respectiva visualização.

```
> (da-carta)  
dois de copas  
> (da-carta)  
dama de ouros
```



Desenvolva e teste o procedimento `da-carta`.

Pista: Sugere-se a constituição local de duas listas, uma com as designações das 13 faces de um baralho (as, duque, tres, quatro, cinco, seis, sete, oito, nove, dez, dama, valete e rei) e outra com as designações dos 4 naipes (copas, ouros, paus e espadas). Bastará gerar dois números aleatórios: um entre 0 e 12, para escolher uma face na lista das faces, e outro entre 0 e 3, para escolher um naipe na lista dos naipes.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

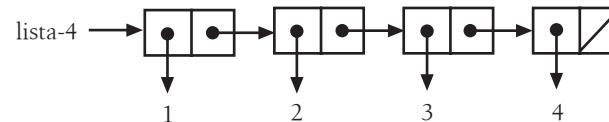
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

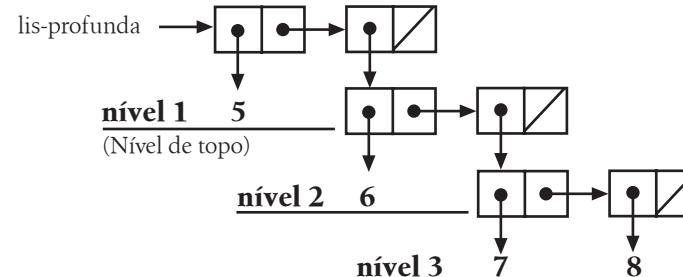
## Processamento das listas em profundidade

A lista designada por `lista-4` é composta por 4 elementos, todos eles números, logo, todos eles elementos simples.

Esta lista tem apenas um nível, já que não é possível visitar os seus elementos em profundidade, por serem elementos simples.



Quando na composição de uma lista entram elementos compostos, por exemplo elementos que também são listas, já será possível visitá-los ou processá-los em profundidade. É o que acontece com a lista `lis-profunda`, (`(5 (6 (7 8)))`), que apresenta 3 níveis de profundidade.



No nível 1, também designado por nível de topo, a lista tem 2 elementos: 5 e `(6 (7 8))`. No nível 2, continua a ter 2 elementos: 6 e `(7 8)`. Finalmente, no nível 3, os elementos da lista são 7 e 8.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Analise agora outros exemplos:

- 50
- (50 51 52)
- '((a b) c () ((d (e))))'

está no nível zero, por ser tratar de um elemento simples  
qualquer um dos elementos da lista está no nível 1  
no nível 1 a lista tem 4 elementos.

Represente graficamente a última lista.

Quais são os elementos desta lista:

- No nível 1? No nível 2? No nível 3? E no nível 4?

A lista lis-profunda tem, no nível 1, apenas um elemento simples, ou seja, um elemento que não é um par, o número 5. Todavia, se considerar a lista em todos os seus níveis, ou seja, se considerar a lista em profundidade, encontra 4 elementos que não são pares: 5, 6, 7 e 8.



Neste contexto, "par" não tem nada a ver com a paridade de um número. Aqui, "não é um par" apenas significa que se trata de um elemento simples.

Uma lista vazia também não é um par.

O Scheme disponibiliza o predicado `pair?` que indica se o respectivo argumento é ou não um par ([Anexo A](#)).

Processar uma lista em profundidade significa que, sempre que se encontre um elemento que seja par, este poderá ser tratado em profundidade. Por exemplo, a lista `((a b) c () ((d (e))))` tem 2 elementos não pares no nível de topo (que são `c` e `()`), mas quando a consideramos em profundidade encontramos 6 elementos não pares: `a`, `b`, `c`, `()`, `d`, `e`.



# S C H E M E

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

### Exemplo 9

Escreva em Scheme o procedimento `conta-todos` que tem uma lista como parâmetro e devolve o número de elementos que não são pares, contados em todos os níveis da lista.

```
> (conta-todos '((a b) c () ((d (e)))))  
6  
> (conta-todos '(() () ()))  
3  
> (conta-todos '())  
0  
  
(define conta-todos  
  (lambda (lis)  
    (cond ((null? lis) 0) ; caso base  
          ;  
          ; caso geral em que o primeiro elemento é um par...  
          ;  
          ((pair? (car lis))  
           (+ (conta-todos (cdr lis)) ; processamento ao nível de topo  
               (conta-todos (car lis)))) ; processamento em profundidade  
          ;  
          ; caso geral em que o primeiro elemento não é um par...  
          ; conta mais um...  
          (else  
            (add1 (conta-todos (cdr lis)))))))
```



Teste o procedimento `conta-todos`.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



No processamento de uma lista, a respectiva redução no passo recursivo toma a forma conhecida

(procedimento-recursivo (cdr lista) ... )

que sugere uma redução ao nível de topo (nível 1).

No processamento em profundidade de uma lista, quando um elemento encontrado (por exemplo, o primeiro) é um par, o padrão de computação típico mostra a forma conhecida

(procedimento-recursivo (cdr lista) ... )

que sugere uma redução ao nível de topo, acompanhada pela forma

(procedimento-recursivo (car lista) ... )

que sugere uma visita em profundidade ao primeiro elemento da lista.

### Exercício 10

Escreva em Scheme o procedimento soma-todos que tem uma lista como parâmetro único e devolve a soma de todos os seus elementos, procurando-os em todos os níveis.

```
> (soma-todos '((8 5) 12 () ((-8 (-10)))))  
7  
> (soma-todos '(() (71) ()))  
71  
> (soma-todos '())  
0
```



Desenvolva e teste o procedimento soma-todos .

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

## Módulo 3.3 - Um jogo para testar a memória

O desenvolvimento de um programa que funciona como teste à memória do utilizador surge como um exemplo de aplicação da abstracção conjunto. Já depois deste programa desenvolvido, introduziram-se alterações à abstracção conjunto, tendo em vista a obtenção de uma versão com melhor desempenho. A aplicação desta última versão no programa já desenvolvido, sem exigir qualquer alteração neste, demonstra bem que a abstracção funciona como uma barreira que esconde perfeitamente as suas características internas. Isto significa que mantendo a interface de uma abstracção, ou seja, a sua funcionalidade, é possível submetê-la a melhoramentos sem inviabilizar as aplicações que entretanto tenham sido desenvolvidas sobre ela.

### Palavras-Chave

Conjunto, lista não ordenada e sem elementos repetidos, lista ordenada e sem elementos repetidos, lista não ordenada e com elementos repetidos, reutilização de código.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Abstracção conjunto

Numa sequência de objectos, a ordem que eles ocupam na sequência é tida em consideração. No início da sequência encontrará o 1º objecto, depois o 2º, e assim sucessivamente até ao último objecto. Mas nem sempre interessa reter a ordem dos objectos, por vezes bastará saber se um objecto faz ou não parte de um certo conjunto de objectos. Por exemplo, se pretender saber quem são as pessoas que estão numa sala, bastará tomar nota dos seus nomes, não importando a ordem de entrada. Esta linha de raciocínio sugere algumas aplicações para a abstracção conjunto.

Um exemplo de aplicação será apresentado como um programa para testar a capacidade de memorização do utilizador. O programa gera números aleatoriamente e o utilizador terá que ir respondendo se o número já terá saído ou não. Os números saídos vão sendo guardados numa estrutura do tipo conjunto.



**Tente identificar uma outra situação em que a abstracção conjunto possa ser útil.**

Vamos começar por identificar a funcionalidade associada à abstracção conjunto e, em seguida, analisar algumas alternativas para a representar. Depois da respectiva implementação, a abstracção conjunto fica pronta para ser utilizada.

Podemos definir informalmente conjunto como sendo uma colecção de objectos distintos, cujas operações básicas são:

- (faz-conjunto)

Não espera qualquer argumento e devolve um conjunto vazio. Trata-se de um construtor;

- (junta-elemento-a-conjunto obj conj)

Toma um objecto e um conjunto como argumentos e devolve um conjunto formado pelos elementos do conjunto dado, aos quais se junta aquele objecto. Trata-se de um construtor;



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

- (*uniao-de-conjuntos conj1 conj2*)  
Toma dois conjuntos como argumentos e devolve um conjunto com os elementos que pertencem a pelo menos um dos conjuntos. Trata-se de um construtor;
- (*interseccao-de-conjuntos conj1 conj2*)  
Toma dois conjuntos como argumentos e devolve um conjunto com os elementos que pertencem simultaneamente aos dois conjuntos. Trata-se de um construtor;
- (*elemento-do-conjunto? obj conj*)  
Toma um objecto e um conjunto como argumentos e devolve #t ou #f, caso o objecto pertença ou não ao conjunto. Trata-se de um selector.

Repare que todas as operações básicas se apresentam como construtores de conjuntos, pois todas devolvem conjuntos, excepto a última, *elemento-do-conjunto?*, que se assume como um selector.

Falta implementar todas estas operações para que a abstracção conjunto passe a estar disponível para o desenvolvimento de aplicações que envolvam este tipo de dados. Para isso é necessário escolher uma estrutura de dados adequada para a representar, decisão importante, mas não dramática, se não se encontrar a melhor estrutura, logo à primeira.



**Imagine que não encontra à primeira uma boa estrutura de dados para representar os objectos da abstracção conjunto.**

**Imagine também que entretanto desenvolveu algumas aplicações com esta abstracção.**

**Se mais tarde resolver desenvolver uma nova versão da abstracção conjunto, optimizando-a com uma estrutura de dados mais apropriada, o que é que acontece às aplicações anteriormente desenvolvidas. Terão que ser alteradas?**





Depois de alguma reflexão, indique uma estrutura de dados para representar os objectos do tipo conjunto.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Tendo como principal preocupação chegar rapidamente a uma abstracção que funcione, mesmo que não seja a mais apropriada, poderia utilizar listas não ordenadas e sem elementos repetidos para representar os objectos do tipo conjunto. Isto significa, por exemplo, que o conjunto constituído por 1, 2, 3, e 4, poderá ser representado pelas listas (1 2 3 4) ou (3 4 1 2), ou uma outra lista qualquer que contenha apenas aqueles valores. O conjunto vazio será representado por uma lista vazia.

Antes de se passar à implementação das operações básicas da abstracção conjunto, veja como elas devem responder.

```
> (define c-1 (faz-conjunto))
> c-1
()
> (define c-2 (junta-elemento-a-conjunto 5 c-1))
> c-2
(5)
> (define c-3 (junta-elemento-a-conjunto 'abc c-2))
> c-3
(abc 5)
> (interseccao-de-conjuntos c-2 c-3)
(5)
> (define c-10 (faz-conjunto))
> (define c-11 (junta-elemento-a-conjunto 4 c-10))
> (define c-12 (junta-elemento-a-conjunto 5 c-11))
> c-12
(5 4)
> (uniao-de-conjuntos c-12 c-3)
(4 abc 5)
```

Segue-se agora a implementação das operações básicas definidas para esta abstracção.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
(define faz-conjunto
  (lambda ()
    (list)))
```



**Justifique a implementação do construtor `faz-conjunto`.**

Para determinar se um objecto pertence a um conjunto, compara-se esse objecto com todos os elementos do conjunto.

- Se o conjunto for vazio, a resposta é imediata e igual a `#f`, pois o objecto não pode estar num conjunto vazio. Trata-se do caso base.
- Não sendo vazio, compara-se o objecto com um elemento do conjunto. Seja, por exemplo, o primeiro elemento da lista que representa o conjunto.
  - Se forem iguais, então a resposta é `#t` e acaba a comparação. Eis mais uma situação do caso base.
  - Sendo diferentes, recorre-se a uma chamada recursiva, em que se reduz a lista que representa o conjunto, retirando-lhe o seu primeiro elemento.

```
(define elemento-do-conjunto?
  (lambda (obj conj)
    (cond ((null? conj) #f)
          ((equal? obj (car conj)) #t)
          (else (elemento-do-conjunto? obj (cdr conj))))))
```



Diga o que aconteceria em `elemento-do-conjunto?` se, por engano, não fosse incluído o caso base correspondente a `(null? conj)`.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



A implementação da operação junta-elemento-a-conjunto é agora apresentada.

```
(define junta-elemento-a-conjunto
  (lambda (obj conj)
    (if (elemento-do-conjunto? obj conj)
        conj
        (cons obj conj))))
```



Esta operação é um construtor.

Analisando a sua implementação, justifique esta afirmação.

Como é garantido que o conjunto criado pela junção de um novo elemento não tem elementos repetidos?



Teste a parte já desenvolvida da abstracção conjunto.

A ideia a explorar, para determinar o conjunto que resulta da união de dois conjuntos, pode resumir-se a percorrer um primeiro conjunto e identificar, um a um, todos os seus elementos que não pertençam ao segundo conjunto. O conjunto resultante da união será constituído pelos elementos assim identificados, juntando-lhes os elementos do segundo conjunto.



Seguindo a ideia indicada, como garante que a união dos dois conjuntos continua a ser um conjunto sem elementos repetidos?

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Numa óptica de solução recursiva, o caso geral resume-se à redução do primeiro conjunto, tomando, um a um, todos os seus elementos, até se atingir o conjunto vazio.

- O conjunto vazio correspondente ao caso base e, nesta situação, a união será o segundo conjunto.
- Mas enquanto não se atinge o conjunto vazio, verifica-se se cada um dos seus elementos já se encontra no segundo conjunto.
  - Se já se encontrar no segundo conjunto, não será considerado para fazer parte da união,
  - caso contrário, será imediatamente considerado.

```
(define uniao-de-conjuntos
  (lambda (conj1 conj2)
    (cond ((null? conj1) conj2)
          ((elemento-do-conjunto? (car conj1) conj2)
           (uniao-de-conjuntos (cdr conj1) conj2))
          (else
            (cons (car conj1)
                  (uniao-de-conjuntos (cdr conj1) conj2))))))
```



Sendo `conj2` a lista `(1 2 3 4)`, indique os vários passos em que se desenvolve a expressão  
`(uniao-de-conjuntos conj1 e conj2)`

hipótese 1- se `conj1` for a lista `()`  
hipótese 2- se `conj1` for a lista `(3)`  
hipótese 3- se `conj1` for a lista `(45)`

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



## Exercício 1

Explique o funcionamento do procedimento interseccao-de-conjuntos que se segue.

```
(define interseccao-de-conjuntos
  (lambda (conj1 conj2)
    (cond ((or
              (null? conj1)
              (null? conj2))
           (list))
          ((elemento-do-conjunto? (car conj1) conj2)
           (cons (car conj1)
                 (interseccao-de-conjuntos (cdr conj1) conj2)))
          (else
           (interseccao-de-conjuntos (cdr conj1) conj2))))
```



Teste a abstracção conjunto.

## Exercício 2

Após a implementação das operações básicas da abstracção conjunto, baseada em listas não ordenadas e sem elementos repetidos, apresenta-se a ordem de crescimento dos processos que cada um deles gera. Analise e comente a tabela que se segue.

	tempo	espaço
faz-conjunto	$O(1)$	$O(1)$
elemento-do-conjunto?	$O(n)$	$O(1)$
junta-elemento-a-conjunto	$O(n)$	$O(1)$
uniao-de-conjuntos	$O(n^2)$	$O(n)$
interseccao-de-conjuntos	$O(n^2)$	$O(n)$

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Projecto - Testa memória

Um programa para testar a capacidade de memorização de quem o utiliza, vai ser desenvolvido sobre a abstracção conjunto, fazendo uso das seguintes operações: faz-conjunto, elemento-do-conjunto? e junta-elemento-a-conjunto. O programa, designado por testa-memoria, visualiza números aleatoriamente, um a um, e pergunta, para cada um deles, se já terá saído. As respostas incorrectas vão sendo contadas, pois, no final, o programa visualiza essa informação. A abstracção conjunto será utilizada para memorizar os números que vão sendo visualizados aleatoriamente, permitindo assim verificar a correcção das respostas dadas.

O programa recebe três argumentos, os dois primeiros definem a gama em que os números aleatórios são gerados, e o terceiro define a quantidade de números que são visualizados em cada sessão. Segue-se um exemplo de uma dessas sessões.

```
> (testa-memoria 1 10 5)
```

Serao visualizados 5 numeros, um a um, situados entre 1 e 10 e tera' que responder, para cada um deles, se ja' saiu.

Mais um numero: 6

Ja' saiu? s/n: n

Mais um numero: 5

Ja' saiu? s/n: n

Mais um numero: 9

Ja' saiu? s/n: n

Mais um numero: 3

Ja' saiu? s/n: h

Ja' saiu? s/n: n

Mais um numero: 3

Ja' saiu? s/n: n

Dos 5 numeros, falhou 1

resposta diferente  
de s ou n faz repetir  
a pergunta

O programa testa-memoria, por apresentar alguma complexidade, justifica uma abordagem do tipo de-cima-para-baixo, que consiste em dividir as tarefas em tarefas cada vez mais pequenas e simples, de tal forma que os procedimentos que as implementam também sejam simples.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

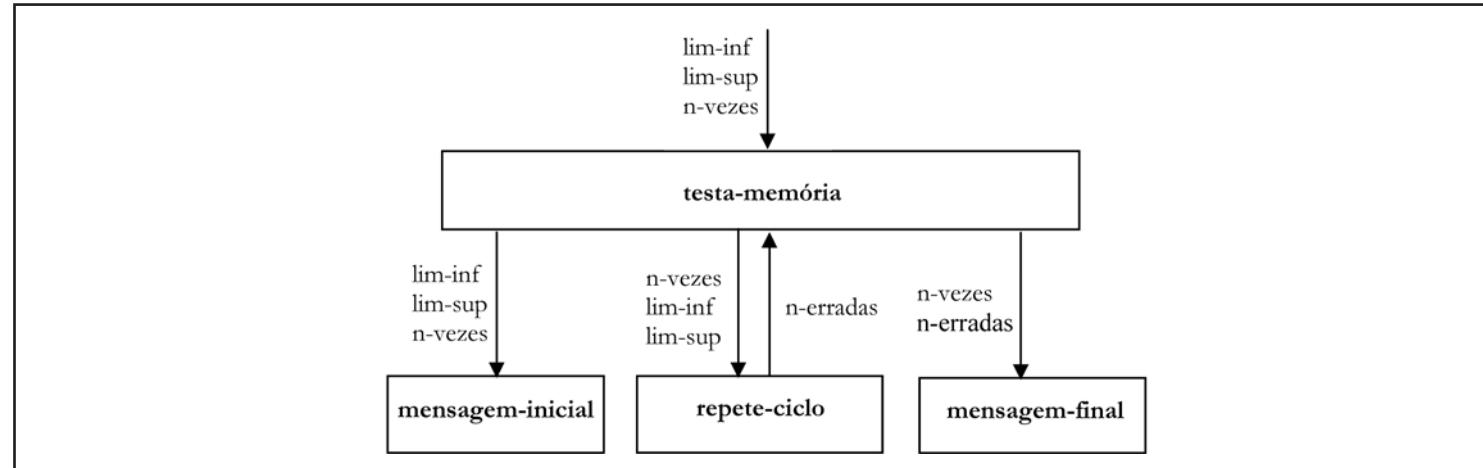
### ANEXOS

Começamos por identificar as grandes tarefas do programa testa-memória.

- Visualizar a mensagem inicial;
- Repetir ciclicamente as perguntas e contabilizar o número de respostas erradas;
- Visualizar a mensagem final.



Verifique a correcção do diagrama de fluxo de dados.



Podemos começar a escrever o programa testa-memória, não esquecendo de requerer a abstracção conjunto, suposta no ficheiro conjuntos.scm do directório PLT\collects\user-feup.

```
(require (lib "conjuntos.scm" "user-feup"))
```



Se tiver dúvidas sobre como reutilizar o código já desenvolvido, sugere-se a consulta do [Anexo C - Reutilização de código](#).



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
(define testa-memoria
  (lambda (lim-inf lim-sup n-vezes)
    (mensagem-inicial lim-inf lim-sup n-vezes)
    ;
    (let ((respostas-erradas (repete-ciclo n-vezes lim-inf lim-sup)))
      (mensagem-final n-vezes respostas-erradas))))
```

Procedimentos `mensagem-inicial` e `mensagem-final` (de `testa-memoria`)

O programa `testa-memoria` não pode ser testado de imediato, pois depende de procedimentos ainda não implementados. Por outro lado, verifique que a primeira e terceira tarefas são relativamente simples, admitindo de imediato as respectivas implementações e testes.

```
(define mensagem-inicial
  (lambda (lim-i lim-s n-perguntas)
    (newline)
    (display "Serao visualizados ")
    (display n-perguntas)
    (display " numeros, um a um, situados entre ")
    (display lim-i)
    (display " e ")
    (display lim-s)
    (newline)
    (display "e tera' que responder, para cada um deles, se ja' saiu.")
    (newline)))
```

Segue-se um teste simples deste procedimento.

```
> (mensagem-inicial 1 10 3)
```

Serao visualizados 3 numeros, um a um, situados entre 1 e 10  
e tera' que responder, para cada um deles, se ja' saiu.

Por seu turno, o procedimento `mensagem-final`, com os parâmetros correspondentes ao número de perguntas e ao número de respostas erradas, deverá responder como agora se indica.

```
> (mensagem-final 3 1)
```

Dos 3 numeros, falhou 1





Desenvolva o procedimento mensagem-final e teste este e o procedimento mensagem-inicial.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Procedimento repete-ciclo (de testa-memoria)

A simplicidade de uma tarefa e do correspondente procedimento pode medir-se, muitas vezes, pelo número de linhas que a escrita deste requer. Por isso, recomenda-se que a escrita dos procedimentos não ultrapasse a meia página, recomendação que nem sempre se segue...

Foi com esta regra de bom senso que se procurou identificar as grandes tarefas da segunda tarefa de testa-memoria, aquela que repete ciclicamente as perguntas e contabiliza o número de respostas erradas, designada por repete-ciclo.

- Enquanto o número de perguntas for diferente de zero, executar as seguintes tarefas:
  - Gerar um novo número aleatório;
  - Visualizar o número gerado;
  - Perguntar se o número já saiu e determinar se a resposta dada é incorrecta (o que implica a memorização de todos os números já saídos, para comparação com o número acabado de sair). Devolve #t se a resposta surgir errada;
  - Preparar novo ciclo...
    - ♦ se ainda não tiver saído, juntá-lo ao conjunto dos números já saídos;
    - ♦ se a resposta for incorrecta, incrementar o número de respostas erradas;
    - ♦ descontar 1 ao número de perguntas que é necessário fazer;
- Depois de todas as perguntas feitas, devolver o número de respostas erradas.

O procedimento repete-ciclo pode conter, localmente, um procedimento auxiliar que executa em ciclo as várias perguntas e também as tarefas anteriormente identificadas.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Apresente o diagrama de fluxo de dados do procedimento **repete-ciclo**.  
Depois deste exercício, verifique se esse diagrama é compatível com a solução que se segue e adapte-o se não for.

```
(define repete-ciclo
  (lambda (n-perguntas lim-i lim-s)
    ;----- definição de um procedimento auxiliar local
    ;
    (letrec ((aux
              (lambda (n-perg n-erros saidos)
                (if (zero? n-perg)
                    n-erros ; caso base: Devolve número de erros.
                    ; caso geral ...
                    ; 1- gera um número aleatório e visualiza-o
                    ;
                    (let ((novo-num (roleta-n-m lim-i lim-s)))
                      (visu-novo-numero novo-num)

                      ; 2- pergunta se o número já tinha saído
                      ; e devolve #t se resposta errada
                      ;
                      (let ((resposta-errada (pergunta-e-resposta-errada?
                                              novo-num saidos)))
                        ; 3- chamada recursiva ...
                        ;
                        (aux (sub1 n-perg) ; menos 1 pergunta
                            (if resposta-errada ; eventual
                                (add1 n-erros) ; actualização
                                n-erros) ; do número de erros
                            (junta-elemento-a-conjunto ; mais um número
                                novo-num saidos))))))) ; para o conjunto

; ----- corpo principal de repete-ciclo -----
; chamada de aux com n-perguntas, 0 erros, e conjunto vazio...
;
(aux n-perguntas 0 (faz-conjunto)))))
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Antes de poder passar ao teste do procedimento **repete-ciclo**, que procedimentos auxiliares deverá desenvolver e testar?

### Procedimentos roleta-n-m e visu-novo-numero (de repete-ciclo)

Os dois procedimentos que se seguem foram definidos fora de repete-ciclo para não complicar nem alongar mais este procedimento.

```
; tem dois parâmetro: inf - o limite inferior; sup - o limite superior
; devolve um número aleatório entre inf e sup
;
(define roleta-n-m
  (lambda (inf sup)
    (+ inf
       (random (add1 (- sup inf))))))

; tem num-aleat como parâmetro e visualiza-o
;
(define visu-novo-numero
  (lambda (num-aleat)
    (newline)
    (display "Mais um numero: ")
    (display num-aleat)
    (newline) ))
```



Teste os procedimentos roleta-n-m e visu-novo-numero.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Procedimento pergunta-e-resposta-errada? (de repete-ciclo)

Com o procedimento pergunta-e-resposta-errada? também se pretendeu não complicar nem alongar aux em demasia, mas neste caso, contrariamente ao que aconteceu com os dois procedimentos anteriores, achou-se conveniente decompor a respectiva tarefa em tarefas mais simples:

- verificar se o número em análise já saiu;
- interrogar o utilizador e determinar se a resposta é afirmativa;
- determinar se a resposta está errada e devolver o resultado.

```
; tem como parâmetros um número e o conjunto dos números já saídos
; devolve #t se resposta errada e #f se resposta correcta.
;
(define pergunta-e-resposta-errada?
  (lambda (num saídos)
    (let ((ja-saiu (elemento-do-conjunto? num saídos))
          (sim (resposta-sim?)))
      ;
      (or
        ; a resposta está errada se:
        (and ja-saiu
             (not sim))
        (and (not ja-saiu)
             ; ou já saiu e a resposta é não
             (not sim))
        (and (not ja-saiu)
             ; ou não saiu e a resposta é sim
             sim)))))
```



Antes de poder passar ao teste do procedimento **pergunta-e-resposta-errada?**, que procedimentos auxiliares deverá desenvolver e testar?

Desenhe o diagrama de fluxo de dados do procedimento **pergunta-e-resposta-errada?** com os respectivos procedimentos auxiliares.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



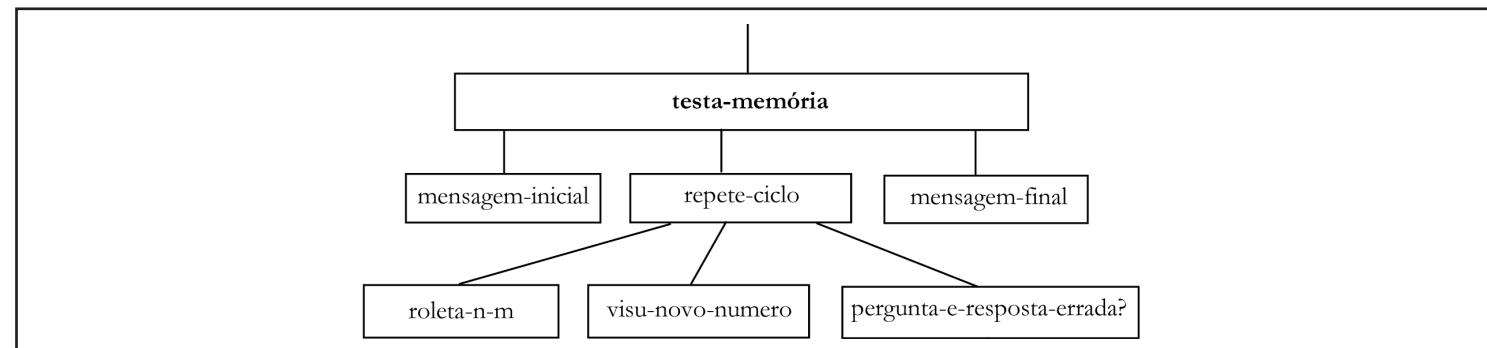
### Procedimento resposta-sim? (de pergunta-e-resposta-errada?)

```
; não tem parâmetros.  
; devolve #t se resposta s  
; devolve #f se resposta n  
; nos outros casos, repete a pergunta  
;  
(define resposta-sim?  
  (lambda ()  
    (display "Ja' saiu? s/n: ")  
    (let ((resposta (read)))  
      (cond ((equal? resposta 's)  
             #t)  
            ((equal? resposta 'n)  
             #f)  
            (else ; se resposta diferente de s e n...  
              (newline) ; repete a pergunta...  
              (resposta-sim?))))))
```



Teste o procedimento `resposta-sim?`, depois o procedimento `pergunta-e-resposta-errada?` e, finalmente, o procedimento `repete-ciclo`.

Está muito próximo do final do projecto, mas antes de passar ao teste do programa `testa-memória` inicialmente pedido, apresenta-se um esboço simplificado do diagrama de fluxo de dados com todos os procedimentos, a fim de poder verificar se falta desenvolver ou testar algum.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Confirme que está tudo desenvolvido e testado para passar ao programa **testa-memoria**.

E agora o teste do programa **testa-memoria**.

```
> (testa-memoria 5 10 3)
```

Serao visualizados 3 numeros, um a um, situados entre 5 e 10  
e tera' que responder, para cada um deles, se ja' saiu.

```
Mais um numero: 8  
Ja' saiu? s/n: n
```

```
Mais um numero: 6  
Ja' saiu? s/n: n
```

```
Mais um numero: 9  
Ja' saiu? s/n: n
```

```
Dos 3 numeros, falhou 0
```



Teste o programa **testa-memoria**.

Com este programa, teste a capacidade de memória dos seus colegas.

Será que algum apresenta características de memorização fora do normal?

## Outra estrutura de dados para os conjuntos

Vamos tentar uma outra representação dos conjuntos baseada em listas, em que os elementos passam a estar em ordem crescente e sem elementos repetidos. Para facilitar o problema, supõe-se que os conjuntos são constituídos apenas por elementos numéricos. O conjunto vazio continua a ser representado por uma lista vazia, pelo que se mantém faz-conjunto.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Depois de alguma reflexão, indique o tipo de impacto que terá esta nova representação dos conjuntos em cada uma das suas operações básicas: **elemento-do-conjunto?**, **junta-elemento-a-conjunto**, **interseccao-de-conjuntos** e **uniao-de-conjuntos**.

Quanto a **elemento-do-conjunto?**, há uma diferença a considerar, pois a pesquisa deve terminar logo que o objecto seja menor que o próximo elemento da lista que representa o conjunto. Não esquecer, a propósito, que o próximo elemento a comparar é o primeiro elemento da lista que vai sendo reduzida em cada passo recursivo. Além disso, como os elementos da lista estão ordenados do menor para o maior, se o objecto é menor que o primeiro da lista, então seria pura perda de tempo continuar a pesquisa.



Apresente para o procedimento **elemento-do-conjunto?**

- a expressão do caso base
- a expressão do passo recursivo

```
(define elemento-do-conjunto?
  (lambda (obj conj)
    (cond ((or
              (null? conj)          ; a pesquisa termina se o conjunto for vazio
              (< obj (car conj)))    ; ou se o objecto for menor
              #f)                   ; que o primeiro elemento da lista
              ((= obj (car conj)) #t)
              (else (elemento-do-conjunto? obj (cdr conj)))))))
```

Em termos de espaço, o processo gerado por **elemento-do-conjunto?** apresenta uma ordem de crescimento  $O(1)$ . O tempo gasto na pesquisa é, em média,  $n/2$ , em que  $n$  representa o número de elementos do conjunto.

Com  $\frac{n}{2} \leq K^*n$ , o processo em termos de tempo tem uma ordem de crescimento  $O(n)$ .



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Apresente para o procedimento **junta-elemento-a-conjunto**

- a expressão do caso base
- a expressão do passo recursivo

```
(define junta-elemento-a-conjunto
  (lambda (obj conj)
    (cond ((null? conj)
           (list obj))
          ((= obj (car conj))
           conj)
          ((> obj (car conj))
           (cons (car conj)
                 (junta-elemento-a-conjunto obj (cdr conj))))
           )
          (else
           (cons obj conj)))))
```

Em termos de espaço, o processo gerado por `junta-elemento-a-conjunto` apresenta um comportamento  $O(n)$ . O tempo gasto na pesquisa é, em média,  $n/2$ , em que  $n$  representa o número de elementos do conjunto. Sendo assim, o processo em termos de tempo tem uma ordem de crescimento  $O(n)$ .

Na representação de conjuntos com listas ordenadas, a intersecção de dois conjuntos realiza-se percorrendo-os simultaneamente. Verifica-se se o primeiro elemento de um conjunto é igual ao primeiro elemento do outro conjunto. Se assim for, esse elemento pertence à intersecção e avança-se para o elemento seguinte nos dois conjuntos. Sendo diferentes, avança-se apenas para o elemento seguinte no conjunto em que o primeiro elemento é menor.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Ou seja, em cada passo acontece sempre um avanço. Apenas num dos conjuntos, quando os primeiros elementos são diferentes, ou nos dois conjuntos, quando os primeiros elementos são iguais.

```
(define interseccao-de-conjuntos
  (lambda (conj1 conj2)
    (if (or
          (null? conj1)
          (null? conj2))
        '()
        (let ((elem-1 (car conj1))
              (elem-2 (car conj2)))
          (cond
            ((= elem-1 elem-2)
             (cons elem-1
                   (interseccao-de-conjuntos (cdr conj1)
                                              (cdr conj2))))
            ((< elem-1 elem-2)
             (interseccao-de-conjuntos (cdr conj1)
                                       conj2))
            (else
             (interseccao-de-conjuntos conj1
                                       (cdr conj2))))))))
```



**Analise as duas situações consideradas no procedimento, quando `elem-1` e `elem-2` não são iguais.  
Justifique as respectivas implementações.**

Por análise do passo recursivo deste procedimento, pode concluir-se que o tempo gasto na pesquisa é, em média,  $n$ , quando praticamente todos os elementos pertencem à intersecção (situação em que se avança nos dois conjuntos) e  $2^*n$ , quando poucos elementos pertencem à intersecção (situação em que se avança apenas num dos conjuntos). Com  $n^*2 - K^*n$ , o processo, em termos de tempo, tem um comportamento  $O(n)$ , muito melhor que o comportamento  $O(n^2)$  da representação não ordenada.

Continuando a análise do passo recursivo do procedimento, verifica-se que só nos casos em que os elementos de cada conjunto, em comparação, são iguais, é que a operação `cons` fica suspensa, requerendo memória.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



O mesmo não acontece quando os elementos em comparação são diferentes.

Estamos perante um procedimento que, em certas circunstâncias, gera processos recursivos e, noutras, processos iterativos. Assim, em termos de espaço, podemos dizer que esta solução apresenta um comportamento  $O(n)$ , correspondente à pior situação, na qual grande parte dos elementos pertencem à intersecção.

### Exercício 3

Escreva em Scheme o procedimento `uniao-de-conjuntos` na representação de conjuntos com listas ordenadas e sem elementos repetidos e identifique a respectiva ordem de crescimento, em relação ao tempo e ao espaço.



Desenvolva e teste o procedimento `uniao-de-conjuntos`. Teste esta nova versão da abstracção conjunto.



No contexto do projecto **Testa memória**, que alterações deveria introduzir no programa antes de o experimentar se utilizasse esta nova versão da abstracção conjunto, baseada em listas ordenadas e sem elementos repetidos? Justifique.

### Exercício 4

Uma outra representação possível para a abstracção conjunto seria uma lista com os elementos não ordenados, mas permitindo elementos repetidos. Por exemplo, o conjunto  $\{1\ \text{d}\ 4\ 3\}$  poderia ser representado pela lista  $(1\ \text{d}\ 4\ 3)$ , mas também pela lista  $(\text{d}\ 1\ 4\ 1\ \text{d}\ 3)$  e por muitas outras.

Para este tipo de representação, implemente as operações básicas da abstracção conjunto e identifique as respetivas ordens de crescimento.



Desenvolva e teste esta nova versão da abstracção conjunto.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

### Exercício 5

Uma primeira versão da abstracção conjunto baseava a representação dos seus objectos em listas não ordenadas e sem elementos repetidos (versão 1), tendo conduzido às seguintes ordens de crescimento.

	versão 1			versão 2			
	tempo	espaço		tempo	espaço	tempo	espaço
faz-conjunto		$O(1)$		$O(1)$			
elemento-do-conjunto?		$O(n)$		$O(1)$			
junta-elemento-a-conjunto		$O(n)$		$O(1)$			
uniao-de-conjuntos		$O(n^2)$		$O(n)$			
interseccao-de-conjuntos		$O(n^2)$		$O(n)$			

Complete o quadro para os dois outros tipos de representação: um baseado em listas ordenadas de valores numéricos e sem elementos repetidos (versão 2) e outro baseado em listas não ordenadas admitindo elementos repetidos (versão 3).



Faça uma análise comparativa das 3 versões, procurando identificar a que achar melhor.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Módulo 3.4 - Utilizar e construir abstracções para reutilizar código

A reutilização de um código desenvolvido pelo próprio ou por outros é uma prática comum e importante, especialmente se aquele código for digno de confiança.

Neste módulo é mostrado como se pode reutilizar um código que foi desenvolvido e disponibilizado sob a forma de uma abstracção. Neste contexto, são apresentados exemplos de utilização das abstracções janela gráfica e tabuleiro e a criação da abstracção eixos, também esta acompanhada de exemplos de utilização.

### Palavras-Chave

Reutilização de código

# S C H E M E

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



### Abstracção janela gráfica

Os procedimentos gráficos disponibilizados pelas várias implementações da linguagem Scheme exibem, em geral, alguma sofisticação e exigem bastante experiência para uma utilização correcta.

Para os exercícios propostos a partir de agora, que requeiram interacção gráfica, optou-se pela definição e implementação de um conjunto de procedimentos simples que funcionam numa janela gráfica. Trata-se da abstracção janela gráfica que esconde os pormenores dos procedimentos gráficos do DrScheme, dando ao utilizador a hipótese de controlar de uma forma simples, numa janela gráfica, uma caneta que pode desenhar, pintar ou escrever texto, com cores à escolha.

Do ponto de vista da entrada de dados, podem ser lidas as coordenadas do ponto da janela gráfica onde se encontra a caneta, o chamado ponto-corrente, bem como as coordenadas do cursor comandado pelo rato.



A abstracção janela gráfica é apresentada no [Anexo B](#).

No entanto, antes de consultar este anexo, tente encontrar alguma razão que justifique a existência do ponto-corrente (ponto onde se encontra a caneta) e do ponto onde se encontra o cursor (este controlado pelo rato). Fará sentido a existência destes dois pontos?

A apresentação da abstracção janela gráfica encontram-se no [Anexo B](#) e a respectiva utilização não exige qualquer conhecimento dos procedimentos gráficos do DrScheme, como se mostrará em exemplos deste módulo.

<b>INTRODUÇÃO</b>
<b>1 - O ESSENCIAL DO SCHEME</b>
<b>2 - RECURSIVIDADE</b>
<b>3 - ABSTRACÇÃO DE DADOS</b>
R E 1 2 3 4
<b>4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE</b>
<b>5 - ABSTRACÇÕES COM DADOS MUTÁVEIS</b>
<b>6 - SCHEME E OUTRAS TECNOLOGIAS</b>
<b>7-EXERCÍCIOS E PROJECTOS</b>
<b>ANEXOS</b>



Teste a abstracção janela gráfica, que fica acessível através de:  
(require (lib "swgr.scm" "user-feup"))

O objectivo é aprender a usar esta abstracção, pelo que se sugere que experimente os exemplos apresentados nas primeiras páginas do **Anexo B**. Neste e nos próximos exemplos e exercícios, deve ter uma janela gráfica aberta, o que se obtém, por exemplo, com (janela 250 200 "experiencia").

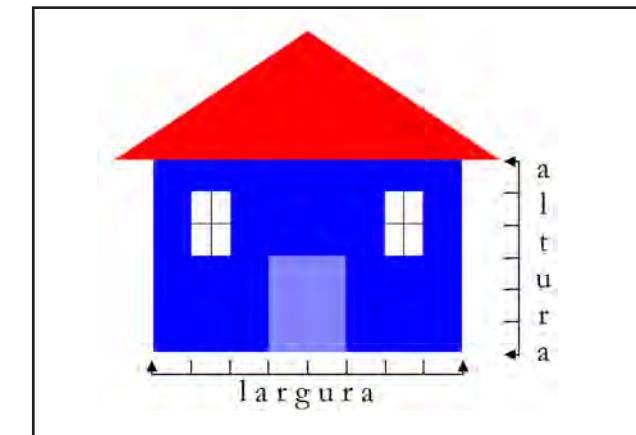
Com os exemplos do **Anexo B**, poderá definir e visualizar uma janela gráfica, escolher a cor da caneta, definir as coordenadas num referencial absoluto (coordenadas da janela) ou em modo relativo (relativo ao ponto-corrente, onde se encontra a caneta), desenhar sequências de segmentos de recta, pintar o interior de polígonos, desenhar e pintar ovais, mover a caneta (sem desenhar nem pintar), mover a caneta e só pintar o ponto onde a caneta é colocada, desenhar texto, ler o ponto-corrente da caneta, ler a posição do cursor controlado pelo rato e limpar a janela. E com isto, é possível imaginar muitos programas com interacção gráfica.

### Exemplo 1 - Utilização da abstracção janela gráfica

Pretende-se desenvolver um programa para visualizar casas no ecrã, todas com o mesmo padrão, mas variando em relação às dimensões (largura e altura) e à posição. As dimensões referidas são fornecidas pelo teclado, enquanto a posição da casa (vértice inferior-esquerdo) é fornecida interactivamente, através do cursor comandado pelo rato.

A figura mostra o padrão das casas.

As dimensões, largura e altura, a fornecer ao programa, referem-se apenas às paredes. Na figura ainda se podem ver as dimensões e o posicionamento da porta, das duas janelas e do telhado. O telhado, no ponto mais alto, mede 2/3 da altura da parede e a sua base é 1.25 da largura.



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

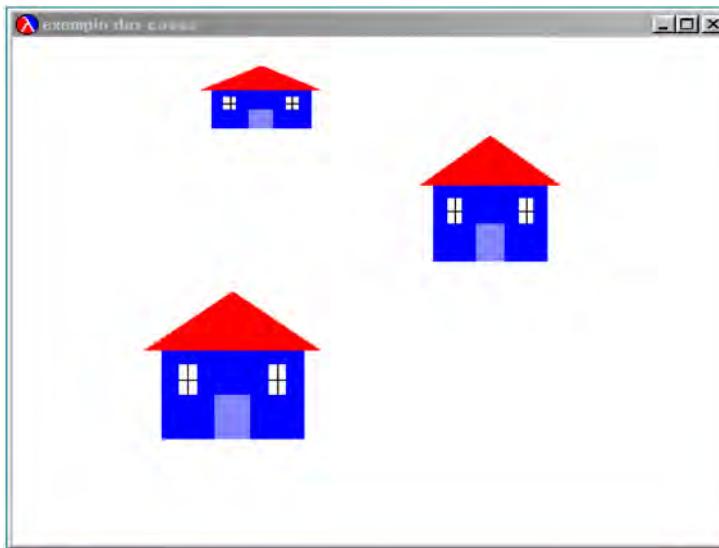
5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

Pode agora imaginar uma sessão com o programa para o posicionamento interactivo de casas.



Os parâmetros associados ao procedimento são, respectivamente, largura, altura e título da janela gráfica que será criada automaticamente e onde decorrerá a visualização das casas.

> (visu-casas 500 400 "exemplo das casas")



Observe a janela com as 3 casas e verifique se as suas características (largura, altura e título) estão de acordo com os argumentos fornecidos ao procedimento no diálogo que se segue.

largura e altura da parede da casa: 100 70  
Com o rato colocar o cursor onde vai surgir a casa...  
e clicar com o botão da esquerda.

a fornecer pelo  
teclado

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



A primeira casa visualizada, a maior das 3, apresenta-se com as dimensões 100 e 70. Como as outras 2, foi posicionada com o cursor que é controlado pelo rato.

Observe o tamanho e posicionamento das outras 2 casas e verifique se estão de acordo com os dados fornecidos pelo utilizador no diálogo que se segue.

```
E' para continuar? (1-sim; outro-nao): 1
largura e altura da parede da casa: 80 60
Com o rato colocar o cursor onde vai surgir a casa...
e clicar com o botão da esquerda.
```

```
E' para continuar? (1-sim; outro-nao): 1
largura e altura da parede da casa: 70 30
Com o rato colocar o cursor onde vai surgir a casa...
e clicar com o botão da esquerda.
```

```
E' para continuar? (1-sim; outro-nao): 0
```

O procedimento visu-casas limita-se a abrir uma janela com as características definidas pelos seus argumentos e a chamar o procedimento desenha-casas para desenhar as casas pretendidas.



O programa visu-casas requer os procedimentos da abstracção janela gráfica, contida no ficheiro **swgr.scm** colocado no directório **user-feup**, que, por sua vez, se encontra em **PLT\collects\** do DrScheme.  
Como fará para incluir esta abstracção no programa?

```
(require (lib "swgr.scm" "user-feup"))

(define visu-casas
  (lambda (lar-janela alt-janela titulo)
    (janela lar-janela alt-janela titulo)
    (desenha-casas)))
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Por seu turno, o procedimento `desenha-casas`, sem parâmetros, pede as dimensões da parede de uma casa, que são lidas pelo teclado e guardadas em duas variáveis locais, `largura` e `altura`.

Em seguida pede que coloque, com o rato, o cursor sobre o ponto a partir do qual vai implantar essa casa. Quando o botão da esquerda é actuado, é lida a posição do cursor. A caneta é movida para o ponto onde se encontra o cursor, deslocando para lá o ponto-corrente e uma chamada ao procedimento `casa` faz visualizar uma casa com as características definidas. Finalmente, o utilizador é questionado sobre se pretende ou não continuar a visualizar mais casas.

```
(define desenha-casas
  (lambda ()
    (display "largura e altura da parede da casa: ")
    (let ((largura (read))
          (altura (read)))
      (display "Com o rato colocar o cursor onde vai surgir a casa...")
      (newline)
      (display "e clicar com o botão da esquerda.")
      (move (cursor))
      (casa largura altura)
      (newline)
      (newline)
      (display "E' para continuar? (1-sim; outro-nao): ")
      (if (equal? 1 (read))
          (desenha-casas)))))
```



Sobre o procedimento `desenha-casas`, explique para que serve e como funciona

- 1- (`move (cursor)`)
- 2- (`casa largura altura`)

O procedimento `desenha-casas` é recursivo. Indique o seu caso base.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

O procedimento casa visualiza uma casa começando pela parede e passando depois à porta, à janela da esquerda, à janela da direita e ao telhado. São definidos localmente os identificadores larg/8, alt/2 e alt/6 para facilitar a escrita do procedimento.

```
(define casa
  (lambda (larg alt) ; largura e altura da parede da casa
    (let ((larg/8 (/ larg 8))
          (alt/2 (/ alt 2))
          (alt/6 (/ alt 6)))
      (parede-casa larg alt)
      (move-rel (list (* 3 larg/8) 0))
      (porta-casa (* 2 larg/8) alt/2)
      (move-rel (list (- (* 2 larg/8)) alt/2))
      (janela-casa larg/8 (* 2 alt/6))
      (move-rel (list (* 5 larg/8) 0))
      (janela-casa larg/8 (* 2 alt/6))
      (move-rel (list (- (* 7 larg/8)) alt/2))
      (telhado-casa (* 1.25 larg) (* alt (/ 2 3))))))
```

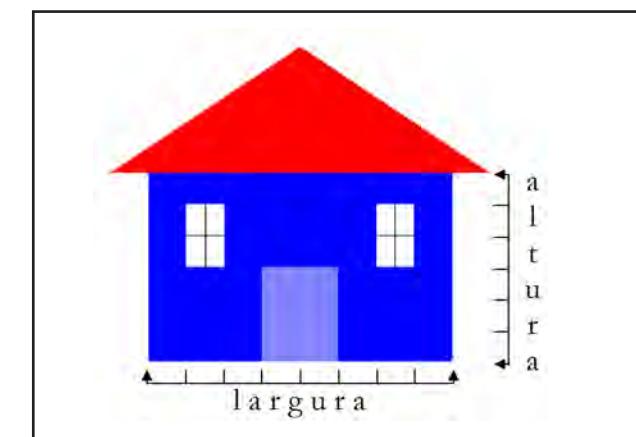


Complete o procedimento casa, com os comentários que achar necessários.

São agora apresentados os procedimentos utilizados para visualizar os vários elementos da casa, acompanhados pelos procedimentos auxiliares rectangulo e cruz. Com excepção destes dois procedimentos, todos os restantes definem a cor utilizada na pintura do respectivo elemento.

```
(define parede-casa
  (lambda (larg alt)
    (cor 2) ; escolhe o azul
    (rectangulo larg alt)))

(define porta-casa
  (lambda (larg alt)
    (cor ... ... ... Para completar
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



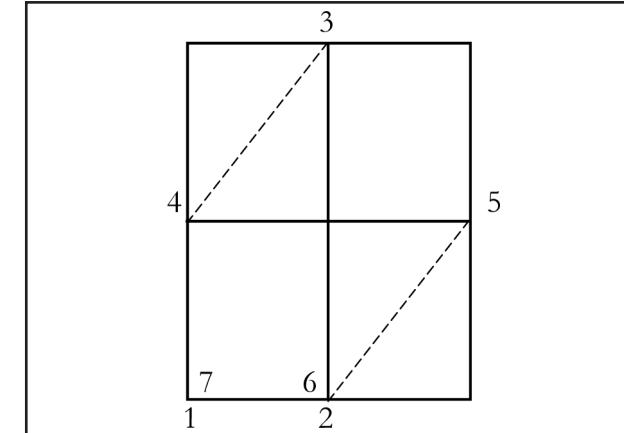
A figura mostra o percurso descrito pela caneta para desenhar uma janela. Em primeiro lugar é desenhado um rectângulo, começando no ponto 1 e acabando nesse mesmo ponto. Depois do rectângulo, a caneta é movida para o ponto 2, desenha a vertical até ao ponto 3, é movida para o ponto 4, desenha a horizontal até ao ponto 5, é movida para o ponto 6, coincidente com o ponto 2 e, finalmente, é movida para o ponto 7, coincidente com o ponto 1.

```
(define janela-casa
  (lambda (larg alt)
    (cor 26) ; escolhe o branco
    (let ((larg/2 (/ larg 2)))
      (rectangulo larg alt)
      (move-rel (list larg/2 0))
      (cor 0) ; escolhe o preto
      (cruz larg alt)
      (move-rel (list (- larg/2) 0)))))

(define telhado-casa
  (lambda (larg alt)
    (cor 18) ; escolhe o vermelho
    (let ... ... ... para completar))

(define rectangulo
  (lambda (larg alt)
    (pinta-rel ... ... ... para completar)

(define cruz
  (lambda (larg alt)
    (desenha-rel (list (list 0 0) (list 0 alt)))
    (move-rel ... ... ... para completar)
```



Complete e teste os procedimentos porta-casa, telhado-casa, rectangulo e cruz e teste o programa visu-casas. Não esqueça de abrir uma janela gráfica para os testes, o que poderá conseguir com, por exemplo, (janela 250 200 "experiencia"). Depois de testar o programa visu-casas, desenhe interactivamente casas de vários tamanhos e posicionadas em locais que escolherá com o rato.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exemplo 2 - Pintura abstracta

Numa outra ocasião, surgiu como exercício a escrita em Scheme do procedimento caminho-errante-2d que esperava, como argumentos, dois valores inteiros, considerados as coordenadas x e y de um ponto-2D, tomado como ponto de partida de um caminho. Para o cálculo do ponto seguinte, o procedimento gerava, aleatoriamente, um número inteiro entre 1 e 3 e quando o número gerado era 1 subtraía 1 à coordenada x, quando era 2 somava 1 a essa mesma coordenada e quando era 3 nada alterava. Depois gerava mais um número aleatório entre 1 e 3 e fazia a mesma coisa para a coordenada y. Este procedimento não tinha fim.

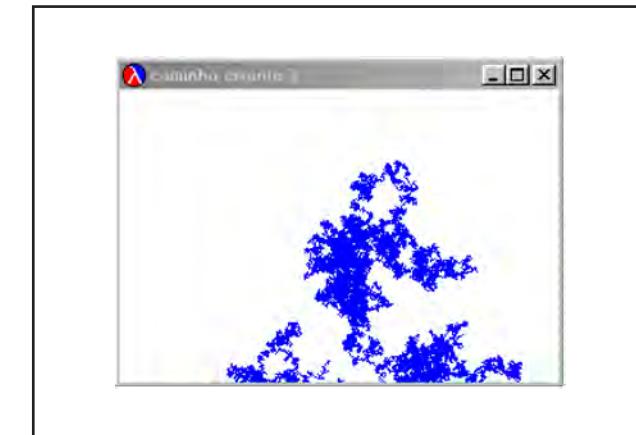
```
> (caminho-errante-2d 20 20)
20:20 21:21 22:20 21:21 21:21 21:20 21:20 22:20 22:19
22:18 21:18 21:19 20:19 21:20 22:20 22:19 21:19

user break
```

Agora propõe-se o desenvolvimento do procedimento caminho-errante-2d-graf que tem como parâmetros ori-x e ori-y, representando as coordenadas do ponto de partida numa janela gráfica, e n-pontos, que define o número de pontos do caminho.

A evolução do percurso segue as mesmas regras da versão caminho-errante-2d, mas agora o resultado é visualizado numa janela gráfica, pois o procedimento vai unindo os pontos gerados através de pequenos segmentos de recta.

```
> (janela 300 200 "caminho errante 1")
(300 200 "caminho errante 1")
> (caminho-errante-2d-graf 150 100 50000)
terminou...
```



O procedimento caminho-errante-2d-graf apresenta uma definição recursiva, em que o caso base corresponde a n-pontos igual a zero, significando que não há mais pontos para visualizar. O caso geral contempla o cálculo do ponto seguinte, de acordo com as regras anteriormente indicadas. É por este facto que são determinados, previamente, dois números aleatórios entre 1 e 3, designados localmente por rol-x e rol-y.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

A utilização de um procedimento auxiliar, recursivo, com parâmetros semelhantes aos de caminho-errante-2d-graf, justifica-se para evitar a repetição da selecção da cor, que é sempre a mesma, em todos os ciclos do processo. Depois do cálculo do ponto seguinte, o procedimento auxiliar desenha o segmento de recta entre o ponto corrente e o ponto calculado e termina chamando-se a si próprio.

```
(define caminho-errante-2d-graf
  (lambda (ori-x ori-y n-pontos)
    ;
    ; definição local de procedimento auxiliar
    ;
    (letrec ((aux
              (lambda (x y n-pontos)
                (if (zero? n-pontos)
                    ;
                    (display "terminou...")
                    ;
                    (let* ((rol-x (roleta-1-n 3))
                           (rol-y (roleta-1-n 3))
                           (x-seg (cond ((= rol-x 1) (+ x 1))
                                         ((= rol-x 2) (- x 1))
                                         (else x)))
                           (y-seg (cond ((= rol-y 1) (+ y 1))
                                         ((= rol-y 2) (- y 1))
                                         (else y))))
                      (desenha (list (list x y) (list x-seg y-seg)))
                      (aux x-seg y-seg (sub1 n-pontos))))))
                  ;
                  (cor 2)      ; escolhe a cor azul
                  (aux ori-x ori-y n-pontos))))
```



Apresente uma explicação para o passo recursivo, definido por (aux x-seg y-seg (sub1 n-pontos))



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

A utilização da abstracção janela gráfica e do procedimento roleta-1-n contido em varios.scm justifica a inclusão, antes da definição do procedimento caminho-errante-2d-graf, de

```
(require (lib "swgr.scm" "user-feup"))
(require (lib "varios.scm" "user-feup"))
```



Teste o procedimento caminho-errante-2d-graf.

Comece por criar uma janela gráfica, com, por exemplo, (janela 250 200 "experiencia"), na qual decorrerão as experiências com este procedimento.

### Exercício 1 - para dominar o desenho em coordenadas relativas

No contexto do exemplo anterior, indique as alterações necessárias ao procedimento caminho-errante-2d-graf para:

- utilizar desenha-rel (que usa coordenadas relativas) em vez de desenha (que usa coordenadas absolutas)
- utilizar o procedimento auxiliar aux apenas com o parâmetro n-pontos, explorando o facto do ponto-corrente ser actualizado automaticamente quando a caneta desenha
- abrir uma janela gráfica onde decorrerá a visualização, cujo título é "caminho-errante" e cujas dimensões são especificadas por mais dois parâmetros do procedimento (estes passam a ser os 2 primeiros parâmetros).



Desenvolva e teste a nova versão do procedimento caminho-errante-2d-graf-v2, numa janela gráfica que poderá criar com, por exemplo, (janela 250 200 "experiencia").



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

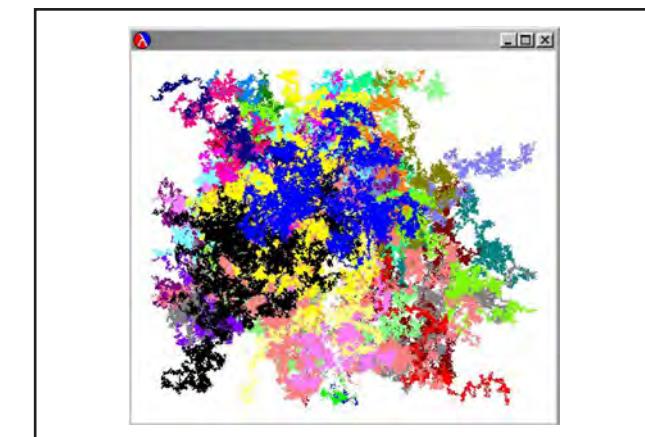
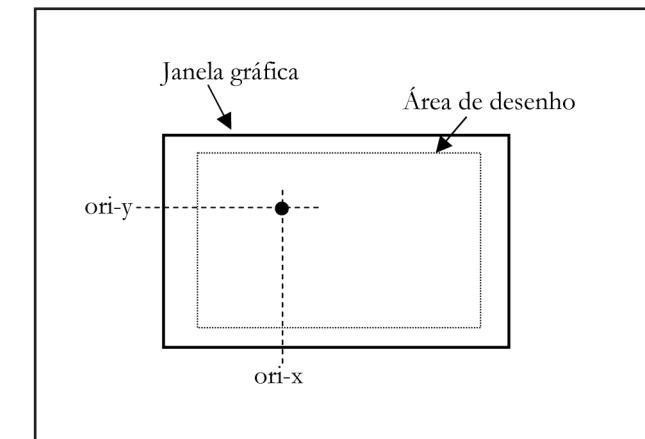
**Exercício 2** - para dominar a manipulação da cor da tinta da caneta

O procedimento caminho-errante-2d-graf utiliza apenas a cor 2 (azul). Pretende-se uma nova versão do procedimento caminho-errante-2d-graf-colorido, que utilize várias cores. A origem do traçado, de coordenadas `ori-x` e `ori-y`, coincide com um ponto no interior da janela gráfica. Agora, ao desenhar, é verificado se o ponto seguinte ultrapassa os limites de uma área rectangular, centrada na janela gráfica, em que os lados têm um comprimento de 90% relativamente ao comprimento dos lados da janela. Sempre que é detectado que o ponto seguinte ultrapassa os limites da área:

- a cor corrente, de índice  $i$ , passa a ser a cor de índice  $i + 1$ , excepto quando  $i = 26$ , situação em que a cor seguinte passa a ser a cor de índice 0;
- o traçado retoma o ponto de partida.

O procedimento `caminho-errante-2d-graf-colorido` tem como parâmetros: `larg-jan`, `alt-jan`, `ori-x`, `ori-y` e `n-pontos`. Os dois primeiros representam a largura e altura da janela gráfica onde a visualização vai ter lugar e os restantes coincidem com os parâmetros utilizados em `caminho-errante-2d-graf`.

Escreva em Scheme o procedimento `caminho-errante-2d-graf-colorido` que, mediante o formato da janela gráfica, a origem do traçado, o número de pontos e a sorte e paciência do utilizador, permita resultados gráficos como o indicado na figura.





Desenvolva e teste o procedimento `caminho-errante-2d-graf-colorido`, numa janela gráfica que poderá criar com, por exemplo, (`janela 250 200 "experiencia"`).

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Abstracção eixos

Nesta secção o objectivo é mostrar, em primeiro lugar, como se pode construir uma abstracção e, seguidamente, como a disponibilizar para futuras utilizações. Com este objectivo em mente, não é essencial preocupar-se muito com os pormenores de implementação da abstracção eixos que, no entanto, serão apresentados.

A representação gráfica de funções do tipo  $y=f(x)$  pode beneficiar da existência de uma maneira flexível de representar sistemas de eixos ortogonais. Com este fim em vista, parte-se para a definição da abstracção eixos. Desta abstracção fazem parte o construtor `eixos` e um conjunto de selectores: `origem-eixos`, `escala-eixos`, `num-divisorias-eixos-xx` e `num-divisorias-eixos-yy`.

- (`eixos origem escala eixo-xx eixo-yy`), procedimento em que os quatro parâmetros são listas de dois elementos.
  - `origem` - lista com as coordenadas da origem do sistema de eixos
  - `escala` - lista com o número de unidades do ecrã entre duas divisões consecutivas do eixo dos `xx` e com o número de unidades do ecrã entre duas divisões consecutivas do eixo dos `yy`
  - `eixo-xx` - lista com o número de divisões na parte negativa do eixo dos `xx` e o número de divisões na parte positiva do mesmo eixo
  - `eixo-yy` - lista com um significado idêntico ao parâmetro `eixo-xx`, mas agora para o eixo dos `yy`

Numa janela gráfica, previamente aberta, este procedimento visualiza um sistema de eixos com as características definidas pelos parâmetros apresentados e, sendo um construtor, devolve um objecto computacional do tipo `eixos`, ou seja, uma lista composta por quatro listas de dois elementos, representando as suas características.

- (`(origem-eixos sistema-de-eixos)`), procedimento com um parâmetro do tipo `eixos`, que devolve uma lista correspondente ao parâmetro `origem` indicado no construtor `eixos`. Trata-se de um selector.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

- (*escala-eixos sistema-de-eixos*), procedimento com um parâmetro do tipo *eixos*, que devolve uma lista correspondente ao parâmetro *escala* indicado no construtor *eixos*. Trata-se de um selector.
- (*num-divisorias-eixos-xx sistema-de-eixos*), procedimento com um parâmetro do tipo *eixos*, que devolve uma lista correspondente ao parâmetro *eixo-xx* indicado no construtor *eixos*. Trata-se de um selector.
- (*num-divisorias-eixos-yy sistema-de-eixos*), procedimento com um parâmetro do tipo *eixos*, que devolve uma lista correspondente ao parâmetro *eixo-yy* indicado no construtor *eixos*. Trata-se de um selector.



Após a implementação desta abstracção, poderia colocá-la em **PLT\collects\user-feup** sob a forma do ficheiro **eixos.scm**. Não se preocupe com esta operação, pois a abstracção *eixos* já se encontra naquele diretório.

Depois, para a utilizar, basta incluir no código

`(require (lib "eixos.scm" "user-feup"))`

Apresenta-se, agora, um exemplo de utilização da abstracção *eixos*, começando pela abertura de uma janela gráfica.

```
> (janela 300 200 "exemplos de eixos")
(300 200 "exemplos de eixos")
> (define exemplo-eixos (eixos '(100 50) '(15 10) '(3 8) '(2 10)))
> (eixos '(10 10) '(10 10) '(0 28) '(0 18))
((10 10) (10 10) (0 28) (0 18))
> (origem-eixos exemplo-eixos)
(100 50)
> (escala-eixos exemplo-eixos)
(15 10)
> (num-divisorias-eixos-xx exemplo-eixos)
(3 8)
> (num-divisorias-eixos-yy exemplo-eixos)
(2 10)
```

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Neste exemplo, a origem dos eixos situa-se no ponto de coordenadas (100, 50). As divisões nos eixos apresentam, entre si, um afastamento de 15 unidades do ecrã no eixo dos xx e de 10 unidades no eixo dos yy. Na parte negativa, o eixo dos xx têm 3 divisões e na parte positiva tem 8. Quanto ao eixo dos yy, são 2 e 10 os números de divisões nas partes negativa e positiva deste eixo.

Passando agora à implementação da abstracção eixos.

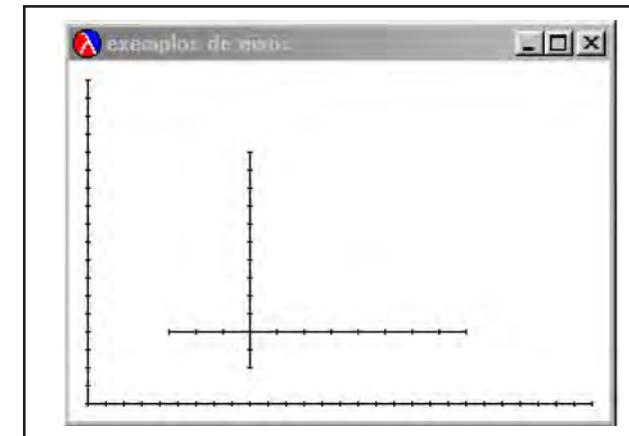
No procedimento eixos são definidos localmente outros procedimentos para desenhar os eixos dos xx e dos yy e as divisões respectivas.

Pode ver, através do corpo principal do procedimento, que o desenho do eixo xx e das suas divisões se inicia no ponto mais à esquerda deste eixo. Por seu turno, o eixo dos yy e respectivas divisões tem origem na parte inferior deste eixo.

Como seria de esperar, as divisões são representadas por segmentos verticais para o eixo dos xx e horizontais para o eixo dos yy. Em ambos os casos, é 2 o comprimento destes segmentos.

A implementação do construtor eixos utiliza procedimentos da abstracção janela gráfica.

```
(require (lib "swgr.scm" "user-feup"))
;
; ----- o construtor eixos -----
;
(define eixos
  (lambda (origem escala eixo-xx eixo-yy)
    (let ((ori-x (car origem))           ; coordenadas da origem dos eixos
          (ori-y (cadr origem)))
        ;
        (escala-x (car escala)) ; número de unidades do ecrã entre
        (escala-y (cadr escala)) ; divisões dos eixos
        ;
        ;                               eixo dos xx
        (n-divisoes-x<0 (car eixo-xx)) ; número divisões parte negativa
        (n-divisoes-x>0 (cadr eixo-xx)) ; número divisões parte positiva
        ;
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
;                                     eixo dos yy
(n-divisoes-y<0 (car eixo-yy)) ; número divisões parte negativa
(n-divisoes-y>0 (cadr eixo-yy)) ; número divisões parte positiva

(letrec ((eixo-x           ; desenha segmento horizontal
          (lambda ()
            (desenha-rel (list (list 0 0)
                                (list (* (+ n-divisoes-x<0
                                           n-divisoes-x>0)
                                         escala-x)
                                      0)))))

;                                     eixo-y           ; desenha segmento vertical
          (lambda ()
            ... .... para completar
          ;

(desivezes-x      ; desenha divisões no eixo dos xx
          (lambda (num)
            (if (not (zero? num))
                (begin
                  (desenha-rel (list (list 0 1) ; sobe 1
                                    (list 0 -2) ; desce 2
                                    (list 0 1))) ; sobe 1
                  (move-rel (list escala-x 0))
                  (divivezes-x (sub1 num)))))) ; mais 1 segmento...
(desivezes-y      ; desenha divisões no eixo dos yy
          (lambda (num)
            ... .... para completar

; ----- o corpo principal do construtor eixos -----
;

; move para o ponto mais à esquerda do eixo xx
;
(move (list (- ori-x (* escala-x n-divisoes-x<0))
            ori-y))

;

(eixo-x           ; desenha o eixo xx

; move para o ponto mais à esquerda do eixo xx
;
(move (list (- ori-x (* escala-x n-divisoes-x<0))
            ori-y))
; desenha divisões do eixo xx
;
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(divisoes-x (+ n-divisoes-x<0 n-divisoes-x>0 1))

; move para o ponto mais baixo do eixo yy
;
(move ... ... ... para completar
;

(eixo-y) ; desenha o eixo yy

; move para o ponto mais baixo do eixo yy
;
(move ... ... ... para completar

; desenha divisões do eixo yy
;
(divisoes-y ... ... ... para completar

; devolve objecto do tipo eixos, com as características definidas
;
(list origem escala eixo-xx eixo-yy))))
;
----- os quatro selectores -----
;
(define origem-eixos
  (lambda (sistema-de-eixos)
    (car sistema-de-eixos)))
(define escala-eixos
  (lambda (sistema-de-eixos)
    (... ... ... para completar

(define num-divisorias-eixos-xx
  (lambda (sistema-de-eixos)
    (... ... ... para completar

(define num-divisorias-eixos-yy
  (lambda (sistema-de-eixos)
    (... ... ... para completar
```



Complete e experimente a abstracção eixos, mas comece por criar uma janela gráfica com o construtor janela.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



No Anexo C - Reutilização de código pode ver como se transforma o código agora desenvolvido numa espécie de caixa-preta, da qual se dará a conhecer apenas o que se desejar. No caso da abstracção eixos, apenas se disponibilizará o construtor e os selectores e ainda o construtor janela da abstracção designada por janela gráfica, abstracção que foi incluída na implementação da abstracção eixos.

Analise com atenção o que se segue.

O texto a cheio indica o que foi acrescentado ao código da abstracção eixo, acabada de desenvolver, para a transformar numa caixa-preta, da qual apenas se dá a conhecer o que for especificado por provide.

```
(module eixos
  mzscheme
  ;
  (require (lib "swgr.scm" "user-feup"))

  (define eixos
    (lambda (origem escala eixo-xx eixo-yy)
      ...
      (define origem-eixos
        ...
        (define escala-eixos
          ...
          (define num-divisorias-eixos-xx
            ...
            (define num-divisorias-eixos-yy
              ...
              (provide eixos)
              (provide origem-eixos)
              (provide escala-eixos)
              (provide num-divisorias-eixos-xx)
              (provide num-divisorias-eixos-yy)

              ; e da abstracção janela gráfica...
              (provide janela)))
```



Em `PLT\collects\user-feup` abra o ficheiro `eixos.scm` e observe a respectiva implementação.  
Abra uma janela e experimente a abstracção `eixos`.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

#### Exemplo 3 - uma utilização da abstracção `eixos`

Numa outra ocasião surgiu como exemplo a escrita em Scheme do procedimento `impar-cima-par-baixo`, com um único parâmetro correspondente a um inteiro positivo. Se esse inteiro fosse par, era dividido por 2. Se fosse ímpar, era multiplicado por 3 e depois somado 1. O resultado assim obtido era sujeito a um tratamento idêntico ao indicado e só terminava quando o resultado fosse 1.

```
> (impar-cima-par-baixo 6)
6 3 10 5 16 8 4 2 1 OK

> (impar-cima-par-baixo 31)
31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310
155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502
251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619
4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577
1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40
20 10 5 16 8 4 2 1 OK
```

Interessa-nos uma visualização gráfica dos resultados deste procedimento, já que em certos casos, como aconteceu com o valor 31, a saída alfanumérica não é muito esclarecedora em relação à forma como os números gerados convergem até ao valor 1. A ideia base reside em criar uma função  $y = f(x)$ , em que  $y$  corresponde aos valores gerados e  $x$  corresponde à ordem de ocorrência. Assim, por exemplo, para a sequência correspondente ao número 6, a  $x=0$  corresponde o valor 6, a  $x=1$  o valor 3, a  $x=2$  o valor 10, ..., a  $x=8$  o valor 1.

O procedimento `impar-cima-par-baixo-graf`, que agora nos interessa, tem os parâmetros `numero`, que representa o valor a processar, e `origem` e `escala` com o mesmo significado do indicado para o construtor `eixos`, apresentado no exemplo anterior.



O procedimento `impar-cima-par-baixo-graf` realiza as seguintes tarefas:

- cria a lista dos números;
  - prepara esta lista para visualizar;
  - visualiza a lista preparada;
  - devolve a lista dos números.

São apresentados uma sessão com este procedimento e os respectivos resultados.

```

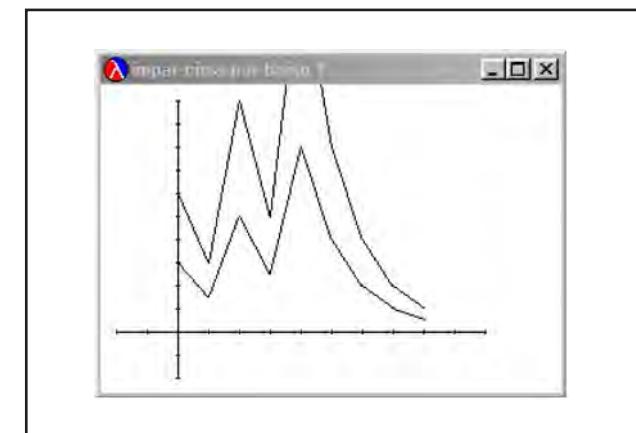
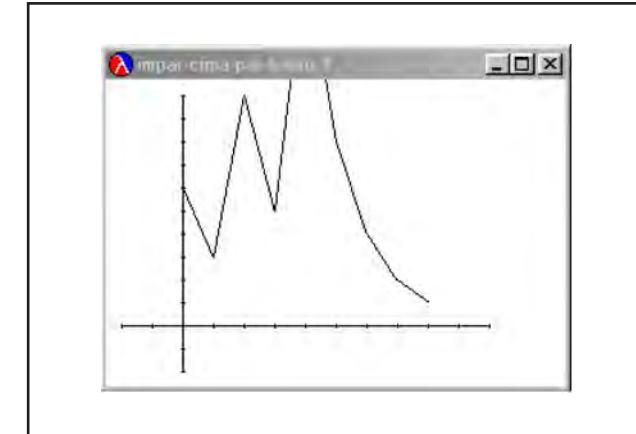
> (eixos '(50 40) '(20 15) '(2 10) '(2 10))
> (impar-cima-par-baixo-graf 6 '(50 40) '(20 15))
(6 3 10 5 16 8 4 2 1)
> (impar-cima-par-baixo-graf 6 '(50 40) '(20 7.5))
(6 3 10 5 16 8 4 2 1)

```

Em ambas as chamadas de **impar-cima-par-baixo-graf**, a origem do traçado é a mesma e é igual à origem dos eixos. Na primeira das duas chamadas, o factor de escala em relação aos  $xx$  e  $yy$  é coincidente com o factor de escala utilizado no desenho dos eixos.

Tal já não acontece na segunda chamada, na qual o factor de escala em relação aos  $yy$  é 7.5, o que significa que os valores de  $y$ , no ecrã, são reduzidos a metade, uma vez que a escala correspondente na chamada do procedimento `eixos` é 15.

Já em relação ao eixo dos  $xx$ , o factor de escala é 10, coincidindo com a correspondente escala na chamada de **eixos**, pelo que os valores de  $x$  são visualizados em verdadeira grandeza. O factor de escala de 7.5 em relação aos  $yy$ , permitiu a visualização completa da função na segunda chamada, o que não aconteceu na primeira.



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

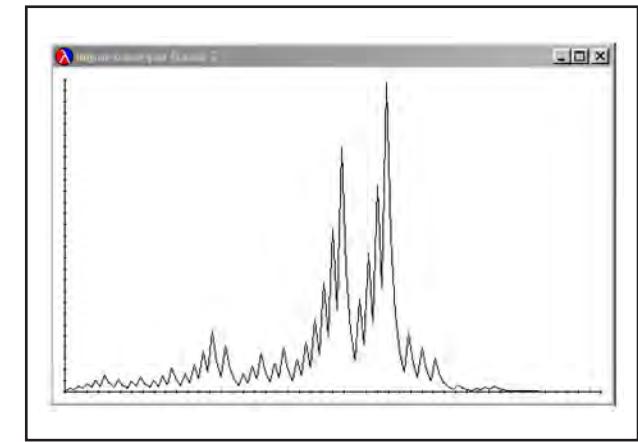
7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

Vejamos outro exemplo, em que o argumento de impar-cima-par-baixo é 31.

```
> (janela 500 300 "impar-cima-par-baixo 2")
(500 300 "impar-cima-par-baixo 2")
> (eixos '(10 10) '(10 10) '(0 48) '(0 28))
> (impar-cima-par-baixo-graf 31 '(10 10) '(4 0.03))
(31 94 47 142 71 214 107 322 161 484 242 121 364 182
91 274 137 412
206 103 310 155 466 233 700 350 175 526 ... 53 160 80
40 20 10 5 16 8 4 2 1)
```

Para a quantidade e gama de números agora gerados, escolheu-se, na chamada impar-cima-par-baixo-graf, uma escala com factores pequenos em x, 4, e muito pequenos em y, 0.03. Este gráfico dá-nos uma ideia da forma como evoluem os números gerados.



O desenvolvimento do procedimento impar-cima-par-baixo-graf apoia-se nas abstracções eixos e janela gráfica, supostas em PLT\collects\user-feup. Justifica-se assim a inclusão de

```
(require (lib "eixos.scm" "user-feup"))
(require (lib "swgr.scm" "user-feup"))
```

As tarefas associadas a impar-cima-par-baixo-graf já foram referidas e são claramente visíveis no corpo deste procedimento.

```
(define impar-cima-par-baixo-graf
  (lambda (numero origem escala)
    (let ((lista-dos-numeros           ; cria a lista dos números
          (lista-impar-cima-par-baixo numero)))
      (desenha           ; visualiza a lista...
        (prepara-lista-para-visualizar ; depois de a preparar
         lista-dos-numeros origem escala))
      ; lista-dos-numeros)))           ; devolve a lista dos números.
```



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

O procedimento lista-impar-cima-par-baixo toma um valor inteiro positivo e devolve a lista com a sequência de números gerados a partir desse valor.

```
(define lista-impar-cima-par-baixo
  (lambda (num)
    (cond ((= num 1) (list 1))
          ((even? num)
           (cons num
                 (lista-impar-cima-par-baixo (quotient num 2)))))
          (else
           (cons num
                 (lista-impar-cima-par-baixo (add1 (* num 3))))))))
```

A preparação de uma lista de pontos-2D, pronta para ser visualizada, a partir da lista dos números gerados, é a tarefa principal do procedimento prepara-lista-para-visualizar. Assim, por exemplo, se o primeiro número da lista é numero1, o primeiro elemento da lista a visualizar deverá ser a lista (ori-x + 0 \* escala-x ori-y + numero1 \* escala-y), e se o segundo é numero2, o segundo elemento da lista a visualizar deverá ser a lista (ori-x + 1 \* escala-x ori-y + numero2 \* escala-y), e assim sucessivamente.

```
(define prepara-lista-para-visualizar
  (lambda (lis origem escala)
    (let ((ori-x (car origem)) ; coordenadas da origem dos eixos
          (ori-y (cadr origem))
          (escala-x (car escala)) ; factores de escala
          (escala-y (cadr escala)))
      (letrec ((aux
                (lambda (lis num)
                  (if (null? lis)
                      '()
                      (cons (list (+ ori-x ; coordenada x
                                    (* num escala-x))
                                  (+ ori-y ; coordenada y
                                    (* (car lis) escala-y)))
                            (aux (cdr lis) (add1 num)))))))
            ;
            (aux lis 0))))))
```



INTRODUÇÃO
1 - O ESSENCIAL DO SCHEME
2 - RECURSIVIDADE
<b>3 - ABSTRACÇÃO DE DADOS</b>
R E 1 2 3 4
4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE
5 - ABSTRACÇÕES COM DADOS MUTÁVEIS
6 - SCHEME E OUTRAS TECNOLOGIAS
7-EXERCÍCIOS E PROJECTOS
ANEXOS



Teste o procedimento `impar-cima-par-baixo`, abrindo previamente uma janela e desenhando nela um sistema de eixos.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

### Exercício 3 - visualização gráfica de listas

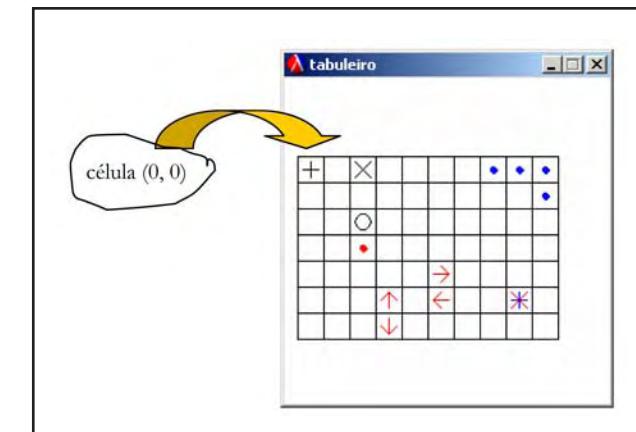
Implemente uma nova abstracção para visualizar entidades do tipo `lista` num sistema de eixos. As listas a visualizar podem ter como elementos valores numéricos positivos e negativos e essa abstracção terá em linha de conta o valor máximo e o valor mínimo contidos na lista e o número dos seus elementos, para escolher as características adequadas do sistema de eixos, numa janela gráfica dada.



Defina, desenvolva e teste uma abstracção gráfica que visualize listas numéricas num sistema de eixos.

### Abstracção tabuleiro

Para responder a situações que fazem uso de um tabuleiro visualizado no ecrã, como acontece com o jogo de damas e labirintos, desenvolveu-se a **abstracção tabuleiro**. Um tabuleiro é criado e visualizado com  $nx \times ny$  células quadradas e o conteúdo visual destas células pode ser alterado colocando em cada uma delas um símbolo à escolha num conjunto de vários símbolos gráficos. Cada célula é identificada por dois índices, situados entre 0 e  $nx - 1$  e 0 e  $ny - 1$ . À célula colocada na parte superior-esquerda correspondem os índices 0 e 0.



Esta abstracção esconde os pormenores dos procedimentos gráficos do DrScheme e permite este tipo de visualização.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Para a utilizar, deverá incluir no seu código

```
(require (lib "tabuleiro.scm" "user-feup"))
```

e abrir uma janela gráfica, suficientemente ampla para conter o tabuleiro.

Desta abstracção fazem parte vários procedimentos.

- (`(tabuleiro org-x org-y lado num-cel-x num-cel-y)`)

em que `org-x` e `org-y` representam a origem do vértice superior-esquerdo do tabuleiro, em coordenadas da janela, `lado` é o comprimento do lado de cada célula em unidades do ecrã e `num-cel-x` e `num-cel-y` representam o número de células, na horizontal e na vertical respectivamente.

Visualiza um tabuleiro com as características especificadas e devolve uma lista com essas características, ou seja, a lista (`(org-x org-y lado num-cel-x num-cel-y)`). Trata-se de um construtor.

- (`(cel-x tab)`)
- (`(cel-y tab)`)

em que `tab` é um tabuleiro.

Devolvem, respectivamente, o número de células na horizontal e o número de células na vertical do tabuleiro.

São ambos selectores.

- (`(celula tab i-x i-y simb cor-actual)`)

em que `tab` é um tabuleiro, `i-x` e `i-y` são os índices que identificam uma célula, `simb` é um símbolo e `cor-actual` é uma cor (conforme a especificação de cor usada na abstracção janela gráfica).

Trata-se de um modificador, pois altera o aspecto visual de uma célula, desenhando nela um símbolo à escolha, de acordo com a seguinte codificação:

+	desenha uma cruz +	x	desenha uma cruz x
c	desenha um círculo	p	desenha um ponto
n	desenha uma seta para norte	s	desenha uma seta para sul
e	desenha uma seta para este	o	desenha uma seta para oeste
l	limpa a célula		

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

- (`celulas tab lis simb espera cor-actual`)

em que `tab` é um tabuleiro, `lis` é uma lista de células (cada uma delas especificada por um par cujos elementos são os índices da célula), `simb` é um símbolo, `espera` é um valor inteiro e `cor-actual` é uma cor (conforme a especificação de cor usada na abstracção janela gráfica).

As células especificadas por `lis` são alteradas visualmente com o símbolo `simb`, a um ritmo condicionado por `espera` (expressa o tempo em milisegundos entre a visualização de células). Trata-se de um modificador.

- (`limpa-tab tab cor-actual`)

em que `tab` é um tabuleiro e `cor-actual` é uma cor (conforme a especificação de cor usada na abstracção janela gráfica).

Todas as células do tabuleiro `tab` são pintadas com a cor especificada. Trata-se de um modificador.

- (`indice-celula tab i-x i-y`)

em que `tab` é um tabuleiro e `i-x` e `i-y` são os índices que identificam uma célula.

Devolve o índice (da tabela de cores) da cor lida no ponto central da célula.

Devolve `-1` se a essa cor não está na tabela de cores. Trata-se de um selector.

Neste contexto, da abstracção gráfica são disponibilizados os procedimentos:

- (`janela largura altura titulo`)

Cria janela gráfica com a dimensão `largura x altura` e com um título (cadeia de caracteres especificada por `titulo`).

Devolve uma lista com as características da janela, ou seja, a lista `(largura altura titulo)`.

- (`limpa`)

Limpa a janela.



# **SCHEME**

## na descoberta da programação

- INTRODUÇÃO**

**ESSÊNCIAL DO SCHEME**

**- RECURSIVIDADE**

**STRACÇÃO DE DADOS**  
R E 1 2 3 4

**OCEDIMENTOS COM  
JECTOS DE 1<sup>a</sup> CLASSE**

**ABSTRACTOES COM  
DADOS MUTÁVEIS**

**SCHEME E OUTRAS  
TECNOLOGIAS**

**7 - EXERCÍCIOS  
E  
PROJECTOS**

**ANEXOS**

O tabuleiro da figura foi criado numa sessão equivalente à que se segue.

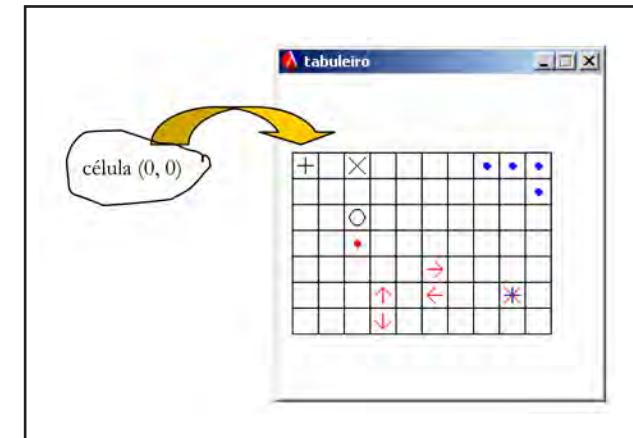
```
> (require (lib "tabuleiro.scm" "user-feup"))

> (janela 250 250 "tabuleiro")
(250 250 "tabuleiro")

> (define tab1 (tabuleiro 10 190 20 10 7))
> (cel-x tab1)
10

> (cel-y tab1)
7

> (celula tab1 0 0 '+ 0)  visualiza + a preto, na célula 0,0;
> (celula tab1 2 0 'x 0)  visualiza x a preto, na célula 2,0;
> (celula tab1 3 0 'X 0)  símbolo não previsto... erro;
erro: comando desconhecido!
```



```
> (celula tab1 2 2 'c 0)    visualiza 0 a preto, na célula 2,2;  
> (celula tab1 2 3 'p 18)   visualiza . a vermelho, na célula 2,3;  
> (celula tab1 3 5 'n 18)   visualiza seta para Norte a vermelho, na célula 3,5;  
> (celula tab1 3 6 's 18)   visualiza seta para Sul a vermelho, na célula 3,6;  
> (celula tab1 5 4 'e 18)   visualiza seta para Este na célula 5,4;  
> (celula tab1 5 5 'o 18)   visualiza seta para Oeste na célula 5,5;  
> (celula tab1 8 5 '+ 2)    visualiza caracteres sobrepostos  
> (celula tab1 8 5 'x 18)      na célula 8,5;  
  
> (celula tab1 8 6 '+ 2)    visualiza + a azul, na célula 8,6  
> (celula tab1 8 6 'l 26)      para limpar (pintar a célula a branco) em seguida;  
                                vai visualizar uma sequência de 4 pontos azuis,  
                                perto do canto superior-direito, ao ritmo de 1 por segundo  
> (celulas tab1 (list (cons 7 0) (cons 8 0) (cons 9 0) (cons 9 1)) 'p 1000 2)
```



Experimente a abstracção tabuleiro, colocada em PLT\collects\user-feup sob a forma do ficheiro tabuleiro.scm. Não esqueça de abrir uma janela, como acontece, por exemplo, na sessão acabada de apresentar.

# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

R E 1 2 3 4

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

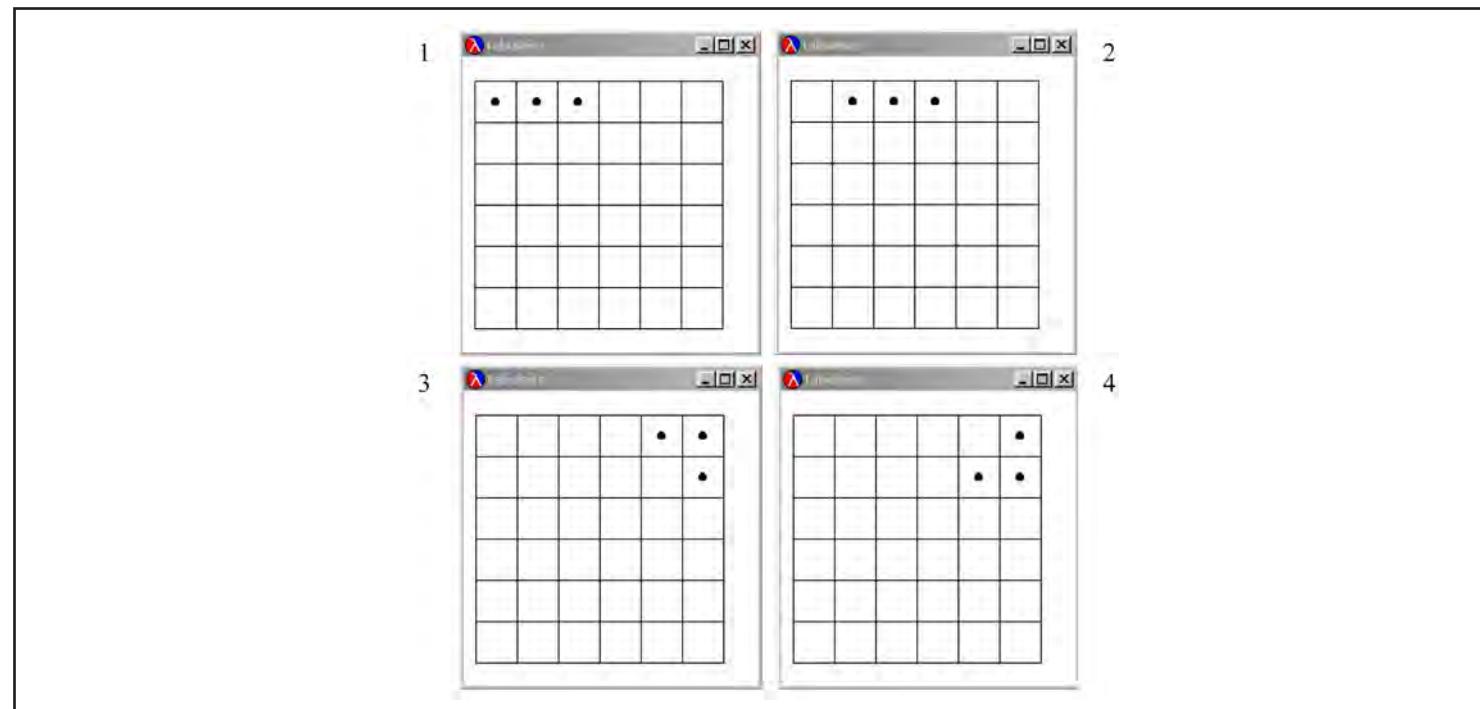
6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

### Exercício 4 - uma utilização da abstracção tabuleiro

Num tabuleiro de 6 x 6 células quadradas com 35 unidades de ecrã de lado, visualize o percurso de um pequeno animal, representado por um ponto negro centrado nas células, desde o canto superior esquerdo até ao canto inferior direito. O pequeno animal percorre a primeira linha da esquerda para a direita a segunda linha no sentido contrário, a terceira linha novamente da esquerda para a direita e assim sucessivamente até atingir o canto inferior direito. O animal deixa um rastro que se vai apagando com o tempo, ou seja, esse rastro terá um comprimento máximo de 3 células. Assim, um ponto começa por aparecer na célula 0, 0, depois um segundo ponto aparece na célula 0, 1 e um terceiro na célula 0,2 (figura 1). De seguida, é apagado o ponto da célula 0, 0 e aparece um novo ponto na célula 0, 3 (figura 2). As figuras 3 e 4 mostram o rastro numa "curva".



Desenvolva e teste uma aplicação da abstracção tabuleiro que simule a deslocação de um pequeno animal.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

## Resumo dos módulos 4.1 e 4.2

Resumo dos assuntos abordados nos módulos

Módulo 4.1 - **Os procedimentos não param de surpreender**

Módulo 4.2 - **Um procedimento pode devolver outro procedimento**



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Resumo dos assuntos abordados nos módulos 4.1 e 4.2

Neste conjunto de dois módulos, o objectivo principal é mostrar que os procedimentos em Scheme também são objectos de 1<sup>a</sup> classe, objectos que podem ser manipulados, como acontece com os números, booleanos, símbolos e pares. As características dos objectos de 1<sup>a</sup> classe aplicam-se aos procedimentos, pois os procedimentos ligam-se a identificadores e podem ser argumentos de outros procedimentos, elementos de estruturas de dados e ainda valores de retorno de outros procedimentos.

Num pequeno desvio inicial, mostra-se como se definem procedimentos que podem ser chamados com um número não fixo de argumentos.

### Módulo 4.1

As propriedades que caracterizam os objectos de 1<sup>a</sup> classe também se aplicam aos procedimentos e a comprovação deste facto inicia-se neste módulo.

Este módulo começa com um pequeno desvio ao tema e apresenta mais uma novidade sobre os procedimentos: os procedimentos podem admitir um número não fixo de argumentos. Ou seja, um procedimento pode ser chamado com qualquer número de argumentos, inclusivamente com zero. Depois deste desvio, entra-se numa primeira abordagem aos procedimentos como objectos de 1<sup>a</sup> classe, mostrando duas das suas características: podem ser argumentos de outros procedimentos e também elementos de estruturas de dados. Neste percurso, surgem naturalmente os procedimentos que aceitam outros procedimentos como argumentos, designados por procedimentos de ordem mais elevada, e deles são apresentados alguns exemplos, tanto de procedimentos primitivos do Scheme (`map`, `for-each` e `apply`), como de procedimentos definidos no decorrer dos módulos (`max-fun` e `somatorio`).

### Módulo 4.2

Os procedimentos são frequentemente associados a identificadores (identificadores que passam a ser o nome desses procedimentos) e podem ser utilizados e desenvolvidos procedimentos que aceitam outros procedimentos como argumentos (procedimentos de ordem mais elevada). Os procedimentos também podem ser elementos de estruturas de dados.



# **S C H E M E**

## **na descoberta da programação**

**INTRODUÇÃO**

**1 - O ESSENCIAL DO SCHEME**

**2 - RECURSIVIDADE**

**3 - ABSTRACÇÃO DE DADOS**

**4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE**

**R E 1 2**

**5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS**

**6 - SCHEME E OUTRAS  
TECNOLOGIAS**

**7-EXERCÍCIOS  
E  
PROJECTOS**

**ANEXOS**

Todas estas características são comprovadas no contexto do módulo anterior.

Neste módulo surge mais uma característica dos procedimentos: podem ser o retorno de outros procedimentos.

Com esta característica dos procedimentos completa-se o conjunto de características que qualifica os procedimentos como objectos de 1<sup>a</sup> classe.



## **Exercícios e Exemplos**

São apresentados alguns exercícios e exemplos para consolidação da matéria tratada nos módulos 4.1 e 4.2.

### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

R E 1 2

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

### Exemplo 1

O procedimento primitivo `max` aceita um número não fixo de argumentos e devolve o valor do maior argumento. Escreva em Scheme uma versão pessoal de `max`, designada por `max-pessoal`.

```
(define max-pessoal
  (lambda args
    (letrec ((max-aux
              (lambda (lis)
                (cond
                  ((null? (cdr lis))
                   (car lis))
                  (else
                    (let
                      ((primeiro (car lis))
                       (maior (max-aux (cdr lis))))
                      (if (< primeiro maior)
                          maior
                          primeiro))))))
                  (if (null? args)
                      (display "erro. Numero incorrecto de argumentos")
                      (max-aux args)))))))
```

Esta primeira versão de `max-pessoal` baseia-se no procedimento interno `max-aux` que gera processos recursivos, com  $O(n)$  em tempo e espaço, em que  $n$  está relacionado com o número de valores a processar. A ideia utilizada é, à partida, muito simples, mas não é fácil de acompanhar: toma-se o primeiro valor a processar que é comparado com o maior valor encontrado entre os restantes...



Siga, passo a passo, o funcionamento da chamada (`max-pessoal 2 7 1 3`).

A versão seguinte baseia-se numa ideia simultaneamente simples e muito fácil de acompanhar: o primeiro valor é considerado o maior valor até essa altura e torna-se o **máximo-temporário**; o **máximo-temporário** é comparado com o valor seguinte e o maior entre eles passa a ser o **máximo-temporário**; a comparação repete-se até não haver mais valores, estando a resposta no **máximo-temporário**.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
(define max-pessoal-v2
  (lambda args
    (letrec ((max-aux-iter
              (lambda (lis max-temp)
                (cond
                  ((null? lis)
                   max-temp)
                  ((> (car lis) max-temp)
                   (max-aux-iter (cdr lis) (car lis)))
                  (else
                   (max-aux-iter (cdr lis) max-temp))))))
      (if (null? args)
          (display "erro. Numero incorrecto de argumentos")
          (max-aux-iter (cdr args) (car args)))))))
```



Teste as duas versões do procedimento max-pessoal.



Indique e justifique a ordem de complexidade dos processos gerados pela versão max-pessoal-v2.



Siga, passo a passo, o funcionamento da chamada (max-pessoal-v2 2 7 1 3).

### Exercício 1

Escreva em Scheme o procedimento pi-aprox, baseado em somatorio, partindo da fórmula:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots$$



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Definir pi-aprox, sabendo que este tem apenas um parâmetro, n, e que devolve o valor de pi, aproximado aos primeiros 2\*n termos da fórmula.



Desenvolva e teste o procedimento pi-aprox.

### Exercício 2

O valor de e pode ser aproximado através da fórmula  $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$

Escreva em Scheme o procedimento e-aprox, com o parâmetro n, e devolve o valor de e aproximado aos primeiros n termos da fórmula.



Desenvolva e teste o procedimento e-aprox.

### Exercício 3

O golden ratio pode ser definido pela fórmula  $\Phi = \frac{1 + \sqrt{5}}{2} \approx 1.61803$

e calculado através de  $\Phi = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \dots}}}}$

Escreva em Scheme o procedimento golden-aprox, com o parâmetro n, que devolve o valor do golden ratio aproximado aos primeiros n termos da fórmula.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Desenvolva e teste o procedimento `golden-aprox.`

### Exercício 4

A fórmula que se apresenta é uma aproximação do integral de uma função  $f(x)$ , entre  $a$  e  $b$ . Verifique, graficamente, que a aproximação é tanto melhor quanto menor for  $dx$ .

Neste contexto, escreva em Scheme o procedimento `integral`, baseado em somatorio.

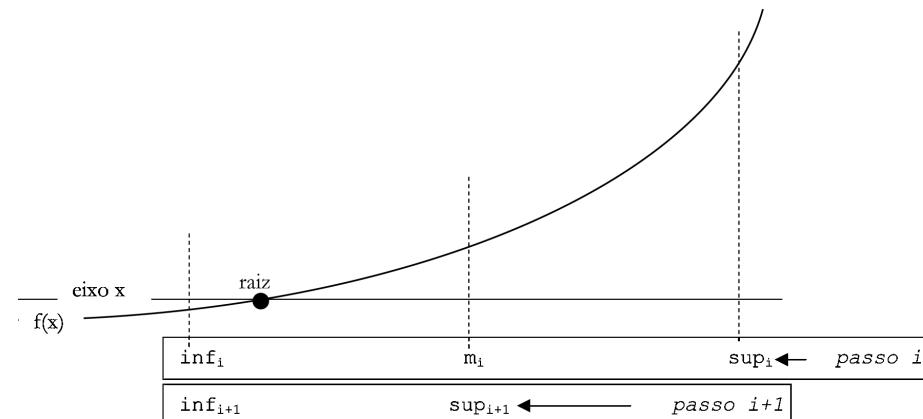
$$\int_a^b f(x) = \left[ f(a + \frac{dx}{2}) + f(a + dx + \frac{dx}{2}) + f(a + 2dx + \frac{dx}{2}) + \dots \right] dx$$



Desenvolva e teste o procedimento `integral`.

### Exercício 5

O método da bissecção, para encontrar as raízes de funções, é esquematizado na figura.





Desenvolva e teste o procedimento bisseccao.

## Exemplo 2

Escreva em Scheme o procedimento `compoе` que tem os procedimentos `p1` e `p2` como parâmetros e devolve um procedimento de um só parâmetro, `x`. O procedimento resultante aplica `p1` a `p2(x)`, ou seja, `p1(p2(x))`.

```
(define compoe
  (lambda (f g)
    (lambda (x)
      (f (g x)))))

> (define soma-1-e-eleva-a-2 (compoе (lambda (x) (* x x)) add1))
> (soma-1-e-eleva-a-2 9)
100
```



Teste o procedimento compõe.

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



### Exercício 6

Escreva em Scheme o procedimento `compoe3` que tem os procedimentos `p1`, `p2` e `p3` como parâmetros e devolve um procedimento de um único parâmetro, `x`. O procedimento resultante faz `p1 (p2 (p3 (x)))`.



Desenvolva e teste o procedimento `compoe3`.

### Exercício 7

Defina o procedimento `compoe-muitas`, que responde da seguinte maneira:

```
> ((compoe-muitas add1 add1 add1 add1) 3)  
7
```



Desenvolva e teste o procedimento `compoe-muitas`.

Pista: Utilizar procedimentos com um número não fixo de argumentos.

### Exercício 8

Escreva em Scheme o procedimento `repetir-fun` que toma como argumentos uma função numérica, `f`, e um inteiro positivo, `n`, e devolve a função `f (f (... (f (x)) ...))`, ou seja, a função `f` aplicada sobre si mesma `n` vezes.

```
> ((repetir-fun add1 3) 5) ;é equivalente a (add1 (add1 (add1 5)))  
8
```



Desenvolva e teste o procedimento `repetir-fun`.

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

## Exemplo 3

Analise as chamadas de **newton-raphson** que se seguem.

Na primeira, determina-se o valor de  $x$  para o qual  $x^2 - 9 = 0$ , equivalente a  $x = \sqrt{9}$ . O zero da função  $f(x) = x^2 - 9$  coincide com a raiz quadrada de 9.

```
> (newton-raphson (lambda (x) (- (* x x) 9)) 1)
3.000000174227237
```

Na segunda, determina-se o valor de  $x$  para o qual  $x^3 - 9 = 0$  equivalente a  $x = \sqrt[3]{9}$ . Aqui, o zero da função  $f(x) = x^3 - 9$  coincide com a raiz cúbica de 9.

```
> (newton-raphson (lambda (x) (- (* x x x) 9)) 1)
2.080083834331853
```

Finalmente, na terceira, o zero da função  $f(x) = x^3 - 8$  coincide com a raiz cúbica de 8.

```
> (newton-raphson (lambda (x) (- (* x x x) 8)) 1)
2.000000033068637
```

Destes exemplos deduz-se uma ideia para a escrita de procedimentos que calculam a raiz quadrada e a raiz cúbica, como casos particulares do procedimento *newton-raphson*.

```
(define raiz-quadrada
  (lambda (numero)
    (newton-raphson (lambda (x) (- (* x x) numero)) 1)))

(define raiz-cubica
  (lambda (numero)
    (newton-raphson (lambda (x) (- (* x x x) numero)) 1)))

> (raiz-quadrada 48)
6.928203438125418
> (sqrt 48)           (o mesmo cálculo recorrendo directamente a
                        um procedimento do Scheme)
6.928203230275509
```



Teste os procedimentos *raiz-quadrada* e *raiz-cubica*.



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



### Exercício 9 - para pensar um bocado...

Desenvolva o procedimento `raiz`, tomando por base outros procedimentos já desenvolvidos, nomeadamente, `repetir-fun` ([exercício anterior](#)) e `newton-raphson`. O procedimento `raiz` apresenta como parâmetros `n` e `num` e devolve a raiz de ordem `n` de `num`.

```
> (raiz 2 49)          (é equivalente a x = √49)  
7  
> (raiz 5 243)         (é equivalente a x = ³√243)  
3
```



Desenvolva e teste o procedimento `raiz`.

### Exemplo 4

Escreva em Scheme o procedimento `faz-ordenador` que aceita como argumento um operador de comparação de dois elementos e devolve um procedimento com um único parâmetro, que representa uma lista. O procedimento devolvido, quando chamado, ordena os elementos da lista que aceita como argumento de acordo com o operador de comparação. As chamadas ajudam a esclarecer o que foi dito sobre `faz-ordenador`.

```
> (define ordenador-crescente (faz-ordenador <))  
> (ordenador-crescente '(23 5 37 1 -8))  
(-8 1 5 23 37)  
> (define ordenador-decrescente (faz-ordenador >))  
> (ordenador-decrescente '(23 5 37 1 -8))  
(37 23 5 1 -8)
```

A solução que se apresenta merece algumas notas:

- `faz-ordenador` devolve um procedimento com um parâmetro que é uma lista e cujo corpo se limita a uma chamada do procedimento `ordena`;

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

ANEXOS

- No passo recursivo do procedimento `ordena`, é chamado o procedimento `insere-por-comparacao`, que insere um elemento numa lista previamente ordenada, continuando esta ordenada após a inserção do elemento. Isto poderá parecer estranho, mas não é mais do que a recursividade a funcionar. De facto, o terceiro argumento da chamada deste procedimento é (`ordena comparacao (cdr lista)`) que deverá devolver a lista resultante de (`cdr lista`) devidamente ordenada...

```
(define faz-ordenador
  (lambda (comparacao)
    (lambda (lista)
      (ordena comparacao lista)))))

(define ordena
  (lambda (comparacao lista)
    (if (null? lista)
        '()
        (insere-por-comparacao (car lista)
                               comparacao
                               (ordena comparacao (cdr lista))))))

; insere um elemento numa lista ordenada,
; continuando ordenada, após a inserção
(define insere-por-comparacao
  (lambda (elem compara lis-ordenada)
    (cond ((null? lis-ordenada) (list elem))
          ((compara elem (car lis-ordenada))
           (cons elem lis-ordenada))
          (else
            (cons (car lis-ordenada)
                  (insere-por-comparacao elem compara
                                         (cdr lis-ordenada)))))))
```



Teste o procedimento `faz-ordenador`.





Caberá ao programador decidir entre construir ordenadores com o procedimento `faz-ordenador` ou então utilizar directamente o procedimento `ordena`, indicando qual o operador de comparação que pretende.  
Comente esta opinião.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 10

Para se obter uma solução iterativa do procedimento considerado no exemplo anterior, agora designada por `faz-ordenador-iter`, complete o que se segue, escrevendo o procedimento `ordena-iter`.

```
(define faz-ordenador-iter
  (lambda (comparacao)
    (lambda (lista)
      (ordena-iter comparacao lista '()))))

(define ordena-iter
  (lambda (comparacao lista lista-ordenada)
    ... para completar ...))
```



Desenvolva e teste o procedimento `faz-ordenador-iter`.

Pista: O parâmetro `lista-ordenada` é uma lista inicialmente vazia, como se verifica na chamada de `ordena-iter`. À medida que surgem elementos de lista para ordenar, cada um deles é inserido em `lista-ordenada`, na posição correcta, de acordo com a regra de ordenação. Assim, garante-se que `lista-ordenada` se encontra sempre ordenada. A inserção de mais um elemento numa lista já ordenada pode ser conseguida pela utilização do procedimento `insere-por-comparacao` do exemplo anterior.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Contrariamente ao que é normal, a solução iterativa apresenta-se mais fácil de entender do que a solução recursiva do exemplo anterior.  
Concorda?

### Exercício 11

Escreva em Scheme o procedimento `faz-funcao-acesso-a-lista` que aceita como argumento um inteiro positivo e devolve um procedimento com um único parâmetro, que representa uma lista. O procedimento devolvido, quando chamado, acede e devolve um elemento da lista, conforme se pode verificar nas chamadas que se seguem.

```
> (define terceiro (faz-funcao-acesso-a-lista 2))
```

O terceiro elemento de uma lista está na posição 2, considerando que na posição zero está o primeiro elemento. Por isso, na criação de `terceiro`, foi utilizado 2 como argumento de `faz-funcao-acesso-a-lista`.

```
> (terceiro '(p0 p1 p2 p3 p4 p5 p6 p7))
p2
> (define setimo (faz-funcao-acesso-a-lista 6))
> (setimo '(p0 p1 p2 p3 p4 p5 p6 p7))
p6
> (define decimo (faz-funcao-acesso-a-lista 9))
> (decimo '(p0 p1 p2 p3 p4 p5 p6 p7))
erro: lista pequena
```



Desenvolva e teste o procedimento `faz-funcao-acesso-a-lista`.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

### Exercício 12

#### Projecto - CODEC

Pretende-se desenvolver um conjunto de procedimentos para codificar e decodificar mensagens. Para este efeito, considera-se que uma mensagem é uma lista de símbolos formados por uma letra, como, por exemplo, '(a d f e g h i).

A codificação baseia-se na substituição dos símbolos originais por outros, de acordo com um código preparado manualmente:

```
> (define codigo-1 '((a 1) (b 2) (c 3)))
> (define codigo-2 '((a b) (b t) (c 4) (d 1)))
```

Segundo `codigo-1`, o alfabeto a utilizar nas mensagens é composto pelas letras a, b e c, às quais corresponde a codificação 1, 2, e 3, respectivamente. Qualquer letra fora do alfabeto é substituída, na mensagem codificada, pelo símbolo `erro`. Para `codigo-2`, o alfabeto utilizado abarca as 4 primeiras letras do abecedário, às quais corresponde a codificação b, t, 4, e 1, respectivamente.

O procedimento `faz-codificador` aceita apenas um argumento que deverá ser um código, previamente definido, e devolve um procedimento com um único parâmetro. A este parâmetro, em cada chamada, deverá ser associada a mensagem a codificar.

```
> (define codificador-1 (faz-codificador codigo-1))
> (codificador-1 '(a b c b a))
(1 2 3 2 1)
> (codificador-1 '(a b c b x y a))
(1 2 3 2 erro erro 1)
> (define codificador-2 (faz-codificador codigo-2))
> (codificador-2 '(a b c b a))
(b t 4 t b)
> (codificador-2 '(a b c b x y a))
(b t 4 t erro erro b)
> (codificador-1 (codificador-2 '(a b c b a)))
(2 erro erro erro 2)
```



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

R E 1 2

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

ANEXOS



Analise agora as chamadas que se seguem, relacionadas com a descodificação de mensagens.

```
> (define descodificador-1 (faz-descodificador codigo-1))
> (descodificador-1 '(1 2 3 2 1))
(a b c b a)
> (descodificador-1 '(1 2 3 2 erro erro 1))
(a b c b erro erro a)
```

1- Escreva em Scheme os procedimentos faz-codificador e faz-descodificador.

2- Para uma maior segurança, os códigos passarão a ser criados automaticamente. Para isso, escreva o procedimento cria-código, com o parâmetro alfabeto, lista com os símbolos correspondentes aos caracteres de um alfabeto, que devolve um código gerado aleatoriamente (o formato é semelhante ao dos códigos utilizados). No entanto, o procedimento deverá garantir que nos códigos criados não haverá caracteres do alfabeto com a mesma codificação. Considere que a codificação de qualquer carácter será sempre um carácter do próprio alfabeto.

```
> (define c1 (cria-código '(a b c 1 2 3)))
> c1
((a 1) (b a) (c 3) (1 2) (2 c) (3 b))
> (define codificador-1 (faz-codificador c1))
> (define descodificador-1 (faz-descodificador c1))
> (codificador-1 '(a b c d e 4 3 2 1))
(1 a 3 erro erro erro b c 2)
> (descodificador-1 '(1 a 3 erro erro erro b c 2))
(a b c erro erro erro 3 2 1)
> (descodificador-1 (codificador-1 '(a b f 2 2 5 c 1 3)))
(a b erro 2 2 erro c 1 3)
```



Desenvolva o projecto CODEC e realize os testes que achar necessários.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Módulo 4.1 - Os procedimentos não param de surpreender

Pode parecer estranho mas, em Scheme, os procedimentos são também considerados objectos de 1<sup>a</sup> classe. E isto surpreende porque, normalmente, a classificação de objectos de 1<sup>a</sup> classe está apenas reservada aos números, booleanos, pares e símbolos, entidades que podem ser processadas e, por isso, reconhecidas como objectos passivos, contrariamente ao que sucede com os procedimentos que são objectos activos devido às actividades que lhes estão associadas.

As propriedades que caracterizam os objectos de 1<sup>a</sup> classe também se aplicam aos procedimentos, e a comprovação deste este facto inicia-se neste módulo e é completada no módulo seguinte.

O presente módulo começa com a apresentação de mais uma novidade sobre os procedimentos em Scheme: os procedimentos podem admitir um número não fixo de argumentos. Ou seja, um procedimento pode ser chamado uma vez com 2 argumentos e da vez seguinte com 3, 4 ou até zero argumentos.

Depois deste desvio, o módulo faz uma primeira abordagem aos procedimentos como objectos de 1<sup>a</sup> classe, mostrando as respectivas características. Neste percurso, vai encontrar os procedimentos de ordem mais elevada, assim designados por admitirem como parâmetros outros procedimentos. Os procedimentos que não admitem procedimentos como parâmetros são designados por procedimentos de 1<sup>a</sup> ordem.

### Palavras-Chave

Objectos de 1<sup>a</sup> classe, procedimentos com um número não fixo de argumentos, procedimentos que são argumentos de outros procedimentos, procedimentos de 1<sup>a</sup> ordem, procedimentos de ordem mais elevada, procedimentos como elementos de estruturas de dados.



## Procedimentos com um número não fixo de argumentos

Os procedimentos desenvolvidos até aqui apresentavam, como característica comum, o facto de o número de argumentos utilizados nas chamadas coincidir sempre com o número de parâmetros especificados, entre parêntesis, a seguir ao `lambda`. Em todos estes casos, o número de argumentos ficava estabelecido quando se definiam os parâmetros dos procedimentos.



Antes de avançar, pense um pouco e tente identificar alguns procedimentos primitivos do Scheme que não cumpram esta regra, ou seja, que admitam chamadas com um número diferente de argumentos.

O próprio Scheme disponibiliza procedimentos que não seguem esta regra, pois aceitam um número não fixo de argumentos. São disto exemplo os operadores aritméticos ( $+$ ,  $-$ ,  $*$ ,  $/$ ,  $<$ ) e outros procedimentos primitivos.

```
> (+ 1 2)           (chamada com 2 argumentos)
3
> (+ 1 2 3)        (chamada com 3 argumentos)
6
```

Os procedimentos com um número fixo de argumentos são definidos colocando uma lista de parâmetros a seguir a lambda.

```
(define proc-com-numero-fixo-de-argumentos
  (lambda (param-1 param-2 param-3 ...)
    corpo-do-procedimento))
```

Os procedimentos com um número não fixo de argumentos são definidos de uma forma diferente. Neste caso, imediatamente a seguir a `lambda`, encontra-se não uma lista com um certo número de parâmetros, mas apenas um identificador à escolha.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(define proc-com-numero-nao-fixo-de-argumentos
  (lambda argumentos
    corpo-do-procedimento))
```

Com a chamada

```
(proc-com-numero-nao-fixo-de-argumentos arg1 arg2 ...)
```

o identificador argumentos fica associado à lista (arg1 arg2 ...), constituída por todos os argumentos utilizados, cujo número poderá variar em cada chamada.



**Em cada uma das chamadas seguintes, indique que estrutura de dados ficará associada ao identificador argumentos.**

```
> (proc-com-numero-nao-fixo-de-argumentos 1 2 3)
> (proc-com-numero-nao-fixo-de-argumentos)
> (proc-com-numero-nao-fixo-de-argumentos 4 (list 5 12) 67)
```

### Exemplo 1

Escreva em Scheme o procedimento soma-primeiro-e-ultimo, que devolve a soma do primeiro e do último argumentos que lhe são passados em cada chamada.

```
> (soma-primeiro-e-ultimo 1 2) ; (chamada com 2 argumentos)
3
> (soma-primeiro-e-ultimo 1 2 3 4 5) ; (chamada com 5 argumentos)
6
> (soma-primeiro-e-ultimo 1) ; (chamada com 1 argumento)
erro. Incorrecto numero de argumentos
> (soma-primeiro-e-ultimo) ; (chamada com zero argumentos)
erro. Incorrecto numero de argumentos
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

ANEXOS



Na definição deste procedimento, o identificador argumentos é tratado como uma lista, de acordo com o que foi dito.

```
(define soma-primeiro-e-ultimo
  (lambda argumentos
    (let ((comprimento (length argumentos)))
      (if(< comprimento 2)
          (display "erro. Incorrecto numero de argumentos")
          (+ (car argumentos) ; soma o primeiro argumento ...
              (car (reverse argumentos))))))) ; e o último.
```

Este procedimento começa por verificar se o comprimento da lista argumentos é inferior a 2, o que corresponde à situação de erro. No entanto, se o comprimento for igual a ou maior que 2, então este devolve a soma do primeiro e último elementos da lista.



Teste o procedimento soma-primeiro-e-ultimo.

Desenvolva e teste uma solução alternativa para o procedimento soma-primeiro-e-ultimo.

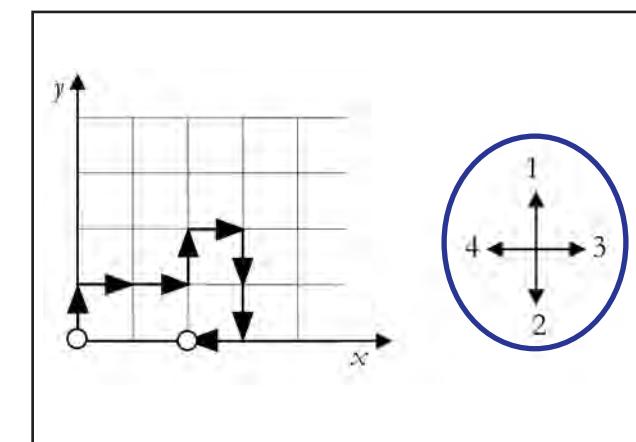
Pista: explore a utilização do procedimento primitivo list-ref.

### Exercício 1

Um dispositivo robotizado desloca-se sobre uma grelha, obedecendo a uma sequência de comandos do tipo: 1, 3, 3, 1, 3, 2, 2, 4, como se pode ver na figura.

Num sistema de coordenadas xy, foi utilizada a seguinte convenção:

Comando	Efeito
1	a coordenada y aumenta 1 unidade
2	a coordenada y diminui 1 unidade
3	a coordenada x aumenta 1 unidade
4	a coordenada x diminui 1 unidade



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



A cada comando corresponde um passo unitário e, no exemplo indicado no início do enunciado, a distância entre os pontos de partida e chegada será de 2 unidades.

Escreva em Scheme o procedimento `distancia-entre-partida-chegada`, que calcula a distância entre o ponto de partida, suposto na origem dos eixos, e o ponto de chegada do dispositivo robotizado. O percurso é especificado por uma lista de comandos.

Exemplos:

```
> (distancia-entre-partida-chegada 1)
1
> (distancia-entre-partida-chegada 1 3 2)
1
> (distancia-entre-partida-chegada 1 3 3 1 3 2 2)
3
> (distancia-entre-partida-chegada 1 3 3 1 3 2 2 4)
2
> (distancia-entre-partida-chegada 1 3 3 1 3 2 2 5)
percurso-ambiguo
>
```



Verifique que o número de argumentos não é fixo, pois depende do número de passos dados em cada percurso.  
Qual será o caso base deste procedimento?



A distância entre dois pontos, com as coordenadas  $x_1, y_1$  e  $x_2, y_2$ , determina-se a partir de

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

Valerá a pena escrever um procedimento auxiliar para o cálculo da distância entre dois pontos?



Desenvolva e teste o procedimento `distancia-entre-partida-chegada`.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



## Os procedimentos em Scheme são objectos de 1<sup>a</sup> classe

Normalmente, a classificação de objectos de 1<sup>a</sup> classe está apenas reservada aos números, booleanos, pares e símbolos, entidades que podem ser processadas e, por isso, reconhecidas como objectos passivos, contrariamente ao que sucede com os procedimentos que são objectos activos devido às actividades que lhes estão associadas.

As propriedades que caracterizam os objectos de 1<sup>a</sup> classe são a seguir indicadas, cada uma das acompanhada por um exemplo referente aos números:

- |   |                |
|---|----------------|
| 1- Ligam-se a identificadores                     | (define x 35)  |
| 2- São argumentos de procedimentos                | (sqrt 5)       |
| 3- São elementos de estruturas compostas de dados | (list 1 2 3)   |
| 4- São valores de retorno de procedimentos        | (+ 3 (sqrt 5)) |

A propriedade 1- dos objectos de 1<sup>a</sup> classe tem sido amplamente demonstrada para os procedimentos. Estes, através das expressões `lambda`, ligam-se aos identificadores escolhidos para nome dos procedimentos.

Quanto às propriedades 2- e 3-, serão verificadas no decorrer do presente módulo, enquanto a propriedade 4- será considerada no módulo seguinte.

## Procedimentos como argumentos de outros procedimentos

Os procedimentos de 1<sup>a</sup> ordem não aceitam outros procedimentos como argumentos. A partir de agora, também serão considerados os procedimentos que aceitam outros procedimentos como argumentos e que, por esse facto, se designam por procedimentos de ordem mais elevada. Estes procedimentos estão normalmente associados a um certo padrão de computação, como se verá. Como exemplos, apontam-se alguns procedimentos primitivos do Scheme, `map`, `for-each` e `apply`, e outros desenvolvidos para a resolução de situações que irá encontrar.

# S C H E M E

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

A chamada do procedimento primitivo `map` apresenta a forma

```
(map proc lista1 lista2 ...)
```

e devolve a lista cujo primeiro elemento é obtido aplicando o procedimento `proc` ao primeiro elemento de todas as listas que são argumentos, o segundo elemento da lista é obtido aplicando `proc` ao segundo elemento de todas as listas, e assim sucessivamente até ao último elemento das listas, devendo todas as listas ter o mesmo comprimento.

```
> (map max '(1 2 3 4) '(4 3 2 1) '(2 1 30 3))  
(4 3 30 4)
```

O procedimento primitivo `max` aceita um ou mais argumentos, no caso desta chamada, 3. O resultado obtido corresponde à lista constituída pelos valores máximos das 3 listas, obtida através da comparação entre o primeiro elemento de cada uma das três listas, para determinar qual deles era o maior, e também da comparação entre o segundo elemento das mesmas listas, também para determinar qual o maior, e assim sucessivamente até ao último elemento.



O que acontecerá se as listas que aparecem como argumentos não tiverem o mesmo comprimento? Por exemplo,

```
> (map max '(1 2 3 4) '(4 3 2 1) '(2 1))
```

Perante esta situação, nem todas as implementações do Scheme respondem da mesma maneira... Logo que possível, verifique como responde o DrScheme que utiliza.

Nos exemplos que se seguem, o procedimento `proc` aceita apenas um argumento e, por este facto, é passada uma única lista em cada chamada.

```
> (map add1 (list 1 2 3 4))  
(2 3 4 5)  
> (map (lambda (x) (* x x)) (list 1 2 3))  
(1 4 9)
```





Indique as respostas dadas pelo Scheme nas situações que se seguem.

```
> (map (lambda (x y) (if (odd? x) x y))
         (list 1 2 3) (list 11 12 13))
???
> (map (lambda (x y) (if (odd? x) x y))
         (list 1 2 3) (list 11 12 13) (list 21 22 23))
???
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Em certas situações, a utilização de `map` não é a solução mais conveniente, por exemplo, quando não espera uma lista como resultado, mas lhe interessam unicamente os efeitos colaterais de um procedimento. É o que acontece quando se aplica `display` aos vários elementos de uma lista. Não espera uma lista como resultado, mas os efeitos de `display` no ecrã. Neste caso, deve utilizar o procedimento primitivo `for-each`.

A chamada de `for-each` apresenta a forma

```
(for-each proc lista1 lista2 ...)
```

e, apenas para efeitos laterais, aplica `proc` ao primeiro elemento de todas as listas que são argumentos, depois ao segundo elemento de todas as listas, e assim sucessivamente até ao último elemento das listas, que deverão ter todas o mesmo comprimento. Também neste caso, como acontecia com `map`, o procedimento `proc` deverá aceitar um número de argumentos igual ao número de listas que surgem em cada chamada.

```
> (for-each display (list "valor x= " 35 " e valor y= " (+ 10 6)))
valor x= 35 e valor y= 16
```

### Exemplo 2

Para ilustrar a funcionalidade de `for-each`, vai definir um procedimento que escreve uma linha de texto e responde como se indica.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

R E 1 2

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (escreve-linha "x = " 5 "; y = " 6)      (chamada com 4 argumentos)
x = 5; y = 6
> (escreve-linha "Olá'. Estas bom?")
Olá'. Estas bom?
```

No primeiro destes dois exemplos, quatro chamadas a `display` e uma a `newline` seriam necessárias para produzir o efeito apresentado. Com o procedimento `escreve-linha`, este tipo de situação simplifica-se.

```
(define escreve-linha
  (lambda args
    (for-each display args)
    (newline)))
```

Sem o procedimento primitivo `for-each`, a solução poderia tomar outro aspecto.

```
(define escreve-linha-sem-for-each
  (lambda args
    (letrec ((aux
              (lambda (lis)
                (if (null? lis)
                    (newline)
                    (begin
                      (display (car lis))
                      (aux (cdr lis)))))))
      (aux args))))
```



Teste os procedimentos `escreve-linha` e `escreve-linha-sem-for-each`.

A chamada do procedimento primitivo `apply` apresenta a forma

```
(apply proc lista)
```

e devolve um resultado equivalente a uma chamada de `proc` em que os argumentos seriam todos os elementos de `lista`.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Convém entender sem qualquer dúvida qual é a diferença entre dizer que os argumentos são todos os elementos de uma lista ou que o argumento é a própria lista.  
Os exemplos ajudam a esclarecer esta questão.

```
> (apply + (list 1 2 3))          (equivalente a (+ 1 2 3))
6
> (apply add1 (list 1 2 3))       (equivalente a (add1 1 2 3))
add1: expects 1 argument, given 3: 1 2 3
> (apply add1 (list 1))           (equivalente a (add1 1))
2
```



Justifique o que resulta das três chamadas que se seguem.

```
> (apply max '(1 2 3 9 4))
9
> (max 1 2 3 9 4)
9
> (max '(1 2))
max: expects argument of type <real number>; given (1 2)
```



E agora justifique o que resulta das duas chamadas que se seguem.

```
> (map max '(1 2 3 9 4))
(1 2 3 9 4)
> (for-each max '(1 2 3 9 4))
>
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Experimente os procedimentos primitivos de ordem mais elevada `map`, `for-each` e `apply`.

Cada um destes três procedimentos primitivos, `map`, `for-each` e `apply`, evidencia um certo padrão de computação. Por exemplo, `apply` chama um procedimento (o primeiro argumento de `apply`), tendo como argumentos os elementos de uma lista (o segundo argumento de `apply`). Trata-se de uma computação genérica, independente de uma função específica. Sem `apply`, seria necessário um procedimento diferente para cada função, apesar de todos eles apresentarem um padrão semelhante. Estas situações ocorrem com frequência e o programador deverá decidir, perante elas, se é preferível uma solução específica ou, alternativamente, desenvolver uma solução genérica, através de um procedimento de ordem mais elevada.

### Exemplo 3

O procedimento `max-fun` determina o máximo de uma função num domínio definido por dois limites. Pretende-se uma solução geral, em vez de resolver o problema para uma função específica. O procedimento `max-fun` tem como parâmetros uma função de uma variável, dois valores que definem os limites inferior e superior de um domínio em que a função é contínua e um valor incremental, com o qual se definem os pequenos saltos constantes, realizados para percorrer o citado domínio da função, desde o limite inferior até ao limite superior.

```
> (max-fun cos 0 (* 0.25 pi) 0.01)
1.0
> (max-fun sin 0 (* 0.25 pi) 0.01)
0.7071067811865475
> (max-fun (lambda (x) (- (* x x) x)) 0 3 0.01)
6.0
```



A constante `pi` foi considerada igual a 3.141592653589793 e reembra-se que, em Scheme, as funções trigonométricas utilizam os ângulos em radianos.

0.25\*pi radianos corresponde a quantos graus? Uma ajuda: pi radianos corresponde a 180°.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Na definição de max-fun é utilizado aux, um procedimento interno recursivo que recebe, garantidamente, os limites que definem o domínio de estudo da função, em ordem crescente. Este cuidado é tido em conta, já que max-fun poderá receber aqueles limites em ordem decrescente. Por seu turno, aux utiliza o procedimento primitivo max.

```
(define max-fun
  (lambda (f lim-inf lim-sup dx)
    (letrec ((aux
              (lambda (inf sup)
                (cond
                  ((> inf sup) (f sup))
                  (else
                    (max (f inf) (aux (+ inf dx) sup)))))))

      (if (> lim-inf lim-sup)
          (aux lim-sup lim-inf)
          (aux lim-inf lim-sup)))))
```

Note, mais uma vez, que o procedimento max-fun não fica limitado a uma única função do tipo  $y = f(x)$  e que resolve um certo padrão de computação, ou seja, o máximo de  $y = f(x)$  num intervalo definido por dois limites. max-fun é um exemplo de um procedimento de ordem mais elevada.



Experimente o procedimento max-fun.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Exercício 2

Reescreva o procedimento max-fun, mas agora sem utilizar o procedimento max.



Desenvolva e teste uma nova versão do procedimento max-fun.



**Na versão inicial, max-fun gera processos recursivos que são  $O(n)$  em tempo e espaço.  
E na nova versão?**

Os procedimentos soma-inteiros e soma-quadrados apresentam padrões de computação muito semelhantes. Têm como parâmetros dois limites e somam, entre esses dois limites, os inteiros e os quadrados desses inteiros, respectivamente.

```
(define soma-inteiros
  (lambda (a b)
    (if (> a b)
        0
        (+ a
            (soma-inteiros (add1 a) b)))))

(define soma-quadrados
  (lambda (a b)
    (if (> a b)
        0
        (+ (* a a)
            (soma-quadrados (add1 a) b)))))
```

Estes dois procedimentos podem ser vistos como um somatório dos valores de uma função, calculados entre dois limites, com passo unitário. No primeiro caso, a função seria  $f(x) = x$ , no segundo,  $f(x) = x * x$ .



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



Se assim for, os dois procedimentos podem ser considerados casos particulares de um procedimento mais geral, somatorio, com 4 parâmetros: a função que calcula os termos a somar, os dois limites que já apareciam em soma-inteiros e soma-quadrados e a função que define o passo.

```
(define somatorio
  (lambda (fun inf sup passo)
    (if (> inf sup)
        0
        (+ (fun inf)
            (somatorio fun (passo inf) sup passo)))))
```

Baseados em somatorio, aqueles procedimentos podem agora tomar a seguinte forma:

```
(define soma-inteiros-baseado-em-somatorio
  (lambda (inf sup)
    (somatorio (lambda(x) x) inf sup add1)))

(define soma-quadrados-baseado-em-somatorio
  (lambda (inf sup)
    (somatorio (lambda(x) (* x x)) inf sup add1)))
```

O procedimento somatorio é também um procedimento de ordem mais elevada que vai ser muito utilizado, nessa qualidade, em exercícios e exemplos que se seguem, para resolver um certo padrão de computação.



Experimente o procedimento somatorio e os que nele se basearam.



Analise a solução alternativa que se segue para somatorio e apresente razões para justificar se se trata ou não de uma solução melhor que a anteriormente apresentada.

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
(define somatorio
  (lambda (fun inf sup passo)
    (letrec ((aux
              (lambda (i total)
                (if (> i sup)
                    total
                    (aux (passo i) (+ total (fun i)))))))
      (if (> inf sup)
          0
          (aux (passo inf) (fun inf)))))))
```

## Exemplo 4

A soma dos n primeiros ímpares é dada por  $1 + 3 + 5 + \dots + (2 * n - 1)$ . Utilizando somatorio, escreva em Scheme o procedimento soma-ímpares-baseado-em-somatorio, com os parâmetros inf e sup, que definem, respectivamente, os limites inferior e superior dos ímpares a somar.

```
> (soma-ímpares-baseado-em-somatorio 3 7)
15
> (soma-ímpares-baseado-em-somatorio 3 1)
0
```

Pista:

- Função que calcula os termos a somar,  $f(x) = x$ ;
- Função que define o passo,  $f(x) = x + 2$ .

```
(define soma-ímpares-baseado-em-somatorio
  (lambda (inf sup)
    (somatorio (lambda (x) x) inf sup (lambda (x) (+ x 2)))))
```



Teste o procedimento soma-ímpares-baseado-em-somatorio.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



No procedimento `soma-impares-baseado-em-somatorio`, o que é que acontece se o limite `inf` fornecido for maior que o limite `sup`?  
E o que acontece se o limite `inf` não for ímpar?

### Exercício 3

O procedimento `pi-aprox-somatorio`, baseado em `somatorio`, inspira-se na fórmula:  $\frac{\pi}{8} = \frac{1}{1*3} + \frac{1}{5*7} + \frac{1}{9*11} + \dots$

Escreva em Scheme o procedimento `pi-aprox-somatorio` sabendo que tem apenas um parâmetro, `n`, e que devolve o valor de `pi`, aproximado aos primeiros `n` termos da fórmula.

Pista:

- Função para cálculo dos termos a somar,  $f(x) = \frac{1}{x * (x+2)}$  ;
- Função que define o passo,  $f(x) = x + 4$ .



Desenvolva e teste o procedimento `pi-aprox-somatorio`.

### Exercício 4

Teve oportunidade de utilizar o procedimento `somatorio` em várias situações. Defina algo equivalente, mas agora para o caso do produto, sendo o respectivo procedimento designado por `produtorio`.



Desenvolva e teste o procedimento `produtorio`.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



### Exercício 5

O procedimento pi-aprox-produtorio, baseado em produtorio, inspira-se na fórmula:

$$\frac{\pi}{4} = \frac{2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdot 8 \dots}{3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdot 7 \dots}$$

Escreva pi-aprox-produtorio em Scheme, sabendo que tem apenas o parâmetro n e que devolve o valor de pi, aproximado aos primeiros n termos da fórmula.

Pista

Comece por identificar

- A função que calcula os termos;
- A função que define o passo.



Desenvolva e teste o procedimento pi-aprox-produtorio.

### Procedimentos como elementos de estruturas de dados

Para ilustrar que os procedimentos podem também ser elementos incorporados em estruturas de dados, o que corresponde à propriedade 3- dos objectos de 1<sup>a</sup> classe, começemos por definir uma lista de procedimentos.

```
> (define lista-de-proc (list (lambda(x) (* x x x)) expt car))
```



Identifique os 3 elementos da lista designada por lista-de-proc.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Nos exemplos que se seguem, fica bem claro que os procedimentos podem ser tratados como elementos de uma lista.



**Justifique os resultados obtidos no diálogo que se segue.**

```
> ((car lista-de-proc) 5)
125
> ((cadr lista-de-proc) 2 5)
32
> ((caddr lista-de-proc) lista-de-proc)
#<procedure>
> ((caddr lista-de-proc) '((1 2) (3 4)))
(1 2)
> (((caddr lista-de-proc) lista-de-proc) 5)
125
```



Experimente o diálogo indicado.  
Sugere-se que introduza algumas variantes.



**No final deste módulo, indique a propriedade dos objectos de 1<sup>a</sup> classe que ainda não foi verificada com os procedimentos.**

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

## Módulo 4.2 - Um procedimento pode devolver outro procedimento

Já associou muitas vezes procedimentos a identificadores (identificadores que passaram a ser o nome desses procedimentos), já utilizou ou escreveu procedimentos que aceitavam outros procedimentos como argumentos e certamente já verificou que os procedimentos podem fazer parte de uma estrutura de dados. Tudo isto foi analisado no módulo anterior. Agora, para que os procedimentos possam ser considerados como objectos de 1<sup>a</sup> classe, faltará mostrar que podem devolver, como valor de retorno, outros procedimentos. A comprovação deste facto é o objectivo deste módulo.

### Palavras-Chave

Objectos de 1<sup>a</sup> classe, procedimento como valor de retorno de outro procedimento, recta que melhor se aproxima de um conjunto de pontos, derivada de uma função  $f(x)$ , raízes de uma função  $f(x)$ , método de *Newton-Raphson*, ponto fixo de uma função.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

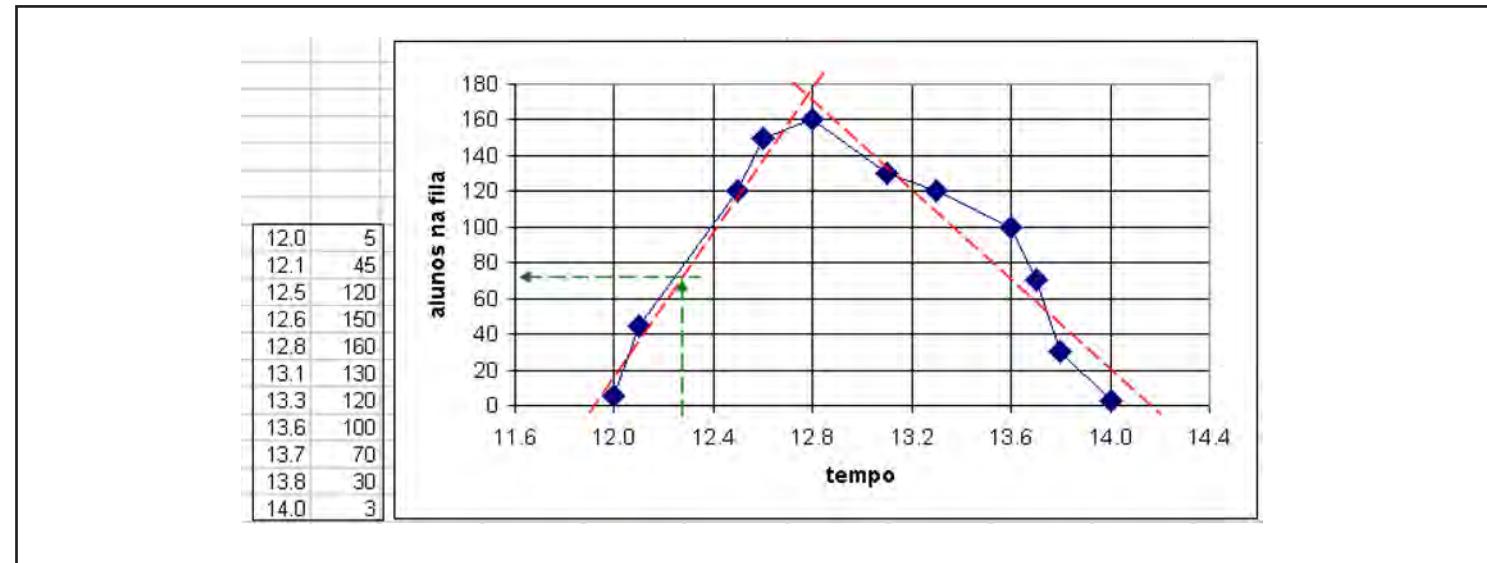
### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Procedimento como valor de retorno de outro procedimento

Para introduzir este tema, procedimento como valor de retorno, vai começar por imaginar a situação dos estudantes que, por volta as 12:00 horas, se aproximam de uma cantina. Cria-se uma fila de espera que aumenta até atingir uma dimensão máxima e que depois começará a diminuir, até desaparecer completamente. Alguém mediou o comprimento da fila de espera, algumas vezes, entre as 12:00 e as 14:00 horas. Supõe-se que, fora deste período, a fila de espera praticamente não existe.



Na figura, são indicados os resultados das medições efectuadas, sob a forma de uma tabela (à esquerda) e através de um gráfico a unir vários pontos (na figura, a azul).

Pretende-se encontrar as rectas (na figura, a vermelho) que melhor se aproximem das medições, uma na zona em que a fila aumenta e a outra na zona em que a fila diminui. Com estas rectas será possível determinar o número aproximado de estudantes na fila de espera, para um determinado instante, fazendo um percurso idêntico ao indicado pelas setas (na figura, a verde).

Para determinar estas rectas, pode começar por encontrar o valor máximo da fila de espera e considerar que ele representa o ponto a partir do qual a fila começa a diminuir. Depois disso, o problema reside em, dado um conjunto de pontos que associam o tempo e o número de estudantes, encontrar a recta que melhor se aproxima desses pontos.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

R E 1 2

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Os pontos referidos poderão ser representados por uma lista de pares, constituindo a lista fila-cantina.

```
(define fila-cantina (list (cons 12.0 5) (cons 12.1 45) (cons 12.5 120)
                             (cons 12.6 150) (cons 12.8 160) (cons 12.8 160)
                             (cons 13.1 130) (cons 13.3 120) (cons 13.6 100)
                             (cons 13.7 70) (cons 13.8 30) (cons 14.0 3)))
```

Manualmente, pode decompor esta lista em duas listas com um ponto em comum, o ponto em que a fila atinge o comprimento máximo.

```
(define fila-aumenta (list (cons 12.0 5) (cons 12.1 45) (cons 12.5 120)
                             (cons 12.6 150) (cons 12.8 160)))
(define fila-diminui (list (cons 12.8 160) (cons 13.1 130) (cons 13.3 120)
                             (cons 13.6 100) (cons 13.7 70) (cons 13.8 30)
                             (cons 14.0 3)))
```

Agora, o que interessava era escrever um procedimento em Scheme, a designar por melhor-aproximacao, que devolvesse não um número ou uma lista de números, como tem acontecido até aqui, mas sim a equação de uma recta,  $y = mx + b$ , em que  $m$  representa a inclinação da recta e  $b$  o ponto em que a recta intersecta o eixo dos  $yy$ . É importante relembrar que, em Scheme, esta equação pode exprimir-se através de um procedimento, como o que agora se indica.

```
(lambda (x)
  (+ (* m x) b))
```

Ou seja, o procedimento melhor-aproximacao, tendo como parâmetro uma lista de pontos, devolveria um procedimento com o parâmetro  $x$ .

Assim, não será de admirar que a última expressão do procedimento melhor-aproximacao seja, exactamente, uma expressão lambda. A sua estrutura é seguidamente indicada.

```
(define melhor-aproximacao
  (lambda (lista-pontos)
    ;
    ; cálculo de m e b
    ; tendo por base a lista em parâmetro
    ...
    ;
    (lambda (x)
      (+ (* m x) b))))
```





Será capaz de completar o procedimento **melhor-approximacao** ou ainda vai precisar de uma ajuda? Se a ajuda, eventualmente necessária, estiver relacionada com o cálculo de  $m$  e  $b$  a partir de um conjunto de pontos, irá encontrá-la um pouco mais à frente, num exemplo cujo tema é: a recta que se aproxima de um conjunto de pontos.



Consolide a situação em que um procedimento apresenta como valor de retorno um outro procedimento. Sugere-se que se criem equações para outras rectas...

```

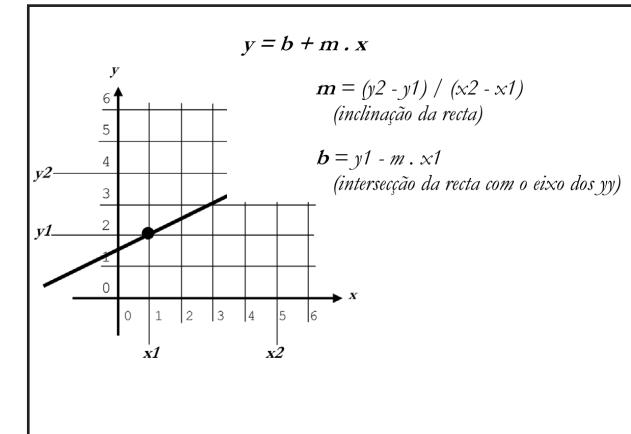
(define m
  (lambda (x1 y1 x2 y2)
    (/ (- y2 y1)
        (- x2 x1)))))

(define b
  (lambda (x1 y1 m)
    (- y1 (* m x1)))))

(define equacao-recta
  (lambda (m b)
  ;
  (lambda (x)
    (+ (* m x) b)))))

> (define m-exemplo (m 1.0 2.0 5.0 4.0))
> m-exemplo
0.5
> (define b-exemplo (b 1.0 2.0 m-exemplo))
> b-exemplo
1.5
> (define recta-ex (equacao-recta m-exemplo b-exemplo))
> (recta-ex 0)
1.5
> (recta-ex 1)
2.0
> (recta-ex 5)
4.0
> (recta-ex 3.5)
3.25
> (recta-ex 4)
3.5
>

```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

ANEXOS

### Exemplo 1- a recta que se aproxima de um conjunto de pontos

Agora, seria necessário consultar um livro de Matemática para encontrar alguma maneira de, dado um conjunto de pontos, determinar o  $m$  e  $b$  da recta que melhor se aproxime desses pontos ou, em alternativa, aceitar a informação que se segue.



Sendo  $n$  o número de pontos e  $P_i$  um ponto genérico de coordenadas  $x_i$  e  $y_i$ , então

$$m = \frac{n * f - c * d}{n * e - c * c} \quad \text{e} \quad b = \frac{d * e - c * f}{n * e - c * c}$$

em que

$$c = \sum_{i=1}^n x_i, \quad d = \sum_{i=1}^n y_i, \quad e = \sum_{i=1}^n x_i * x_i \quad \text{e} \quad f = \sum_{i=1}^n x_i * y_i$$

Depois disto, e relembrando a funcionalidade de `map` e `apply`, rapidamente encontrará uma solução para o procedimento `melhor-aproximacao`.

```
(define melhor-aproximacao
  (lambda (lista-pontos) ; a partir de uma lista de pontos...
    (let* ((Pontos-x (map car lista-pontos)) ; cria uma lista com
           ; as coordenadas xx
           (Pontos-y ... ... completar) ; cria uma lista com
           ; as coordenadas yy
           (n-pontos (length lista-pontos)) ; determina o número
           ; de pontos
           ;;
           (c (apply + Pontos-x)) ; calcula c
           (d (apply + Pontos-y)) ; calcula d
           (e (apply + (map (lambda (t) (* t t)) Pontos-x))) ; calcula e
           (f (apply ... ... completar) ; calcula f
               ; calcula divisor de m e b
               (divisor ... ... completar)
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
; calcula a inclinação da recta
(m (/ (- (* n-pontos f)
           (* c d))
           divisor))

; calcula a intersecção da recta com o eixo dos yy
(b ... ... completar

; ---- devolve a equação da recta ----
;
(lambda (x)
  (+ (* m x) b)))))
```

Com duas chamadas ao procedimento melhor-aproximação, resultam as equações das rectas que aproximam os pontos em que a fila de espera aumenta, equacao-fila-aumenta, e os pontos em que a fila diminui, equacao-fila-diminui.

```
> (define equacao-fila-aumenta (melhor-aproximacao fila-aumenta))
> (define equacao-fila-diminui (melhor-aproximacao fila-diminui))
```

Com estas equações é possível fazer algumas experiências sobre a evolução aproximada da fila de espera, das 12:00 às 14:00.

```
> (equacao-fila-aumenta 12.0)
17.30434782608063
> (equacao-fila-aumenta 12.8)
174.69565217389436
> (equacao-fila-diminui 12.8)
172.5625000000125
> (equacao-fila-diminui 14.0)
20.663563829771647
> (equacao-fila-aumenta 12.3)
76.32608695651106
```



Complete e teste o procedimento melhor-aproximacao.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

### Exercício 1

Com base no procedimento `melhor-aproximacao` e nas listas `fila-aumenta` e `fila-diminui`, apresentados anteriormente, escreva um programa em Scheme que forneça ao utilizador o comprimento aproximado da fila de espera da cantina, em instantes à escolha, situados no período entre as 12:00 e as 14:00 horas.



Desenvolva e teste o programa pedido.

### Exercício 2

Com base no procedimento `melhor-aproximacao`, exposto anteriormente, escreva um programa que forneça ao utilizador o comprimento aproximado da fila de espera da cantina, em instantes à escolha. O programa recebe como argumento uma lista com todos os pares que representam as medições na fila de espera. Os pares da lista serão apresentados em ordem crescente em relação tempo.



Se as medições fossem

12.00 45, 12.20 87, 13.00 387, 12.75 310, 13.50 20,  
13.30 150, 13.10 275, 13.95 5, 13.35 120

indique qual a lista que o programa pedido esperaria como argumento.



Desenvolva e teste o programa pedido.



# S C H E M E

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

Um outro exemplo, já utilizado noutras publicações para ilustrar o tema "procedimento como valor de retorno", vai ser agora apresentado.

Quando coloca a questão qual é a derivada de  $x^3$ ? a resposta correcta deverá ser  $3x^2$ .

Ou seja, a derivada de uma função de  $x$  é ainda uma função de  $x$ . E se a derivada da função for calculada através de um procedimento, então não restam dúvidas de que este procedimento terá de devolver uma função, sob a forma de um procedimento.

No desenvolvimento do procedimento derivada utiliza-se a seguinte definição, que supõe  $dx$  um valor muito

$$\text{pequeno: } Df(x) = \frac{f(x + dx) - f(x)}{dx}$$

```
(define derivada
  (lambda (fun dx)
    ;
    (lambda (x)
      (/ (- (fun (+ x dx)) (fun x))
          dx))))
```

Neste exemplo, também aparecem dois `lambda` seguidos, significando que o procedimento `derivada`, com dois parâmetros `fun` e `dx`, devolve não um número, um booleano, um símbolo ou um par, mas sim um outro procedimento, neste caso, um procedimento com o parâmetro `x`.



**Verifique, nas chamadas que se seguem, dois níveis de parêntesis.  
Do nível interno, relacionado com uma chamada a derivada, portanto com 2 argumentos, o que é que resulta?  
O nível externo é uma chamada ao procedimento que resulta da chamada do nível interno, exibindo, portanto, apenas um argumento.**

```
> ((derivada cos .001) 0)
-0.00049999958325503
> ((derivada (lambda(x) (* 2 x x)) .001) 2)
8.0019999999994
> ((derivada (lambda(x) (* 2 x x)) .001) 3)
12.001999999999
```



<b>INTRODUÇÃO</b>
<b>1 - O ESSENCIAL DO SCHEME</b>
<b>2 - RECURSIVIDADE</b>
<b>3 - ABSTRACÇÃO DE DADOS</b>
<b>4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE</b>
RE 1 2
<b>5 - ABSTRACÇÕES COM DADOS MUTÁVEIS</b>
<b>6 - SCHEME E OUTRAS TECNOLOGIAS</b>
<b>7-EXERCÍCIOS E PROJECTOS</b>
<b>ANEXOS</b>



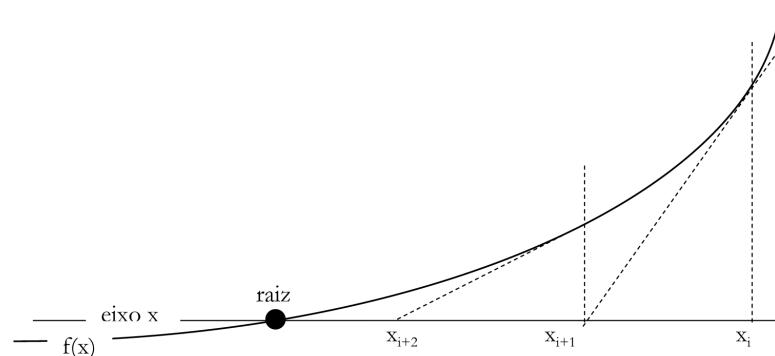
Experimente o procedimento derivada.

Sugere-se a utilização do procedimento com funções que conheça.

### Exemplo 2 - as raízes de uma função

Segundo o método de *Newton-Raphson*, para encontrar as raízes de uma função diferenciável  $f(x)$ , se  $x_i$  é uma aproximação para uma raiz de  $f(x)$ , uma aproximação melhor será  $x_{i+1}$ , em que  $x_{i+1} = x_i - \frac{f(x_i)}{Df(x_i)}$

Na figura, é esboçado o método de *Newton-Raphson*.



Desenha-se a tangente a  $f(x)$  em  $x_i$  e verifica-se que a tangente intersecta o eixo dos  $xx$  em  $x_{i+1}$ , mais perto da raiz que  $x_i$ . Se, todavia,  $x_{i+1}$  não está suficientemente perto da raiz, desenha-se a tangente a  $f(x)$ , mas agora em  $x_{i+1}$  como se fez para  $x_i$ .

A ideia a explorar é tentar saber se uma aproximação já é suficientemente boa para ser uma raiz de  $f(x)$  e, se não for, procurar outra melhor. Assim, o procedimento *newton-raphson* baseia-se em três procedimentos internos, respectivamente *boa-aprox?*, predicado que indica se a aproximação já é boa, *melhora* que, a partir de uma aproximação, devolve uma aproximação melhor e *aux*, um procedimento recursivo que vai requerendo uma melhor aproximação até que esta seja suficientemente boa.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Internamente, ainda é associado ao identificador `derivada-fun` o resultado de uma chamada ao procedimento `derivada`, que devolve, como já se disse, a derivada de uma função.

```
(define newton-raphson
  (lambda (func aprox)
    (let* ((derivada-fun (derivada func 0.001))

          (melhora
            (lambda (ap) ; ap é uma aproximação a uma raiz
              (- ap (/ (func ap) ; de f, que se pretende melhorar...
                    (derivada-fun ap)))))

          (boa-aprox? ; para verificar se a aproximação já
            (lambda (ap) ; é suficientemente boa...
              (< (abs (func ap)) .00001)))))

      (letrec ((aux
                (lambda (ap)
                  (if (boa-aprox? ap)
                      ap
                      (aux (melhora ap)))))

              ;
              (aux aprox)))))

    > (newton-raphson (lambda (x) (+ (* x x) (* -5 x) 6)) 1)
2.0000000107312625
```



O procedimento `newton-raphson`, ao receber o valor 1 como aproximação à raiz de  $f(x) = x^2 - 5x + 6$ , devolve 2.0000000107312625, valor muito próximo da raiz 2.  
O que será necessário alterar para conseguir resultados ainda mais próximos das raízes?



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



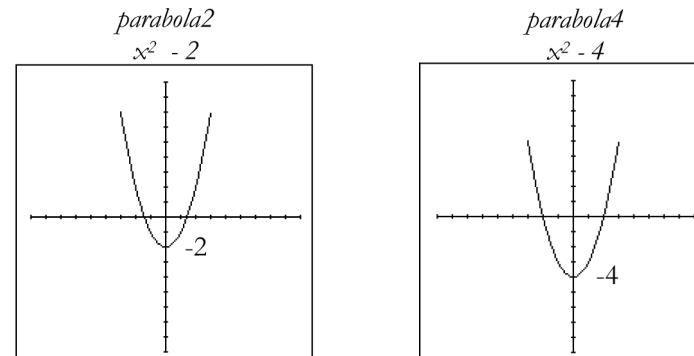
Observe agora o que acontece quando o procedimento newton-raphson recebe o mesmo valor, 1, como aproximação à raiz de  $f(x) = x^3 - 5x + 6$ .

```
> (newton-raphson (lambda (x) (+ (* x x x) (* -5 x) 6)) 1)
-2.689095320750167
```



Teste o procedimento newton-raphson.

Seguem-se algumas experiências com o procedimento newton-raphson aplicado a equações de parábolas.



```
> (define parabola2 (lambda (x) (- (* x x) 2)))
> (define parabola4 (lambda (x) (- (* x x) 4)))
> (newton-raphson parabola2 5)
1.4142136830753582
> (newton-raphson parabola2 -5)
-1.4142134968220743
> (newton-raphson parabola4 5)
2.0000000016164567
> (newton-raphson parabola4 -50)
-1.9999999708113276
> (newton-raphson parabola4 -5)
-1.9999999990140929
```

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

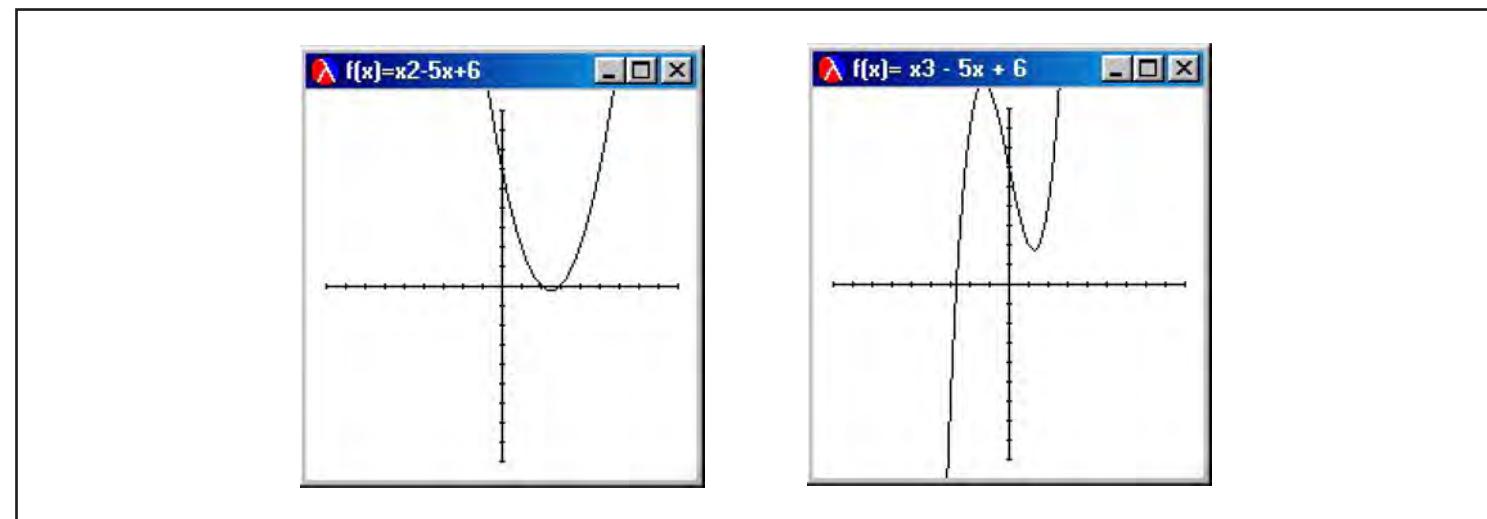
## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Nos exemplos das parábolas, foram calculadas duas raízes, para cada uma das funções, pois sabia-se de antemão, pelos desenhos das mesmas, que elas passavam duas vezes pelo eixo dos  $x$ . A presença das figuras foi fundamental para dar a conhecer não só a existência das duas raízes, mas também os respectivos valores aproximados. Foi assim possível escolher as aproximações às raízes que viriam a ser utilizadas nas chamadas do procedimento newton-rapson.

Já para as funções  $f(x) = x^2 - 5x + 6$  e  $f(x) = x^3 - 5x + 6$ , apenas foi encontrada uma raiz para cada uma delas. Será mesmo assim?



Pela análise das figuras conclui-se que, em relação à primeira das funções, uma raiz ficou por encontrar, pois existem duas, uma nas proximidades do valor 2, que foi encontrada, e outra perto do valor 3. De facto, utilizando, por exemplo, a aproximação 5, em vez da de 1, obtém-se um valor muito próximo da outra raiz.

```
> (newton-raphson (lambda (x) (+ (* x x) (* -5 x) 6)) 5)
3.0000040484485284
```



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

R E 1 2

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

O valor de  $x$  para o qual  $x = f(x)$  é o chamado ponto-fixo de  $f(x)$ . Como pode ver, o ponto-fixo de uma função é o ponto em que essa função e o seu argumento exibem o mesmo valor.

No caso da função  $f(x) = x^2 - 5x + 6$ , o ponto-fixo  $x$  é definido por  $x = x^2 - 5x + 6$  e pode ser determinado fazendo  $x - (x^2 - 5x + 6) = 0$ , o que sugere a utilização do procedimento newton-raphson.

```
> (newton-raphson (lambda (x) (- x (+ (* x x) (* -5 x) 6))) 1)
1.267949215205654

(define ponto-fixo
  (lambda (f)
    (newton-raphson (lambda (x) (- x (f x))) 1)))

> (ponto-fixo (lambda (x) (+ (* x x) (* -5 x) 6)))
1.267949215205654
                                                 (resultado esperado...)
> (ponto-fixo (lambda (x) (* x x)))
1
```

O procedimento ponto-fixo é um procedimento de ordem mais elevada, desenvolvido com base no procedimento newton-raphson, também ele um procedimento de ordem mais elevada.



Teste o procedimento ponto-fixo.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

## Resumo dos módulos 5.1 a 5.5

Resumo dos assuntos abordados nos módulos

Módulo 5.1 - **Não só cria, o Scheme também modifica...**

Módulo 5.2 - **Filas, Pilhas e Tabelas são estruturas mutáveis de grande utilidade**

Módulo 5.3 - **Vectores são dados mutáveis do Scheme**

Módulo 5.4 - **Cadeias de caracteres também são dados mutáveis do Scheme**

Módulo 5.5 - **Pesquisas, Ordenações e Árvores binárias**



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Resumo dos assuntos abordados nos módulos 5.1 e 5.5

Neste conjunto de módulos, o objectivo principal é mostrar que o Scheme também permite a modificação de dados, designados por dados mutáveis, disponibilizando para tal os chamados modificadores. Neste contexto, surgem as listas mutáveis, e com elas constroem-se importantes abstracções, filas de espera, pilhas e tabelas. Também se analisam abstracções de dados mutáveis disponibilizadas pelo próprio Scheme, vectores e cadeias de caracteres. No seguimento das cadeias de caracteres, os ficheiros de texto são também considerados. Com os ficheiros, os dados produzidos numa sessão de trabalho podem ser guardados para utilização em sessões futuras. A parte final é organizada em torno das árvores binárias, dando-se um relevo especial às árvores de pesquisa binária.

As grandes diferenças entre vectores e listas mutáveis, com as quais se implementam as filas de espera, as pilhas e as tabelas, são as seguintes: as listas são estruturas de dados dinâmicas, de grande flexibilidade, pois é possível juntar-lhes ou retirar-lhes elementos. As listas têm também um acesso do tipo sequencial, pois para chegar a um dos seus elementos é necessário passar por todos os elementos que o antecedem, ao passo que os vectores são normalmente estruturas estáticas, de dimensão fixa, ainda que todos os seus elementos sejam igualmente acessíveis através do respectivo índice. As listas mutáveis aplicam-se sobretudo em situações em que a dimensão da estrutura de dados varia no tempo e o respectivo processamento se adequa a um acesso do tipo sequencial. Os vectores encontram um campo de aplicação favorável quando o acesso aos seus elementos não privilegia uns em relação a outros e não é importante ter uma estrutura com dimensão variável.

As árvores de pesquisa binária surgem como a estrutura que reúne o melhor dos dois mundos, que suporta a pesquisa binária (como o vector), mas com a facilidade de poder crescer ou diminuir que caracteriza uma estrutura dinâmica (como a lista mutável).

### Módulo 5.1

Até este ponto do nosso estudo, o Scheme não permitia modificar as entidades criadas, obrigando-nos a recorrer a artifícios do tipo: a modificação de uma entidade é simulada criando uma nova entidade com o mesmo nome da entidade que se pretendia modificar. Este tipo de solução, que tem por base um construtor, pode não ser a mais adequada e nem sempre resulta. A forma correcta de realizar este tipo de operação recorre aos chamados modificadores que o Scheme disponibiliza para modificar entidades já existentes, com o modificador `set!`, ou com os modificadores `set-car!`, `set-cdr!` e `append!`. Surge assim a lista mutável.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Módulo 5.2

As abstracções filas de espera, pilhas e tabelas são analisadas e implementadas através de listas mutáveis. Todas elas são estruturas dinâmicas, uma vez que a respectiva dimensão pode variar, pela introdução ou eliminação de elementos. As duas primeiras organizam os seus elementos em função do tempo, por exemplo, pela ordem de chegada, enquanto as últimas se constituem em registos, cada um deles caracterizado por uma chave e um valor associado, sendo o acesso a qualquer um dos elementos feito através da sua chave.

## Módulos 5.3 e 5.4

Com os modificadores que o Scheme disponibiliza (`set!`, `set-car!`, `set-cdr!` e `append!`) é possível criar abstracções de dados mutáveis de acordo com as características de cada situação. Algumas destas abstracções são definidas e implementadas, nomeadamente as filas de espera, pilhas e tabelas, mas outras são, logo à partida, disponibilizadas pelo Scheme e nelas se incluem os vectores (Módulo 5.3) e as cadeias de caracteres (Módulo 5.4).

As soluções recursivas sobre estas estruturas de dados surgem com uma forma que difere da utilizada no caso das listas, por não existir o equivalente ao `cdr`. No caso dos vectores e das cadeias de caracteres, principia-se por calcular o comprimento destes e um índice vai sendo incrementado até alcançar o valor do comprimento, situação que constitui uma das condições de terminação. No entanto, a implementação do procedimento `string-cdr`, um construtor, permite encarar a recursividade em cadeias de caracteres de uma forma semelhante à utilizada para as listas, embora de uma maneira muito mais ineficiente (cada chamada de `string-cdr` implica a criação de uma nova cadeia...)

A propósito das cadeias de caracteres e da necessidade de guardar dados de uma sessão de trabalho para outra, introduzem-se os ficheiros.

## Módulo 5.5

As operações de pesquisa e ordenação são as operações em destaque neste módulo, incidindo especialmente sobre árvores binárias e vectores. Das árvores binárias, distingue-se a de pesquisa binária pelo excelente desempenho  $O(\log_2 n)$  que oferece quando se encontra equilibrada ou balanceada.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Exercícios e exemplos

São apresentados alguns exercícios e exemplos para consolidação da matéria tratada nos módulos desta Parte. Neste sentido, recomenda-se o estudo do [Anexo A](#), no que se refere a modificadores (`set!`, `set-car!`, `set-cdr!` e `append!`), vectores, cadeias de caracteres e ficheiros.



# S C H E M E

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

### Exercício 1

Utilizando a abstracção fila de espera, escreva em Scheme um programa que simule e visualize a fila de espera de uma cantina, de acordo com a interacção que se indica, na qual se considera a seguinte codificação:

- |   |                            |   |                      |
|---|----------------------------|---|----------------------|
| e | estudante de Electrotecnia | q | estudante de Química |
| i | estudante de Informática   | m | estudante de Minas   |
| c | estudante de Civil         |   |                      |

```
> (cantina)
()
> quem-entra? e
(e)
> quem-entra? q
(e q)
> quem-entra? d
Nao pode ser!...
(e q)
> quantos-saem? 0
(e q)
> quem-entra? q
(e q q)
> quantos-saem? 2
(q)
> quem-entra? i
(q i)
> quantos-saem? 3
Nao pode ser!...
(q i)
> quem-entra? i
(q i i)
> quantos-saem? -1
Vai terminar...
```

Sabe-se que as perguntas `quem-entra?` e `quantos-saem?` são geradas aleatoriamente pelo programa de simulação, mas a probabilidade da pergunta `quem-entra?` é 3 vezes superior à pergunta `quantos-saem?`.



Desenvolva e teste o programa `cantina`.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

### Exercício 2

Na fila de espera modelada como se fosse um par, as operações de saída (do início) e de entrada (no fim) apresentavam-se com um comportamento  $O(1)$ .

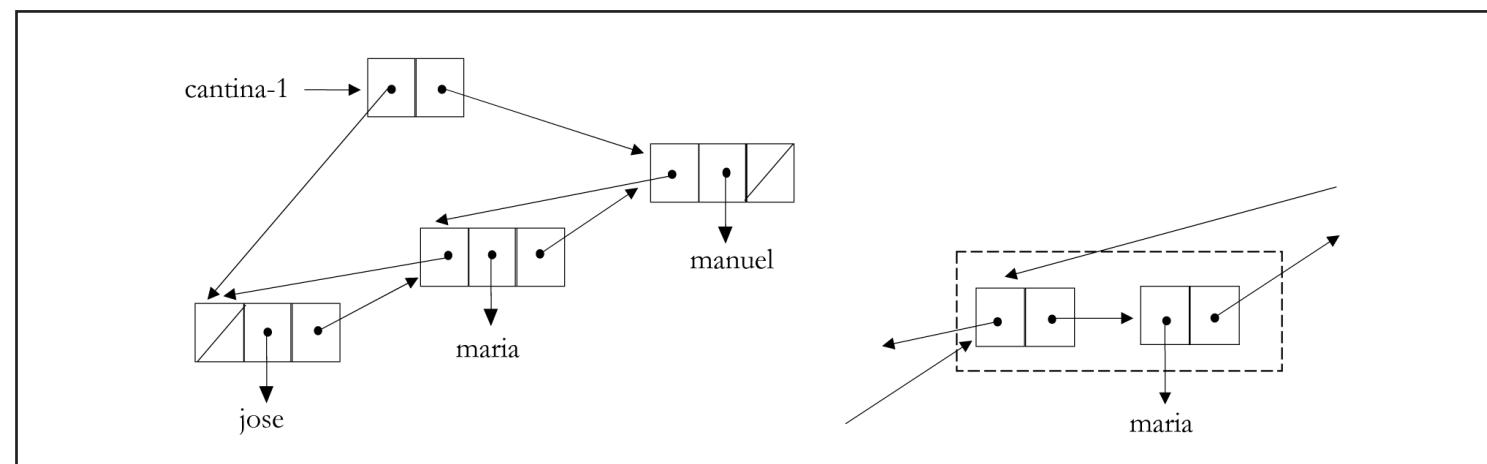
Imagine que estava agora perante uma fila de espera especial, que tinha também operações de entrada no início e de saída no fim!



Acha que estruturar este tipo especial de fila de espera como se fosse um par manteria todas as operações com um comportamento  $O(1)$ ?

Se a resposta for negativa, apresente uma solução em que todas as operações serão  $O(1)$ .

Para que também estas operações se apresentem com um comportamento  $O(1)$ , seria necessário recorrer a uma lista duplamente ligada ou ligada nos dois sentidos, como se pode ver na figura, em que cada elemento da lista duplamente ligada é modelada por dois pares.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



1- Desenvolva o procedimento para visualizar o conteúdo das filas de espera, `visualiza-fila-dupla`, no seguinte formato:

```
> (visualiza-fila-dupla cantina-1)
  (fila-dupla: jose maria manuel)
```

2- Desenvolva, para as filas duplamente ligadas, os seguintes procedimentos:

Construtor

```
(cria-fila-dupla)
```

devolve uma fila duplamente ligada vazia.

Selectores

```
(fila-dupla-vazia? fila)
```

devolve #t, se fila vazia, ou #f, no caso contrário.

```
(frente-da-fila-dupla fila)
```

devolve o primeiro elemento de fila, mas não a altera. Devolve mensagem de erro se fila vazia.

```
(tras-da-fila-dupla fila)
```

devolve o último elemento de fila, mas não a altera. Devolve mensagem de erro se fila vazia.

Modificadores

```
(entra-frente-da-fila-dupla! fila item)
```

insere item no início de fila e devolve o símbolo ok.

```
(entra-tras-da-fila-dupla! fila item)
```

insere item no fim de fila e devolve o símbolo ok.

```
(sai-frente-da-fila-dupla! fila)
```

retira o primeiro elemento de fila e devolve o símbolo ok. Devolve erro se fila vazia.

```
(sai-tras-da-fila-dupla! fila)
```

retira o último elemento de fila e devolve o símbolo ok. Devolve erro se fila vazia.

<b>INTRODUÇÃO</b>
<b>1 - O ESSENCIAL DO SCHEME</b>
<b>2 - RECURSIVIDADE</b>
<b>3 - ABSTRACÇÃO DE DADOS</b>
<b>4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE</b>
<b>5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS</b>
R E 1 2 3 4 5
<b>6 - SCHEME E OUTRAS TECNOLOGIAS</b>
<b>7-EXERCÍCIOS E PROJECTOS</b>
<b>ANEXOS</b>



Desenvolva e teste a abstracção fila de espera duplamente ligada.

### Exercício 3

Escreva em Scheme o procedimento `procura-no-vector` com dois parâmetros, o vector `vec` e o objecto `obj`, que devolve o índice da primeira ocorrência de `obj` em `vec`. Devolve `-1` se `obj` não for um dos elementos de `vec`.

```
> (procura-no-vector '#(g n p r a d l b s) 'a)  
4  
> (procura-no-vector '#(29 13 96 -5 24 11 9 11 2) 11)  
5  
> (procura-no-vector '#(29 13 96 -5 24 11 9 11 2) 10)  
-1
```



Desenvolva e teste o procedimento `procura-no-vector`.



**Indique uma razão que justifique a utilização do valor `-1`, quando o vector não contém o objecto que se procura.**

### Exercício 4

Os preços unitários dos produtos adquiridos por um cliente foram registados no vector `precos`, enquanto as quantidades adquiridas foram registadas no vector `quantidades`.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Escreva em Scheme o procedimento `gasto-total` que espera como argumentos dois vectores, um do tipo `precos` e outro do tipo `quantidades` e responde como a seguir se indica.

```
> (define precos (vector 15.50 8.95 12.00))  
> (define quantidades (vector 2 5 3))  
> (gasto-total precos quantidades)  
111.75
```



Desenvolva e teste o procedimento `gasto-total`.

### Exercício 5

1- Escreva em Scheme o procedimento `soma-elementos-de-vector` que toma como argumento um vector de elementos numéricos e devolve a soma desses elementos.

```
> (soma-elementos-de-vector (vector 1 3 5 7 9 11))  
36  
> (soma-elementos-de-vector '#(10 30 50))  
90
```

2- Escreva em Scheme o procedimento `multiplica-elementos-de-vector`, semelhante ao procedimento `soma-elementos-de-vector`, mas que devolve o produto dos elementos do vector.

```
> (multiplica-elementos-de-vector (vector 1 3 5 7 9))  
945
```

3- Os procedimentos `soma-elementos-de-vector` e `multiplica-elementos-de-vector`, das alíneas anteriores, pela estrutura semelhante que apresentam, sugerem a definição de um procedimento de ordem mais elevada que vamos designar por `acumulador-de-vector`. Com `acumulador-de-vector` a definição daqueles procedimentos seria a seguinte:

```
(define soma-de-elementos-de-vector  
  (lambda (vec)  
    ((acumulador-de-vector + 0) vec)))
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(define multiplica-de-elementos-de-vector
  (lambda (vec)
    (acumulador-de-vector * 1) vec)))
```

Escreva em Scheme o procedimento `acumulador-de-vector` que, como se observa nas duas definições anteriores, espera dois argumentos, um operador e o respectivo valor neutro, e devolve um outro procedimento que tem um vector como único parâmetro.



Desenvolva e teste os procedimentos pedidos nas 3 alíneas.

Indique o que resulta de `(acumulador-de-vector cons '())`.

Procure identificar outras aplicações para `acumulador-de-vector`.

### Exercício 6

Escreva em Scheme o procedimento `vector-map` em duas versões, construtor e modificador, sabendo que na versão construtor responde da seguinte maneira:

```
> (define v (vector 10 11 12 13))
> v
#(10 11 12 13)
> (vector-map add1 v)
#(11 12 13 14)
> (vector-map even? v)
#(#t #f #t #f)
> (vector-map (lambda (el) (if (even? el)
                                  (list 'par el)
                                  (list 'impar el)))
              v)
#((par 10) (impar 11) (par 12) (impar 13) (par 14))
```



Desenvolva e teste as duas versões do procedimento `vector-map`. Na versão modificador, responde com o símbolo `ok`.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



## Exemplo 1

Pretende-se desenvolver uma abstracção designada equipa-de-futebol, baseada nos procedimentos:

Construtor

(cria-equipa)

Cria e devolve uma equipa, completamente vazia.

Selectador

(visu-equipa equipa)

Visualiza a constituição de equipa.

Modificador

(entra-equipa! equipa nome-jogador)

Em equipa entra o jogador designado por nome-jogador. Se já fizer parte da equipa, esta não sofre qualquer alteração. Em ambos os casos devolve o símbolo ok.

(sai-equipa! equipa nome-jogador)

De equipa sai o jogador designado por nome-jogador. Se não fizer parte da equipa, esta não sofre qualquer alteração. Em ambos os casos devolve o símbolo ok.

Na solução que se apresenta, uma equipa de jogadores veteranos é modelada por uma lista mutável com cabeça, de comprimento variável.

```
> (define fcp (cria-equipa))
> (visu-equipa fcp)
()
> fcp
(equipa)
> (entra-equipa! fcp 'jardel)
ok
> (visu-equipa fcp)
(jardel)
> (entra-equipa! fcp 'rui-correia)
ok
> (visu-equipa fcp)
(rui-correia jardel)
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

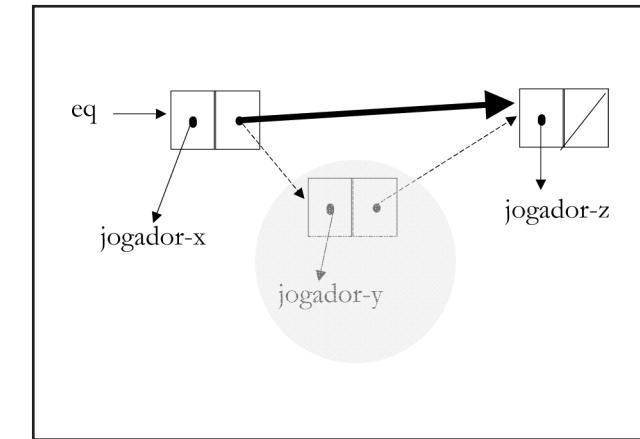
```
> (entra-equipa! fcp 'rui-barros)
ok
> (visu-equipa fcp)
(rui-barros rui-correia jardel)
> (sai-equipa! fcp 'folha)
ok
> (visu-equipa fcp)
(rui-barros rui-correia jardel)
> (sai-equipa! fcp 'jardel)
ok
> (visu-equipa fcp)
(rui-barros rui-correia)
```

Antes de passar à análise da implementação da abstracção, observe a figura que mostra o que acontece quando se retira um jogador-y de uma equipa, com o modificador `sai-equipa!`.

```
(define cria-equipa
  (lambda ()
    (list 'equipa)))

(define entra-equipa!
  (lambda (equipa jogador)
    (if (not (member jogador (cdr equipa)))
        (set-cdr! equipa ; ainda não faz parte da equipa...
                  (cons jogador (cdr equipa))) ; então entra novo jogador
        'ok))) ; em qualquer dos casos, devolve ok

(define sai-equipa!
  (lambda (equipa jogador)
    (letrec ((aux
              (lambda (eq)
                (cond
                  ((null? (cdr eq)) 'ok); não faz parte da equipa... Nada a fazer.
                  ((equal? (cadr eq) jogador)
                   (set-cdr! eq ; faz parte da equipa...
                             (cddr eq)) ; então o jogador é retirado
                   'ok)
                  (else ; não foi encontrado neste ciclo...
                   (aux (cdr eq))))))) ; nova tentativa
;
                (aux equipa)))))
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(define visu-equipa
  (lambda (equipa)
    ... ... ... para completar
```



Complete e teste a abstracção equipa-de-futebol.

### Exercício 7

Estude o exemplo anterior e desenvolva uma nova versão dessa abstracção, agora designada por equipa-de-futebol-listas, em que a equipa é estruturada como se fosse uma lista de 3 elementos, inicialmente com a seguinte composição:

```
((? ? ? ? ? ? ? ? ? ?) (? ? ? ? ? ?) ?)
```

O primeiro elemento será preenchido com os 11 jogadores da equipa principal, o segundo com 6 jogadores suplementares e o terceiro com o treinador. Os procedimentos para criação e manipulação das equipas são os seguintes:

(cria-equipa-fute)

Cria uma equipa, completamente vazia, com o formato acima indicado.

(entra-equi! equipa lugar nome-jogador)

Em equipa entra um jogador, designado por nome-jogador para um certo lugar da equipa principal, especificado por lugar (inteiro de 1 a 11). Se lugar já estiver ocupado, será desocupado, para entrar o novo jogador. Devolve o símbolo ok. Se lugar > 11, visualiza mensagem de erro.

(entra-sup! equipa lugar nome-jogador)

Em equipa entra um jogador, designado por nome-jogador para um certo lugar da equipa suplementar, especificado por lugar (inteiro de 1 a 6).



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Se lugar já estiver ocupado, será desocupado, para entrar o novo jogador. Devolve o símbolo ok. Se lugar > 6, visualiza mensagem de erro.

(entra-treina! equipa nome-treinador)

O treinador designado por nome-treinador ocupa o seu lugar em equipa. Se lugar já estiver ocupado, será desocupado, para entrar o novo treinador. Devolve o símbolo ok.

(sai-equi! equipa nome)

Em equipa, o elemento designado por nome é procurado, primeiro na equipa principal, depois na equipa de suplentes e, finalmente, no lugar do treinador, e é retirado da equipa, ficando o lugar vago. Devolve o símbolo ok. Se não for encontrado, visualiza mensagem de erro.

Nesta abstracção não é controlado o facto de um mesmo jogador ocupar mais do que um lugar na equipa.

```
> (define fcp (cria-equi-fute))
> fcp
((? ? ? ? ? ? ? ? ?) (? ? ? ? ? ?) ?)
> (entra-equi! fcp 1 'correia)
ok
> fcp
((correia ? ? ? ? ? ? ? ?) (? ? ? ? ? ?) ?)
> (entra-equi! fcp 11 'artur)
ok
> fcp
((correia ? ? ? ? ? ? ? ? artur) (? ? ? ? ? ?) ?)
> (entra-equi! fcp 10 'jardel)
ok
> fcp
((correia ? ? ? ? ? ? ? ? jardel artur) (? ? ? ? ? ?) ?)
> (entra-sup! fcp 2 'folha)
ok
> fcp
((correia ? ? ? ? ? ? ? ? jardel artur) (? folha ? ? ? ?) ?)
> (entra-treina! fcp 'oliveira)
ok
> fcp
((correia ? ? ? ? ? ? ? ? jardel artur) (? folha ? ? ? ?) oliveira)
> (sai-equi! fcp 'folha)
ok
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
> fcp
((correia ? ? ? ? ? ? ? jardel artur) (? ? ? ? ? oliveira)
> (entra-equi! fcp 15 's-conceicao)
lugar estranho!...
> (entra-equi! fcp 5 's-conceicao)
ok
> fcp
((correia ? ? ? s-conceicao ? ? ? ? jardel artur) (? ? ? ? ? oliveira)
> (sai-equi! fcp 'artur)
ok
> fcp
((correia ? ? ? s-conceicao ? ? ? ? jardel ?) (? ? ? ? ? oliveira)
> (sai-equi! fcp 'oliveira)
ok
> fcp
((correia ? ? ? s-conceicao ? ? ? ? jardel ?) (? ? ? ? ? ?))
> (sai-equi! fcp 'folha)
este jogador nao esta na equipa!...
> fcp
((correia ? ? ? s-conceicao ? ? ? ? jardel ?) (? ? ? ? ? ?))
```



Desenvolva e teste a abstracção equipa-de-futebol-listas.

Numa segunda fase, introduza as alterações necessárias para evitar que o mesmo jogador possa ocupar, ao mesmo tempo, mais do que um lugar na equipa.

### Exercício 8

Ainda no contexto do exercício anterior, considere agora a modelação de uma equipa como uma lista de 3 elementos, sendo vectores os 2 primeiros. A equipa é inicializada da seguinte maneira:

(#( ? ? ? ? ? ? ? ? ? ? ) #( ? ? ? ? ? ? ) ? ).

Nesta abstracção, equipa-de-futebol-vectores, não é controlado o facto de um mesmo jogador ocupar mais do que um lugar na equipa.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (define fcp (cria-equi-fute-vec))
> fcp
(#(?????????) #(???????))
> (entra-equi-vec! fcp 11 'artur)
ok
> fcp
(#(????????? artur) #(???????))
> (entra-equi-vec! fcp 1 'rui-correia)
ok
> fcp
(#(rui-correia ??????? artur) #(?????))
> (entra-sup-vec! fcp 4 'folha)
ok
> fcp
(#(rui-correia ??????? artur) #(??? folha ??))
> (entra-treina-vec! fcp 'oliveira)
ok
> fcp
(#(rui-correia ??????? artur) #(??? folha ?? oliveira))
> (sai-equi-vec! fcp 'folha)
ok
> fcp
(#(rui-correia ??????? artur) #(??? ?? oliveira))
```



Desenvolva e teste a abstracção equipa-de-futebol-vectores.

Numa segunda fase, introduza as alterações necessárias para evitar que o mesmo jogador ocupe, ao mesmo tempo, mais do que um lugar na equipa.



#### Uma reflexão sobre estruturas de dados

O exemplo e os exercícios anteriores sobre equipas de futebol apresentam como principal diferença a estrutura de dados em que se baseia a modelação da equipa. Afinal, o que se pretende mostrar é que, de uma forma geral, qualquer estrutura pode servir para resolver o problema, mas que algumas se adaptam melhor do que outras. Analise as implementações que tenha desenvolvido e identifique o impacto das diferentes estruturas nas respetivas implementações.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



## Exercício 9

Escreva em Scheme o programa `ocorrencias-de-digitos` que lê um número inteiro e visualiza, para cada dígito decimal, quantas vezes ocorre nesse número.

> (`ocorrencias-de-digitos`)

**Indicar um inteiro:** 1999

```
digito 0: 0    digito 1: 1    digito 2: 0    digito 3: 0  
digito 4: 0    digito 5: 0    digito 6: 0    digito 7: 0  
digito 8: 0    digito 9: 3
```



Desenvolva e teste o procedimento `ocorrencias-de-digitos`.

Pista: Utilizar um vector de 10 posições para ir acumulando a ocorrência de cada um dos dígitos à medida que o número fornecido vai sendo analisado.

## Exercício 10

Numa outra ocasião, foi proposto um exercício que permitia testar o nível de conhecimento sobre a tabuada de multiplicar. Tratava-se do programa `teste-da-tabuada`, com um só parâmetro, `num`, e que colocava num perguntas sobre a tabuada de multiplicar. Os números que surgem nas questões são gerados aleatoriamente.

> (`teste-da-tabuada 10`)

; (para um teste com 10 perguntas)

3 x 8 = 24  
Resposta certa

2 x 7 = 15  
Resposta errada

...

Muito bem

Deve estudar melhor a tabuada

no final das perguntas:

esta mensagem, se 1 ou zero erros.

ou esta, se mais do que 1 erros.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Pretende-se agora desenvolver o programa teste-da-tabuada-melhorado que, relativamente à versão referida, difere apenas nas respostas que vai dando. Assim, em vez de Resposta certa vai aleatoriamente utilizando uma das seguintes hipóteses: Muito Bem; Excelente; Boa resposta; Continue assim. Por outro lado, em vez de Resposta errada vai também escolhendo, aleatoriamente, uma das mensagens: Errou. Tente de novo; Incorrecto. Tente novamente; Tenha calma, pois errou a resposta; Tenha calma, para que o resto corra bem.



Desenvolva e teste o programa teste-da-tabuada.

### Exercício 11

Escreva em Scheme o procedimento cadeia-ao-contrario que recebe uma cadeia de caracteres como argumento e visualiza-a de trás para a frente.

```
> (cadeia-ao-contrario "ab cdef")
fedc ba
```



Desenvolva e teste o procedimento cadeia-ao-contrario.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 12

Escreva em Scheme o predicado `capicua?` que recebe uma cadeia de caracteres como argumento e devolve `#t` se a cadeia recebida tiver a mesma leitura da frente para trás e de trás para a frente.

```
> (capicua? "123abba321")
#t
> (capicua? "")
#t
> (capicua? "12 3abba321")
#f
```



Desenvolva e teste o procedimento `capicua?`.

### Exercício 13

Projecto - Jogo adivinhar-palavras

O programa `adivinhar-palavras` implementa um jogo que desafia o utilizador a tentar adivinhar palavras. O único parâmetro do programa corresponde a um dicionário de palavras, como se indica no exemplo.

```
(define dicionario-1
  (list "curso"
        "nota"
        "livro"
        "jantar"
        "erro"
        "elevador"
        "letra"))
```

pode ter  
vários dicionários

Analise com muita atenção a interacção que se segue.

```
> (adivinhar-palavras dicionario-1)
palavra a adivinhar: (* * * *)      (palavra com 4 letras)
letras erradas: ()                  (para já, não há letras erradas...)
Proxima letra:
n
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
palavra a adivinhar: (n * * *)      (acertou a 1a letra)
letras erradas: ()
Proxima letra:
y
errou...

palavra a adivinhar: (n * * *)      (já há uma letra errada)
letras erradas: (y)
Proxima letra:
r
errou...

palavra a adivinhar: (n * * *)
letras erradas: (y r)
Proxima letra:
t

palavra a adivinhar: (n * t *)
letras erradas: (y r)
Proxima letra:
o

palavra a adivinhar: (n o t *)
letras erradas: (y r)
Proxima letra:
a

palavra a adivinhar: (n o t a)
letras erradas: (y r)
Acertou...
```

1. Faça uma abordagem de-cima-para-baixo ao programa adivinhar-palavras, começando por identificar as suas tarefas principais e, se for necessário, defina uma abstracção de dados adequada.
2. Escreva em Scheme o programa adivinhar-palavras.



Desenvolva e teste o programa adivinhar-palavras.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Escreva a versão `adivinar-palavra-v2` que também tem um único parâmetro e que, em vez de ser um dicionário de palavras como acontecia na versão `adivinar-palavras`, é uma cadeia de caracteres que representa o nome de um ficheiro onde se encontram as palavras do dicionário.

Utilize um editor de texto para preparar os ficheiros de palavras.

Pista: Formato que se sugere para este tipo de ficheiro (uma palavra por linha):

nota  
jantar  
computador

> (`adivinar-palavras-v2 "dicionario-1.txt"`)

`palavra a adivinhar: (* * * * *)`

...

### Exercício 14

Projecto - Helicóptero digital

Um helicóptero imaginário é controlado através do teclado, utilizando-se para tal um conjunto de comandos. O helicóptero, quando no ar, não deixa rasto. Para deixar rasto, ao deslocar-se, terá que estar em contacto com o terreno. Os comandos disponíveis são os seguintes:

Comando	Efeito
1	Põe o helicóptero no ar
2	Põe o helicóptero em contacto com o terreno
3	Faz o helicóptero rodar 90° para a direita
4	Faz o helicóptero rodar 90° para a esquerda
5 d	Desloca o helicóptero (pelo ar ou em contacto com o terreno) d posições para a frente
6	Visualiza o terreno (com os rastos deixados pelo helicóptero)
7	Limpa os rastos deixados pelo helicóptero no terreno



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

- 8 Código não utilizado
- 9 Guarda o estado do terreno e do helicóptero num ficheiro e termina o programa

O terreno é suposto ser um quadrado de dimensão 20 x 20, a que corresponde 20 x 20 células, cada uma identificada pela linha e coluna respectivas. O helicóptero é caracterizado pela sua posição, dada pela linha e coluna onde se encontra (não necessariamente uma linha e uma coluna do terreno, pois o helicóptero pode deslocar-se para fora dos limites daquele), e é ainda caracterizado pela sua orientação (um dos pontos cardinais: N, S, E ou O) e situação (no ar ou no plano do terreno).

```
linha 20 -----  
-----  
-----  
-----  
-----H-----  
...  
-----  
-----  
Linha 1 -----
```

Helicóptero: L16, C11; no terreno; N

O helicóptero é visualizado pela letra H, as células ainda não visitadas são visualizadas com -, enquanto as já visitadas serão representadas por O. Por exemplo, após o comando 5 com d=2, obtém-se:

```
linha 20 -----  
-----  
-----H-----  
-----O-----  
-----O-----  
...  
-----  
-----  
Linha 1 -----
```

Helicóptero: L18, C11; no terreno; N



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

Exemplo de uma sessão com o programa helicóptero-digital, em que as condições iniciais correspondem a terreno limpo e o helicóptero com posição: 16 11; situação: no-terreno; orientação: N.

```
> (helicóptero-digital)
Começa de novo? (s/n): s
```



Com a resposta n, é visualizada a mensagem **Nome do ficheiro:**.  
Esta mensagem será seguida do nome de um ficheiro a abrir em leitura e a simulação será retomada a partir do seu conteúdo.  
Convém relembrar, no Anexo A, os procedimentos relacionados com ficheiros.

```
Comando: 3           (roda para a direita 90°)
Comando: 5           (avança
Posicoes em frente: 3   3 posições)
Comando: 4           (roda para a esquerda 90°)
Comando: 5           ...
Posicoes em frente: 2
Comando: 4
Comando: 5
Posicoes em frente: 6
Comando: 6           (visualiza o terreno)
```

```
-----
-----
-----H000000-----
-----O-----
-----0000-----
...
-----
-----
-----
```

```
Comando: 9
E' para guardar em ficheiro? (s/n): s      com a resposta n, o programa termina de imediato.
Nome do ficheiro: "heli.txt"
Terminou a viagem...
```



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



1. Defina e implemente, se necessário, uma abstracção compatível com o problema exposto.
2. Defina a estrutura dos ficheiros onde guardará o resultado de uma sessão para que possa ser retomada mais tarde.
3. Faça uma abordagem de-cima-para-baixo ao programa helicóptero-digital para identificar as suas tarefas e sub-tarefas principais.
4. Escreva em Scheme o programa helicóptero-digital, considerando os resultados obtidos em 3.

O helicóptero, na sua deslocação, pode ultrapassar os limites do terreno, situação em que não deixa rasto. No entanto, o programa continuará a actualizar as suas características, de acordo com os comandos que vai recebendo.



Desenvolva e teste o programa helicóptero-digital.

Pista para uma abstracção de dados

O terreno pode ser representado por um vector de 21 posições, associando a cada uma delas uma cadeia de 21 caracteres. Sugerem-se 21 e não 20, em ambos os casos, para ser possível desprezar a posição de índice 0 e associar, por exemplo, a posição de índice *i* à linha genérica *i*. A sugestão da cadeia de caracteres como elemento do vector tem em vista facilitar a implementação do comando que visualiza o terreno.

Para representar o helicóptero sugere-se uma lista de dois elementos, *posicao* e *caracteristicas*.

O elemento *posicao* poderá ser uma lista de dois inteiros, que representarão a linha e coluna onde se encontra o helicóptero. Deslocar o helicóptero será fácil de implementar, pois traduzir-se-á em somar ou subtrair o valor correspondente à deslocação em termos de linha ou coluna.

Para o elemento *caracteristicas*, propõe-se uma lista de dois elementos, em que o primeiro, designado por *no-terreno*, é um booleano (#f significa helicóptero no ar e #t sobre o terreno) e o segundo elemento, *direccao*, é um inteiro que pode assumir valores entre 0 e 3, com a codificação seguinte: 0- Norte, 1- Este, 2- Sul, e 3- Oeste. Com esta codificação de *direccao* pretendeu-se simplificar a implementação dos comandos rodar 90° para a direita (modulo (add1 *direccao*) 4) e rodar 90° para a esquerda (modulo (sub1 *direccao*) 4).

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Exercício 15

Projecto - Jogo comer-esferas

O jogo comer-esferas funciona, normalmente, sobre um suporte como o indicado na fotografia.

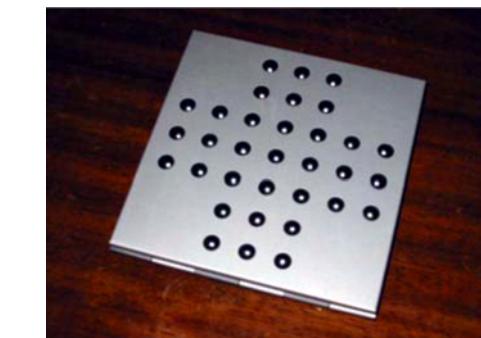
Imagine que, na situação inicial, a esfera central foi retirada, criando um buraco (não é o caso da fotografia, onde todas as esferas estão presentes).

Agora, uma esfera, ao passar por cima de UMA (e UMA só) esfera para tomar o lugar do buraco central, "come" essa esfera.

Vamos tentar explicar melhor, supondo um tabuleiro de jogo em que o símbolo o representa um buraco e o símbolo + representa uma esfera.

O jogo é activado com (comer-esferas) e começa por mostrar o tabuleiro na situação inicial, ou seja, com um buraco na posição central.

```
> comer-esferas)
-----
--- 1 2 3 4 5 6 7  ---
- 1      + + +      1 -
- 2      + + +      2 -
- 3      + + + + + + 3 -
- 4      + + + o + + + 4 -
- 5      + + + + + + + 5 -
- 6      + + +      6 -
- 7      + + +      7 -
--- 1 2 3 4 5 6 7  ---
```



Seguidamente, vai indicar que a esfera situada na linha 4 e coluna 6 vai saltar para o buraco situado na linha 4 e coluna 4. O buraco central desaparece, pois vai lá ser colocada uma esfera, mas, em contrapartida, surgem dois buracos: um correspondente ao local onde estava a esfera que "come" e outro onde estava a esfera "comida".

# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
out, linha/coluna: 4 6
in, linha/coluna: 4 4
-----
--- 1 2 3 4 5 6 7 ---
- 1      + + +
- 2      + + +
- 3      + + + + + +
- 4      + + + + o o +
- 5      + + + + + +
- 6      + + +
- 7      + + +
--- 1 2 3 4 5 6 7 ---
```

Agora, é a esfera situada na linha 2 e coluna 5 que vai saltar para o buraco situado na linha 4 e coluna 5, passando por cima da esfera situada na linha 3 e coluna 5. Observe os buracos criados...

```
out, linha/coluna: 2 5
in, linha/coluna: 4 5
-----
--- 1 2 3 4 5 6 7 ---
- 1      + + +
- 2      + + o
- 3      + + + + o + +
- 4      + + + + + o +
- 5      + + + + + +
- 6      + + +
- 7      + + +
--- 1 2 3 4 5 6 7 ---
```

O jogo termina quando não é possível fazer mais jogadas, situação essa que é detectada pelo programa. O objectivo principal é terminar apenas com uma esfera. Todavia, ganha quem ficar com o menor número de esferas por comer...

Tenha em atenção que, na indicação de uma posição de saída ou de entrada, o programa rejeita situações de deslocação impossível. Como situações de deslocação impossível temos:

- Ao indicar uma posição de saída, 1- esta não é uma esfera ou 2- na sua linha ou coluna, não existe um percurso que termina num buraco e com uma única esfera na passagem.



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

- Ao indicar uma posição de entrada, 1- esta não é um buraco ou 2- não se encontra na linha ou na coluna da posição indicada como saída, a uma distância de 2 e com uma única esfera entre elas.

Pretende-se que desenvolva o projecto relativo ao jogo comer-esferas, tendo em conta os seguintes aspectos:

1. Defina e implemente uma abstracção para apoio à concretização do jogo, indicando uma estrutura de dados adequada e os respectivos operadores (construtores, selectores e modificadores).
2. Faça uma abordagem de-cima-para-baixo ao programa comer-esferas para identificar as suas tarefas e sub-tarefas principais.
3. Escreva em Scheme o programa comer-esferas, considerando os resultados obtidos em 2.

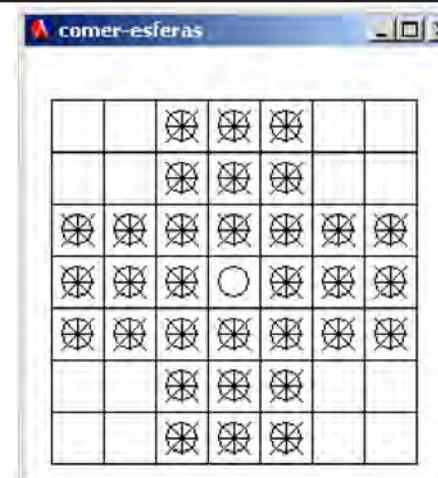


Desenvolva e teste o programa do jogo comer-esferas.

Depois de verificar a correcção do programa pedido, desenvolva uma interface gráfica para este jogo, baseada na abstracção tabuleiro. Esta abstracção, apresentada no **Anexo D**, fica disponível através de

(require (lib "tabuleiro.scm" "user-feup"))

Atrevo-me a adivinhar que, caso tenha definido uma abstracção adequada para apoio à implementação do jogo, é muito natural que o desenvolvimento da interface gráfica não exija um grande esforço da sua parte... Será que adivinhei?



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

## Módulo 5.1 - Não só cria, o Scheme também modifica...

Até aqui, o Scheme não permitia modificar as entidades criadas, obrigando-nos a recorrer a artifícios do tipo: a modificação de uma entidade é simulada criando uma nova entidade com o mesmo nome da entidade que se pretendia modificar. Como verá, esta solução pode não ser a mais adequada e nem sempre resulta. A forma correcta para realizar este tipo de operação recorre aos chamados modificadores que o Scheme disponibiliza para modificar entidades já existentes, com o modificador `set!`, ou com os modificadores `set-car!`, `set-cdr!` e `append!`. Surge assim a lista mutável.

Com estes modificadores poderá criar abstracções de dados mutáveis com alguma complexidade, nomeadamente filas de espera e tabelas, temas a tratar num outro módulo.

### Palavras-Chave

Modificadores, lista mutável.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## A necessidade de usar modificadores

As entidades ou objectos computacionais têm uma história associada, que será actualizada sempre que ocorra algum acontecimento relevante com eles relacionado. Suponha que uma conta bancária é apenas caracterizada pelo saldo e que este é associado a uma variável. Quando se processam movimentos naquela conta, depósitos ou levantamentos, o respectivo saldo deverá ser actualizado, ou seja, a situação da variável deverá ser devidamente alterada. Até aqui, o Scheme não nos permitia actualizar os valores associados aos seus objectos, limitação essa que complica a representação do saldo da conta bancária. Poderá imaginar que aplicando, por exemplo, um `define` a um objecto anteriormente definido, o seu valor é alterado.

```
> (define objecto-a 10)
> objecto-a
10
> (define objecto-a (+ 3 objecto-a))
> objecto-a
13
```

De facto, o valor não foi alterado, apenas foi criada uma nova entidade com o mesmo nome, desaparecendo a anterior.

Apesar de `objecto-a` ser uma nova entidade, parece que tudo se comporta como se fosse uma actualização da entidade inicial. No exemplo descrito, a criação de um novo objecto até nem se torna demasiado inconveniente. Todavia, assim já não acontece se o objecto em causa fosse, por exemplo, uma lista de centenas ou milhares de elementos. Neste caso, a simples alteração de um desses elementos obrigaria à criação de uma nova lista, com o mesmo nome da lista inicial, copiando as centenas ou milhares dos seus elementos e alterando apenas um deles.

Como vimos, a alteração de um objecto não é possível com a funcionalidade que o Scheme nos ofereceu até aqui. Esta impossibilidade acabou por ser camouflada criando uma nova entidade, a partir da entidade inicial, com os inconvenientes já referidos. Mas esta solução nem sempre funciona. Por exemplo, uma entidade local, criada dentro de um procedimento, podendo ter o mesmo nome de uma outra existente no exterior do procedimento, não será a mesma coisa. Aliás, a entidade local nem sequer será acessível fora desse procedimento. Os objectos definidos localmente num procedimento só dentro dele serão reconhecidos e acessíveis, pois ao terminar a execução de um procedimento também terminam os objectos nele definidos. A propósito, analise o exemplo que se segue.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(define altera-obj-a
  (lambda ()
    (let ((objecto-a (+ 5 objecto-a)))
      (* objecto-a 2)))

> objecto-a
13
> (altera-obj-a)
36
> objecto-a
13
```

Ao definir `objecto-a` dentro do procedimento `altera-obj-a`, a expressão `(+ 5 objecto-a)` é calculada procurando `objecto-a` no nível imediatamente acima, neste caso, o Ambiente Global do Scheme, uma vez que `objecto-a` não existe ainda definido dentro do procedimento. No entanto, o cálculo de `(* objecto-a 2)` já utiliza `objecto-a` acabado de definir no interior do procedimento. Não esquecer que esta variável local `objecto-a` desaparece logo que termina a execução do procedimento onde foi criada, mas a variável com o mesmo nome, definida no Ambiente Global, continua a existir e mantém o valor 13. Não foi portanto modificada, como se imaginaria.

Este tipo de situação poderá ser resolvida através de uma forma especial do Scheme, o modificador `set!`, que apresenta a forma genérica que agora se indica.

```
(set! nome expressão)
```

Segundo a regra de cálculo de `set!`, expressão é calculada e o valor resultante é associado ao identificador `nome` que, entretanto, já deve existir. Analise com atenção o exemplo que se segue.

```
(define altera-mesmo-obj-a!
  (lambda ()
    (set! objecto-a (+ 5 objecto-a))
    (* objecto-a 2))

> objecto-a
13
> (altera-mesmo-obj-a!)
36
> objecto-a
18
```



# S C H E M E

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

Como objecto-a não é reconhecido no procedimento `altera-mesmo-obj-a!`, por não ter sido criado no procedimento nem ser seu parâmetro, vai ser procurado no nível acima, neste caso, no Ambiente Global do Scheme. O objecto-a é, no procedimento referido, um objecto livre, e `set!` vai conseguir modificá-lo onde ele se encontrar.

Quando se considerou o tema Abstracção de dados, foram referidos, do Scheme, o construtor `cons` e os selectores `car` e `cdr`. A partir de agora, dispõe-se também do modificador `set!`. Os objectos manipulados pelos modificadores são designados por objectos mutáveis ou dados mutáveis, pois foi alterado o valor que lhes está associado.

**Exemplo 1** - o modificador `set!`

O José, o António e a Maria compram e vendem um certo tipo de artigo, do qual apenas pretendemos conhecer a quantidade que cada um tem em armazém. Inicialmente, o armazém de cada um está completamente vazio. Para resolver este problema, do ponto de vista dos dados, vamos considerar 3 entidades simples, independentes, para representar os 3 armazéns, designando-os por `jose`, `antonio` e `maria`. Para completar esta abstracção, vamos considerar o procedimento `mostra-situacao`, que funciona como selector, e os procedimentos `vende!` e `compra!`, que são dois modificadores, cuja funcionalidade pode deduzir-se da interacção que se segue:

```
> (mostra-situacao)

jose: 0
antonio: 0
maria: 0
> (compra! 'maria 70)

jose: 0
antonio: 0
maria: 70
> (compra! 'jo 50)
engano!!!
jose: 0
antonio: 0
maria: 70
> (compra! 'antonio 60)
```



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
jose: 0
antonio: 60
maria: 70
> (compra! 'maria 50)
```

```
jose: 0
antonio: 60
maria: 120
> (vende! 'antonio 20)
```

```
jose: 0
antonio: 40
maria: 120
> (vende! 'maria 100)
```

```
jose: 0
antonio: 40
maria: 20
```

A solução que se apresenta parece ser suficientemente simples, dispensando grandes comentários. A representação dos armazéns é feita definindo três variáveis globais com o conteúdo inicial igual a zero.

```
(define jose 0)      ; a representação
(define antonio 0)    ; dos três
(define maria 0)      ; armazéns

(define mostra-situacao
  (lambda ()
    (newline)
    (display "jose: ")
    (display jose)
    (newline)
    (display "antonio: ")
    (display antonio)
    (newline)
    (display "maria: ")
    (display maria)))
```

Os modificadores, cujos nomes em Scheme terminam habitualmente com um ponto de exclamação, são agora apresentados.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(define compra!          ; compra de artigo
  (lambda (nome quantidade) ; aumenta a quantidade de artigo em armazém
    (cond
      ((equal? nome 'jose) (set! jose (+ jose quantidade)))
      ((equal? nome 'antonio) (set! antonio (+ antonio quantidade)))
      ((equal? nome 'maria) (set! maria (+ maria quantidade)))
      (else
        (display "engano!!!")
        (newline)))
      (mostra-situacao)))
```

```
(define vende!          ; venda de artigo
  (lambda (nome quantidade) ; diminui a quantidade de artigo em armazém
    (cond
      ((equal? nome 'jose) (set! jose (- jose quantidade)))
      ((equal? nome 'antonio) (set! antonio (- antonio quantidade)))
      ((equal? nome 'maria) (set! maria (- maria quantidade)))
      (else
        (display "engano!!!")
        (newline)))
      (mostra-situacao)))
```



Teste os modificadores `compra!` e `vende!`.

Posteriormente, preveja as tentativas de venda superiores à quantidade de artigo armazenado, definindo a atitude a tomar face a este tipo de situação e as alterações correspondentes ao nível do procedimento `venda!`.



Neste exemplo, há um aspecto que lhe retira qualquer interesse prático (a não ser a ilustração da utilização do modificador `set!`).

Identifique o aspecto a que nos estamos a referir.

Vai, provavelmente, confirmar a sua resposta no que se segue.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Modificadores que actuam sobre pares

Até aqui, os dados mutáveis têm sido manipulados através de `set!`. Para dados compostos, baseados em pares, o Scheme disponibiliza três modificadores:

`(set-car! par expressão)`

O modificador `set-car!` calcula `expressão` e associa o valor resultante ao elemento da esquerda de `par`. O valor que devolve é indefinido.

`(set-cdr! par expressão)`

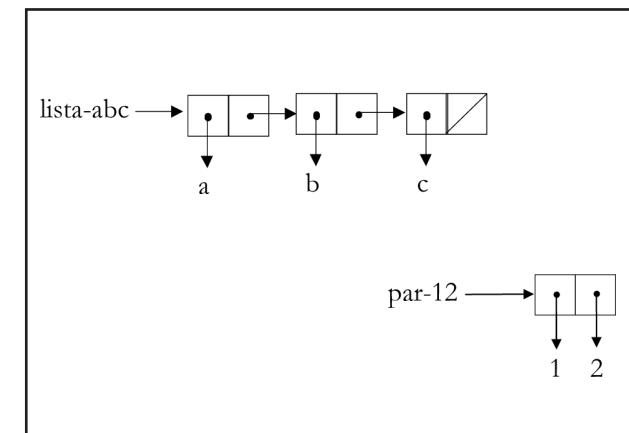
O modificador `set-cdr!` calcula `expressão` e associa o valor resultante ao elemento da direita de `par`. O valor que devolve é indefinido.

`(append! lista-1 lista-2 lista-3 ... lista-n)`

O modificador `append!` rompe a parte direita do último par da `lista-1` e fá-lo apontar para `lista-2`, em seguida faz algo de semelhante à `lista-2`, cuja parte direita do último par é posta a apontar para a `lista-3`, e assim sucessivamente até `lista-n`, que não sofre qualquer modificação. O valor que devolve é indefinido.

É conveniente que analise com muita atenção os exemplos que se seguem.

```
> (define lista-abc (list 'a 'b 'c))
> lista-abc
(a b c)
> (define par-12 (cons 1 2))
> par-12
(1 . 2)
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

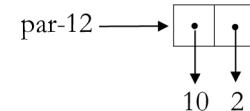
## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

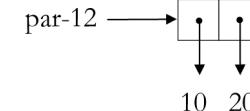
## ANEXOS

Através de `set-car!`, o elemento da esquerda de `par-12` é modificado, passando a ser 10. Por outro lado, através de `set-cdr!`, o elemento da direita deste par passa a ser 20.

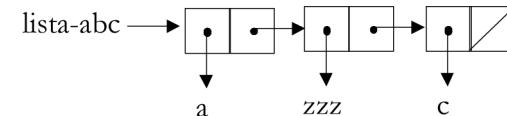
```
> (set-car! par-12 10)
> par-12
(10 . 2)
```



```
> (set-cdr! par-12 20)
> par-12
(10 . 20)
```

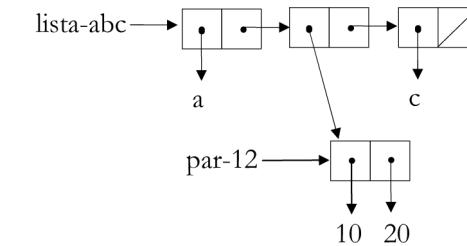


Como fará para alterar o segundo elemento de `lista-abc`, como se indica na figura?



O segundo elemento de `lista-abc` é outra vez modificado e passa a apontar para o par designado por `par-12`.

```
> (set-car! (cdr lista-abc) par-12)
> lista-abc
(a (10 . 20) c)
```



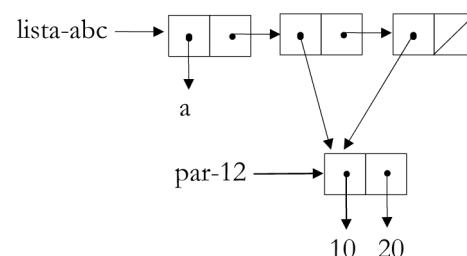


**Esta alteração teve algum impacto em par-12?**



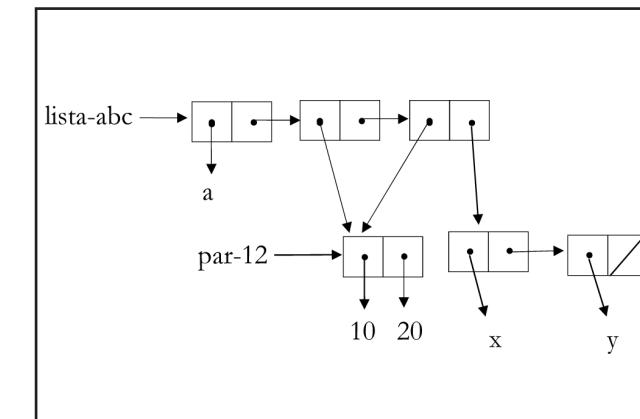
E agora, como fará para alterar `lista-abc`, como se indica na figura que se segue? Depois dessa alteração, como responde o Scheme face a

> lista-abc  
???



Para finalizar esta sequência de exemplos, o elemento da direita do último par de `lista-abc` é modificado, ficando a apontar para a lista composta pelos símbolos `x` e `y`.

```
> (set-cdr! (cddr lista-abc) '(x y))  
> lista-abc  
(a (10 . 20) (10 . 20) x y)
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



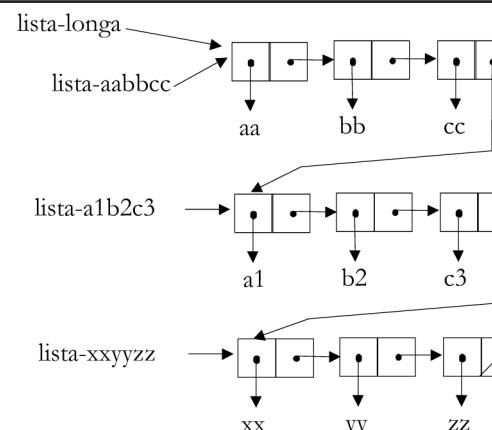
Os próximos exemplos mostram como funciona o modificador `append!`.

Inicialmente são criadas as listas `lista-aabbcc`, `lista-a1b2c3` e `lista-xxyyzz`, cada uma delas com três elementos.

```
> (define lista-aabbcc '(aa bb cc))
> (define lista-a1b2c3 '(a1 b2 c3))
> (define lista-xxyyzz '(xx yy zz))
```

Através do modificador `append!` é criada uma lista, `lista-longa`, com nove elementos.

```
> (define lista-longa (append! lista-aabbcc
                                 lista-a1b2c3 lista-xxyyzz))
> lista-longa
(aa bb cc a1 b2 c3 xx yy zz)
```



Convém agora que verifique como ficam as listas utilizadas na criação de `lista-longa`.

```
> lista-aabbcc
???
> lista-a1b2c3
???
> lista-xxyyzz
???
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

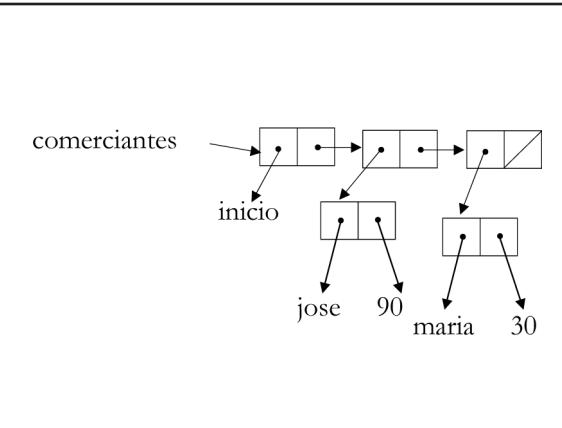
### ANEXOS



Experimente os modificadores `set-car!`, `set-cdr!` e `append!`

### Exemplo 2 - os modificadores `set-car!`, `set-cdr!` e `append!`

Vai agora retomar o exemplo dos comerciantes, mas em vez de representar os armazéns como dados independentes, consideram-se constituindo uma lista única. Na figura, a lista foi designada por `comerciantes` e os seus elementos são pares. Em cada um desses pares o elemento da esquerda corresponde ao nome do comerciante e o da direita à quantidade de artigo armazenado.



Como pode verificar, existe um elemento no início da lista, o símbolo `inicio`, que garante, mesmo na ausência de comerciantes, que a lista nunca será nula.

Este elemento é designado por cabeça da lista.

Fique atento ao desenvolvimento dos modificadores que se seguem, pois pretende-se que apresente uma justificação para a inclusão da cabeça da lista.

Com os modificadores estudados, torna-se relativamente fácil alterar a lista, nomeadamente acrescentar novos elementos ou alterar elementos existentes.

Para completar a definição da abstracção, considere os procedimentos que se indicam de seguida.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Construtor

(faz-lista-de-comerciantes)

devolve uma lista constituída apenas pelo elemento cabeça de lista, neste caso, o símbolo `inicio`.

Selector

(mostra-situacao lista-comerciantes)

visualiza a lista `lista-comerciantes`, sem mostrar a cabeça da lista.

Modificadores

(junta-novo-comerciante! lista-comerciantes comerciante quantidade)

imediatamente após a cabeça da lista de `lista-comerciantes`, junta comerciante, associando-lhe quantidade e devolve o símbolo `ok`.

(compra! lista-comerciantes comerciante quantidade)

Em `lista-comerciantes`, adiciona quantidade a comerciante e devolve o símbolo `ok`. No entanto, se comerciante não existir, visualiza uma mensagem adequada.

(vende! lista-comerciantes comerciante quantidade)

Em `lista-comerciantes`, subtrai quantidade a comerciante e devolve o símbolo `ok`. Se comerciante não existir, visualiza uma mensagem adequada.

A funcionalidade destes procedimentos pode clarificar-se com o que se apresenta.

> (define comerciantes (faz-lista-de-comerciantes))

> comerciantes

(**inicio**)

> (mostra-situacao comerciantes)

Nao ha' mais comerciantes



Esteja atento à diferença na visualização de uma lista de comerciantes através de `mostra-situacao` ou pedindo o seu valor ao Scheme como se fosse uma lista qualquer.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

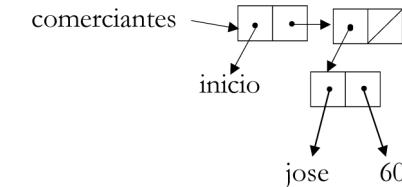
## ANEXOS

Verifique que cada novo elemento é colocado imediatamente após a cabeça da lista.

```
> (junta-novo-comerciante! comerciantes 'jose 60)
ok
> (mostra-situacao comerciantes)
```

```
jose: 60
Nao ha' mais comerciantes

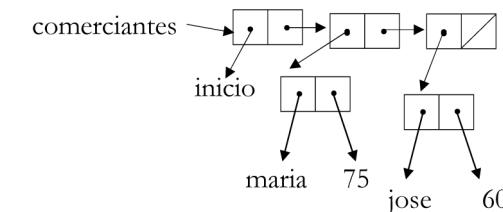
> comerciantes
(inicio (jose . 60))
```



```
> (junta-novo-comerciante! comerciantes 'maria 75)
ok
> (mostra-situacao comerciantes)
```

```
maria: 75
jose: 60
Nao ha' mais comerciantes

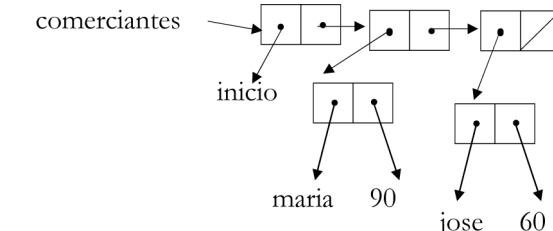
> comerciantes
(inicio (maria . 75) (jose . 60))
```



Agora surge o modificador `compra!` para aumentar a quantidade associada a um comerciante.

```
> (compra! comerciantes 'maria 15)
ok
> (mostra-situacao comerciantes)

maria: 90
jose: 60
Nao ha' mais comerciantes
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

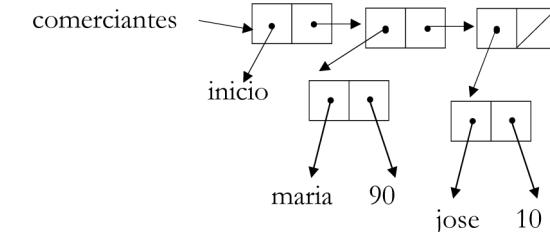
## ANEXOS

Por outro lado, com o modificador `vende!`, diminui-se a quantidade associada a um comerciante.

```
> (vende! comerciantes 'jose 50)
ok

> (compra! comerciantes 'jos 45)
jos nao e' comerciante
> (mostra-situacao comerciantes)

maria: 90
jose: 10
Nao ha' mais comerciantes
```



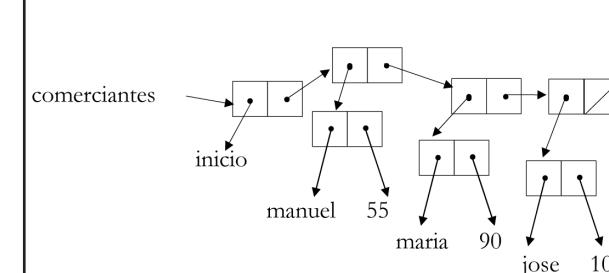
Atenção à forma como reagiu o modificador `compra!` quando se lhe apresentou um nome não incluído na lista de comerciantes.

Algo de semelhante acontecerá também com o modificador `vende!`!

Para finalizar, junte mais um comerciante à lista.

```
> (junta-novo-comerciante! comerciantes 'manuel 55)
ok
> (mostra-situacao comerciantes)

manuel: 55
maria: 90
jose: 10
Nao ha' mais comerciantes
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Depois desta sequência de exemplos com que se pretendeu clarificar a funcionalidade desta abstracção, apresenta-se uma solução que deve analisar cuidadosamente e completar.

```
(define faz-lista-de-comerciantes
  (lambda ()
    (list 'inicio)))

(define mostra-situacao
  (lambda (lista)
    (letrec ((aux
              (lambda (lis)
                (cond ((null? lis)
                       (newline)
                       (display "Nao ha' mais comerciantes"))
                      (else
                        .... ... para completar

                        (aux (cdr lista)))))))
      (define junta-novo-comerciante!
        (lambda (lista nome quantidade)
          (let ((novo (cons
                      (cons nome quantidade)
                      (cdr lista))))
            (set-cdr! lista novo)
            'ok)))))


```



**Não deve avançar para o modificador `compra!` sem entender muito bem o modificador `junta-novo-comerciante!`!**

Ajudará muito comparar, por exemplo, as duas figuras anteriores que representam a lista `comerciantes` antes e depois da inclusão do comerciante `manuel`...

No procedimento `junta-novo-comerciante!`, veja como se definiu a entidade `novo` e depois como esta foi incluída na lista `comerciantes`, através de `set-cdr`!



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
(define compra!
  (lambda (lista nome quantidade)
    (letrec ((aux           ; o procedimento interno aux procura
              (lambda (lis)      ; na lista comerciante...
                .... .... para completar
                (aux (cdr lista))))))
      ...)
```



Se não consegue completar o modificador **compra!**, sugere-se que analise o modificador **vende!** que se segue.

```
(define vende!
  (lambda (lista nome quantidade)
    (letrec ((aux           ; o procedimento interno aux procura
              (lambda (lis)      ; na lista um certo comerciante...
                (cond ((null? lis)
                        (display nome)           ; se não encontra...
                        (display " nao e' comerciante"))
                      ((equal? nome (caar lis))
                        (set-cdr! (car lis)       ; se encontra...
                                  (- (cdar lis)
                                     quantidade)))
                        'ok)
                      (else
                        (aux (cdr lis)))))))
      (aux (cdr lista))))
```



Complete e teste a abstracção apresentada.

Nos testes, tente também vender para além do que um comerciante tenha em armazém. O que acontece?

Tente também juntar um comerciante que já exista. O que acontece?

Indique como deve o programa reagir face a estas duas situações e introduza as alterações necessárias, provavelmente ao nível de **vende!** e **junta-novo-comerciante!**.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Quando seleccionámos uma lista, com uma cabeça de lista, para estrutura de dados desta abstracção foi-lhe sugerido que estivesse com atenção ao desenvolvimento dos modificadores, a fim de descobrir a utilidade deste pormenor (existência de uma cabeça de lista).

Descobriu essa utilidade?

Se não descobriu, tente reescrever o modificador **junta-novo-comerciante!** com uma lista sem cabeça de lista.

Certamente encontrará algumas surpresas quando fizer o seu teste!...

Mas se as dúvidas persistirem, não desanime e esteja atento até ao final da próxima secção...

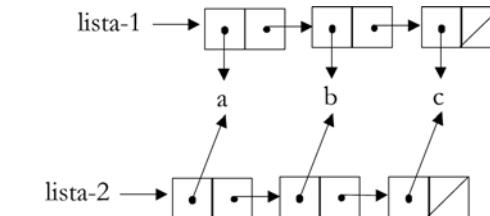
### A utilização de modificadores requer muita atenção

A utilização dos modificadores origina, por vezes, algumas surpresas, como se pretende mostrar nos exemplos que se seguem.

```
> (define lista-1 '(a b c))  
> (define lista-2 '(a b c))
```

A representação gráfica de lista-1 e lista-2 mostra que o Scheme não duplica os símbolos que cria, pois a, b e c são partilhados pelas duas listas. Apesar do conteúdo destas ser exactamente o mesmo, vejamos a resposta de eq? e equal? quando as duas listas são comparadas.

```
> (eq? lista-1 lista-2)  
#f  
> (equal? lista-1 lista-2)  
#t
```





Tente alguma justificação para estas respostas do Scheme, antes de ver o que se segue...

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



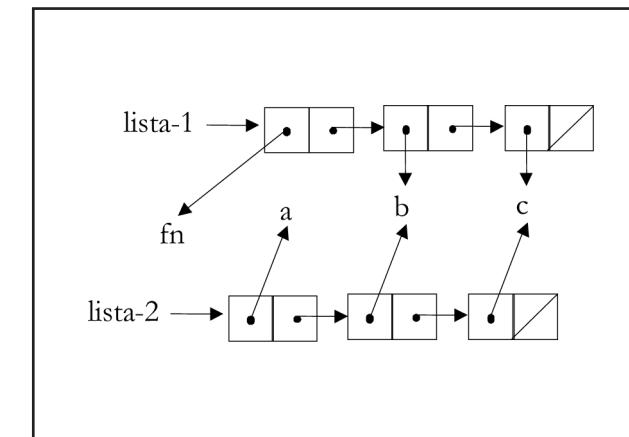
A resposta #f justifica-se pelo facto de eq? comparar os apontadores para o início de cada uma das listas. Esses apontadores são diferentes, como pode observar na figura.

A resposta #t justifica-se, pois equal? compara não os referidos apontadores, como acontecia com eq?, mas sim os conteúdos das listas.

Analise o que acontece perante a alteração de um elemento de uma das listas.

```
> (set-car! lista-1 'fn)
> lista-1
(fn b c)
> lista-2
(a b c)
```

A modificação de elementos de lista-1 não influenciou lista-2, pois, apesar de partilharem os seus elementos, são listas criadas de uma forma independente, ou seja, enumerando os elementos de cada uma delas.



Uma análise visual desta situação parece muito útil.

Verifique que ao criar lista-1 e lista-2 por enumeração dos elementos, ou seja, (`(define lista-1 '(a b c))` e `(define lista-2 '(a b c))`, está de facto a definir apontadores independentes, lista-1 e lista-2, simbolizados por setas.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

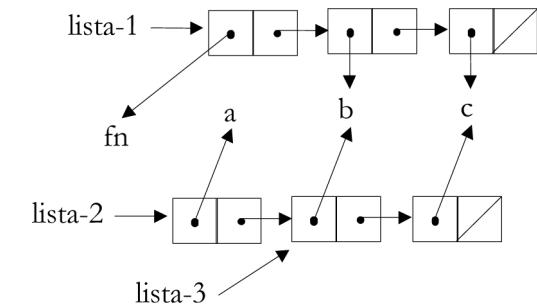
## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Agora uma situação completamente diferente: `lista-3` é criada como sendo uma parte de `lista-2`, ou seja, `(cdr lista-2)`, e não por enumeração de elementos.

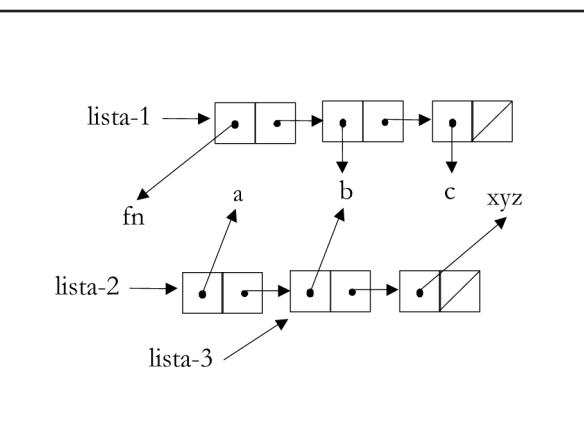
```
> (define lista-3 (cdr lista-2))  
> lista-3  
(b c)
```



Mas será que este pormenor é realmente importante quando se modificam valores em `lista-2` e `lista-3`?

```
> (set-car! (cddr lista-2) 'xyz)  
> lista-1  
(fn b c)  
> lista-2  
(a b xyz)  
> lista-3  
(b xyz)
```

A modificação de elementos de `lista-2` acabou por se repercutir em `lista-3`, mas até pela observação da figura poderia ver que isso ia acontecer.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Que elementos de `lista-2` poderiam ser modificados sem influenciar `lista-3`?  
E, por outro lado, há elementos de `lista-3` que poderiam ser modificados sem influenciar `lista-2`?



Experimente os exemplos apresentados.

Atenção, portanto, às modificações quando envolvem entidades constituídas a partir de outras entidades já existentes...

### Exercício 1

Com as definições

```
> (define lista-a '(a b c))  
> (define lista-b lista-a)
```

indique e justifique as respostas do Scheme representadas por ??? no diálogo que se segue.

```
> (set-car! lista-a 'fn)  
???  
> lista-a  
???  
> lista-b  
???
```



Verifique as suas respostas com a ajuda do Scheme.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

O que segue mostra o comportamento do modificador `append!` perante as listas vazias que lhe são apresentadas como argumentos.



**Analise a sequência que se segue e tente deduzir o comportamento de `append!` quando encontra listas vazias como argumentos.**

```
> (define lis1 (list))
> (define lis5 (list 5 6 7))
> (define lis9 (list 9 10))

> (append! lis1 lis5 lis9)
(5 6 7 9 10)
> lis1
()
> lis5
(5 6 7 9 10)
> lis9
(9 10)
```

Deve ter concluído que o modificador `append!` não processa as listas vazias que lhe surgem como argumentos, pois tudo se passa como se elas não fizessem parte desses argumentos.

### Exercício 2

Com as definições

```
> (define lis1 (list))
> (define lis5 (list 5 6 7))
> (define lis9 (list 9 10))
```

indique e justifique as respostas do Scheme representadas por ???, no diálogo que se segue.

Neste exercício, sugere-se que utilize a representação gráfica das listas para procurar as respostas.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (append! lis9 lis1 lis5)
(9 10 5 6 7)
> lis1
???
> lis5
???
> lis9
???
```



Verifique as suas respostas com a ajuda do Scheme.



**Não se pode esquecer que, quando um procedimento é chamado, só nesse momento é criado espaço para receber os argumentos e que esse espaço será ocupado por cópias dos valores que lhes dão origem...  
Ou seja, quando um argumento é um objecto simples, o espaço criado será ocupado com uma cópia desse objecto simples.**

**Por outro lado, quando um argumento é um objecto composto, por exemplo, uma lista, o espaço criado não será ocupado por uma cópia da lista, mas sim por uma cópia de um apontador para essa lista...**



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
; o procedimento recebe um objecto simples...
;
(define tenta-mudar-entidade-simples!
  (lambda (entidade)
    (display "entidade no inicio: ")
    (display entidade)
    (newline)
    (set! entidade 200)
    (display "entidade no fim: ")
    (display entidade)))
> (define x 154)
> (tenta-mudar-entidade-simples! x)
entidade no inicio: 154
entidade no fim: 200
> x
154
```



Verifique a evolução do valor **x** e o que acontece no interior do procedimento...  
Por que razão não foi alterado **x**?

Deverá ter concluído que **x** não foi alterado, pois o procedimento apenas alterou uma cópia de **x** ...

```
; o procedimento recebe um apontador para uma lista...
;
(define tenta-mudar-lista!
  (lambda (lista)
    (display "lista no inicio: ")
    (display lista)
    (newline)
    (set! lista (list 11 12 13))
    (display "lista no fim: ")
    (display lista)))

> (define listal (list 1 2 3))
> (tenta-mudar-lista! listal)
lista no inicio: (1 2 3)
lista no fim: (11 12 13)
> listal
(1 2 3)
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Verifique agora a evolução da lista designada por `list1` e o que acontece no interior do procedimento...  
Por que razão não foi alterada `list1`?

Aqui deverá ter concluído que a lista `list1` não foi alterada, pois o procedimento apenas alterou uma cópia de um apontador para essa lista...

```
; o procedimento recebe um apontador para uma lista...
;
(define muda-lista!
  (lambda (lista)
    (display "lista no inicio: ")
    (display lista)
    (newline)
    (set-car! (cdr lista) 1234)
    (display "lista no fim: ")
    (display lista)))

> (define lista5 (list 5 6 7))
> (muda-lista! lista5)
lista no inicio: (5 6 7)
lista no fim: (5 1234 7)
> lista5
(5 1234 7)
```



Verifique agora a evolução da lista designada por `list5` e o que acontece no interior do procedimento...  
Por que razão foi alterada `list5`?

Neste caso, a lista `list5` foi alterada, pois, através de uma cópia de um apontador para a lista `list5`, o procedimento conseguiu alterar essa lista...



Experimente os três procedimentos apresentados.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



Explique o que acontece no diálogo que se segue.

```
> (define lista-v1 (list))  
> (muda-lista! lista-v1)  
lista no inicio: ()  
 cdr: expects argument of type <pair>; given ()  
>
```

E o que resultaria de

```
> (tenta-mudar-lista! lista-v1)  
???
```

### Exercício 3

Com estas definições:

```
(define muda-lista-v2!  
  (lambda (lista)  
    (display "lista no inicio: ")  
    (display lista)  
    (newline)  
    (append! lista (list 21 22 23))  
    (display "lista a meio: ")  
    (display lista)  
    (newline)  
    (set-car! (cdr lista) 1234)  
    (display "lista no fim: ")  
    (display lista)))  
  
(define lista1 (list 1 2 3))  
(define lista-v1 (list))
```

Indique e justifique as respostas do Scheme representadas por ??? no diálogo que se segue.

```
> (muda-lista-v2! lista1)  
???  
> lista1  
???  
> (muda-lista-v2! lista-v1)  
???  
> lista-v1  
???
```



# S C H E M E

## na descoberta da programação



Verifique as suas respostas com a ajuda do Scheme.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



- 1- Dúvida ainda da importância de utilizar uma lista com cabeça, na lista `comerciantes` do Exemplo 2?
- 2- Consegue agora identificar o tipo de situação em que é importante ancorar uma lista, num certo ponto, numa cabeça, evitando assim que a lista se apresente ao Scheme completamente vazia?



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



## Módulo 5.2 - Filas, Pilhas e Tabelas são estruturas mutáveis de grande utilidade

Uma forma correcta de trabalhar com entidades que admitem alterações, os dados mutáveis, recorre aos chamados modificadores que o Scheme disponibiliza para alterar entidades já existentes, com o modificador `set!`, ou com os modificadores `set-car!`, `set-cdr!` e `append!`.

Com estes modificadores pode criar abstracções de dados mutáveis com alguma complexidade, nomeadamente filas de espera, tabelas e pilhas.

A técnica de tabulação (*memoization* ou *tabulation*) surge como uma forma de guardar resultados numa tabela local para serem reutilizados posteriormente, tendo uma enorme repercussão nos tempos de resposta.

### Palavras-Chave

Filas de espera, pilhas, topo da pilha, tabelas, chave, tabela unidimensional, tabela bidimensional, tabulação (*memoization* ou *tabulation*).

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

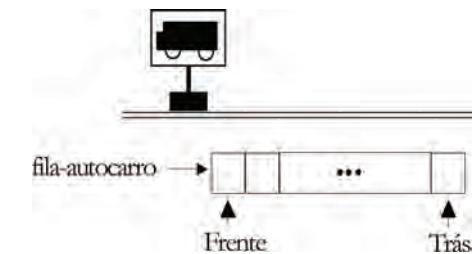
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## No nosso quotidiano surgem frequentemente filas de espera

Uma abstracção de dados de grande interesse é a chamada fila de espera, identificada muitas vezes por FIFO – *First In First Out*, em que o primeiro elemento a chegar deverá ser o primeiro a sair. Facilmente se reconhece a sua utilidade na simulação de uma fila de espera para o autocarro, para a cantina ou para veículos que se aproximam de um cruzamento. Na fila de espera, o primeiro elemento a ser servido é o que se encontra na posição da frente. Quem acaba de chegar ocupa o lugar na parte de trás da fila.



Um conjunto de operações, a utilizar na criação e manipulação de objectos ou entidades do tipo fila de espera, pode ser o que se indica.

#### Construtor

(cria-fila)

devolve uma fila vazia.

#### Selectores

(fila-vazia? fila)

devolve #t, se fila vazia, ou #f, no caso contrário.

(frente-da-fila fila)

devolve o primeiro elemento de fila, mas não a altera. Devolve mensagem de erro, se fila vazia.

#### Modificadores

(entra-na-fila! fila item)

insere item na parte de trás de fila e devolve o símbolo ok.

(sai-da-fila! fila)

retira o primeiro elemento de fila e devolve o símbolo ok.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Uma primeira hipótese a explorar, para modelar a fila de espera, será uma lista, cuja cabeça não fará parte da fila. Esta cabeça é apenas um símbolo que, no exemplo que se segue, é `fila:`, e que garante que uma fila vazia será representada por uma lista não vazia, o que pode facilitar a implementação de alguns modificadores.

Comece por imaginar a criação das filas de espera com a designação `cantina` e `taxi`.

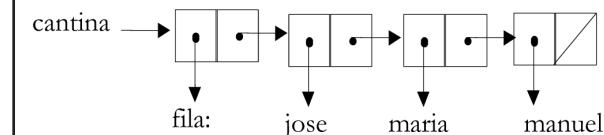
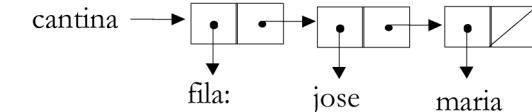
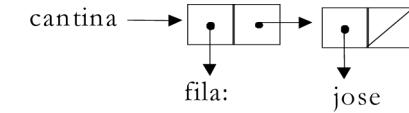
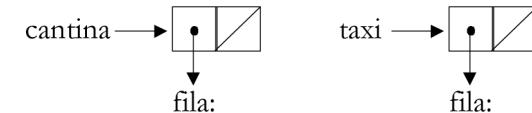
```
> (define cantina (cria-fila))
> cantina
(fila:)
> (define taxi (cria-fila))
> taxi
(fila:)
```

Quem entra vai para o fim da fila. É o que acontece em primeiro com `jose` e depois com `maria`, na fila `cantina`.

```
> (entra-na-fila! cantina 'jose)
ok
> cantina
(fila: jose)
> (entra-na-fila! cantina 'maria)
ok
> cantina
(fila: jose maria)
```

Agora entra o `manuel` na fila `cantina`.

```
> (entra-na-fila! cantina 'manuel)
ok
> cantina
(fila: jose maria manuel)
```



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

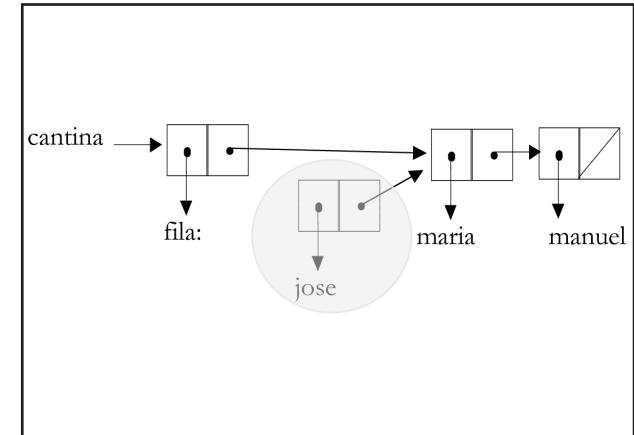
7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



Seguidamente, sai um elemento da fila cantina e sai quem está à frente na fila, ou seja, jose.

```
> (sai-da-fila! cantina)
ok
> cantina
(fila: maria manuel)
```



E agora, entra o elemento jose na fila taxi.

```
> (entra-na-fila! taxi 'jose)
ok
> taxi
(fila: jose)
```

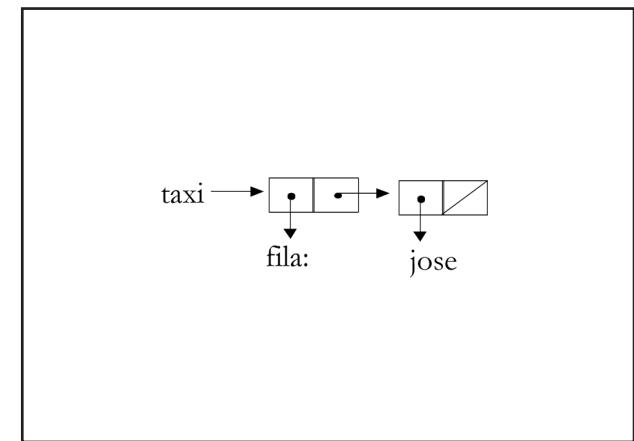
Está em condições para passar à implementação dos procedimentos para criação e manipulação de filas de espera.

```
(define cria-fila           ; construtor
  (lambda ()                 ; devolve uma fila vazia
    (list 'fila:)))

(define fila-vazia?         ; selector: verifica se fila vazia
  (lambda (fila)              ; nesta verificação, a cabeça da lista
    (null? (cdr fila))))     ; não é considerada como fazendo parte da fila

(define frente-da-fila       ; selector: devolve o primeiro elemento da fila
  (lambda (fila)              ; mas não o retira da fila
    (if (fila-vazia? fila)
        (display "erro: Fila vazia!...")
        (cadr fila)))))

(define entra-na-fila!       ; modificador: introduz um elemento, no final da
  (lambda (fila item)         ; fila, através de append!
    ... ... ... para completar
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
(define saí-da-fila!          ; modificador: retira o primeiro elemento da fila
  (lambda (fila)
    (if (fila-vazia? fila)
        (display "erro: Fila vazia!...")  

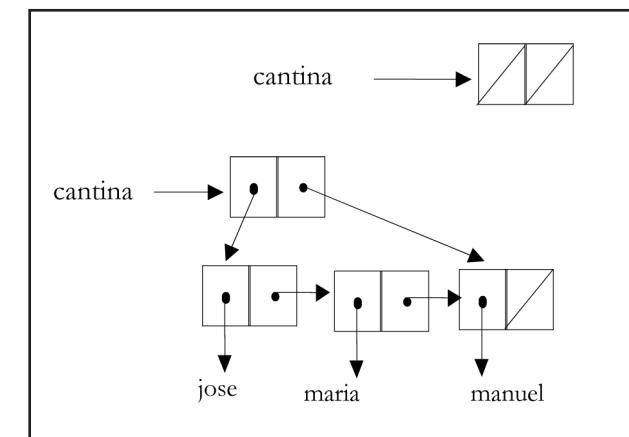
        (begin
          . . . . . para completar
```



Complete e teste a abstracção fila de espera.

O procedimento `entra-na-fila!` terá que aceder ao último par da lista que representa a fila de espera para lhe juntar um novo elemento. A não ser que o Scheme tivesse uma forma privilegiada de aceder ao último par de uma lista, a lista deverá ser percorrida do primeiro ao último elemento. Nesta situação, tratar-se-á de uma operação  $O(n)$ , pelo menos em relação ao tempo. Por outro lado, o acesso ao primeiro elemento da fila, como acontece nos procedimentos `sai-da-fila!` e `frente-da-fila`, é uma operação  $O(1)$ , pois estes procedimentos acedem sempre ao segundo elemento de uma lista, considerando que a cabeça, sendo o primeiro elemento da lista, não interessa para este efeito.

Para se conseguir um acesso  $O(1)$  também em relação ao último elemento, pode-se modelar a fila de espera como se fosse um par, em que o elemento da esquerda aponta para o primeiro elemento da fila e o elemento da direita para o último. Esta modelação disponibiliza um acesso  $O(1)$ , tanto para o primeiro como para o último elemento da fila. Neste caso, a fila é sempre vista como um par, mesmo quando está completamente vazia. Assim, não há necessidade de recorrer ao artifício de colocar um falso primeiro elemento, uma cabeça de lista, como se fez anteriormente, para garantir que uma fila vazia não surgisse como uma lista vazia.



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

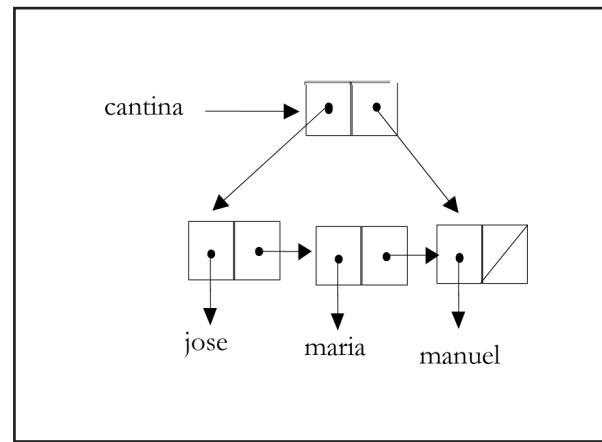
6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

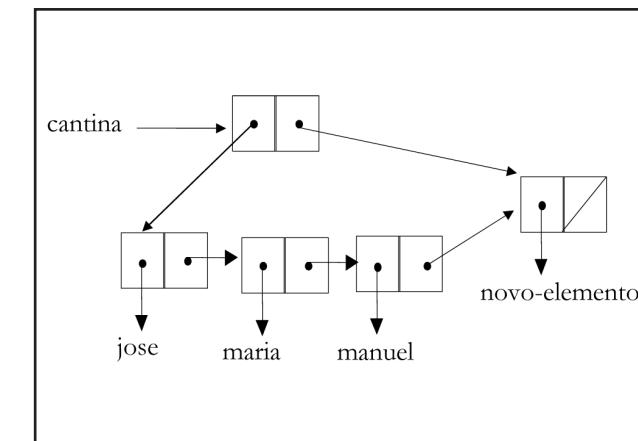
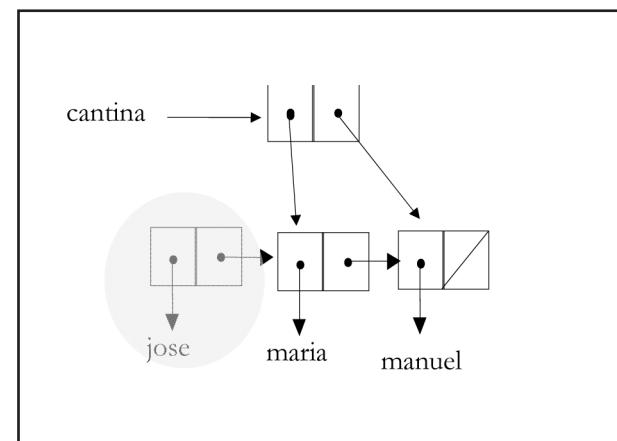
**Exercício 1** - é fundamental que faça e entenda muito bem este exercício...

Escreva os procedimentos para criação e manipulação de filas de espera modeladas como se fossem um par. Para facilitar, juntam-se algumas figuras de apoio.



Situação após a saída de um elemento

Situação após a entrada de um elemento





Desenvolva e teste a abstracção fila de espera vista como se fosse um par

Aproveite para acrescentar à abstracção o procedimento `visualiza-fila`, que recebe uma fila de espera e comporta-se da seguinte maneira:

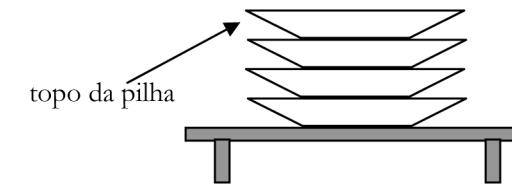
```
> (define fila-surpresa (cria-fila))
> (visualiza-fila fila-surpresa)
(fila: )
> (entra-na-fila! fila-surpresa 'jose)
ok
> (entra-na-fila! fila-surpresa 'maria)
ok
> (visualiza-fila fila-surpresa)
(fila: jose maria)
> (sai-da-fila! fila-surpresa)
ok
> (visualiza-fila fila-surpresa)
(fila: maria)
```

**No nosso quotidiano também surgem com alguma frequência estruturas do tipo pilha**

A pilha é uma abstracção de dados de grande utilidade, também reconhecida por LIFO – *Last In First Out*, em que o último elemento a chegar é o primeiro a sair.

Frequentemente, faz-se a analogia da pilha informática com a pilha de pratos que vai crescendo em cima da mesa da cozinha. Também nesta, o elemento mais facilmente acessível é o que se encontra no topo da pilha dos pratos, ou seja, o último prato que lá foi colocado.

Um novo elemento é colocado antes do elemento mais recente da pilha, o topo da pilha, passando ele a ser o topo da pilha.



O primeiro elemento a sair é o último elemento colocado na pilha, ou seja, o topo da pilha. Depois de sair um elemento, o seguinte passará a ser o novo topo da pilha, salvo se já não houver mais elementos, situação em que a pilha fica vazia.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Uma possível abstracção pilha pode ser definida com a funcionalidade que seguidamente se indica.

#### Construtor

(cria-pilha)

devolve uma pilha vazia.

#### Selectores

(pilha-vazia? pilha)

se pilha vazia devolve #t, se não devolve #f.

(topo-da-pilha pilha)

devolve o elemento do topo de pilha, mantendo-a intacta ou devolve mensagem, se pilha vazia.

(mostra-pilha pilha)

visualiza o conteúdo de pilha.

#### Modificadores

(poe-na-pilha! pilha item)

insere item no topo de pilha e devolve o símbolo ok.

(tira-da-pilha! pilha)

tira o elemento do topo de pilha e devolve o símbolo ok, ou devolve mensagem, se pilha vazia.



**Não se surpreenda se encontrar na bibliografia da especialidade as designações push! e pop! para as operações poe-na-pilha! e tira-da-pilha!. Tem alguma explicação para isto?**

Para clarificar o funcionamento da pilha, segue-se um pequeno diálogo.

```
> (define pilha-1 (cria-pilha))  
> (mostra-pilha pilha-1)  
topo:  
> (poe-na-pilha! pilha-1 54)  
ok  
> (poe-na-pilha! pilha-1 35)  
ok
```

# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
> (mostra-pilha pilha-1)
topo: 35 54
> (topo-da-pilha pilha-1)
35
> (mostra-pilha pilha-1)
topo: 35 54
> (tira-da-pilha! pilha-1)
ok
> (mostra-pilha pilha-1)
topo: 54
> (pilha-vazia? pilha-1)
#f
> (tira-da-pilha! pilha-1)
ok
> (pilha-vazia? pilha-1)
#t
> (tira-da-pilha! pilha-1)
sem elementos!...
> (mostra-pilha pilha-1)
topo:
> (topo-da-pilha pilha-1)
sem elementos!...
> (poe-na-pilha! pilha-1 456)
ok
> (mostra-pilha pilha-1)
topo: 456
>
```

## Exercício 2

Desenvolva a abstracção pilha, de acordo com a especificação apresentada, sugerindo que a mesma seja estruturada em torno de uma lista mutável, cuja cabeça não fará parte da pilha. Esta cabeça é apenas um símbolo, por exemplo, topo:, que garante que uma pilha vazia será representada por uma lista não vazia, característica que poderá facilitar a implementação de alguns modificadores.



Como terá feito com as filas de espera, parece-lhe adequado implementar a pilha como um par de apontadores: um para o início e outro para o fim da pilha? Justifique.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



Desenvolva e teste a abstracção pilha.

Experimente o diálogo apresentado no apoio à especificação.

Sugestão: junte à abstracção pilha desenvolvida e testada um selector que forneça o comprimento da pilha.

```
> (define pilha-1 (cria-pilha))  
> (comprimento pilha-1)  
0  
> (poe-na-pilha! pilha-1 54)  
ok  
> (comprimento pilha-1)  
1
```

### A importância das tabelas é também enorme...

Para além dos conjuntos, filas de espera, pilhas e outras abstracções de dados, também as tabelas se apresentam como uma abstracção de grande utilidade. Perante várias abstracções, cada uma com características próprias, caberá a cada um escolher a que melhor se adequa ao problema que pretenda resolver ou, caso não encontre nenhuma interessante, criar uma nova abstracção. Nos conjuntos, os elementos são pura e simplesmente lançados no seu interior e, quando é necessário aceder a algum deles, a procura faz-se elemento a elemento. Nas filas de espera e nas pilhas os elementos já se encontram organizados, em geral, por ordem de chegada. Nas tabelas, os elementos constituem-se em registos, cada um deles com uma chave e os dados associados. É a partir da chave de um registo que se accede aos seus dados.

A tabela que se segue é composta por três registos.

chave	dados associados
jose	10
maria	273
antonio	31



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Pode começar por imaginar esta tabela representada por uma lista de registos, cada um deles implementado através de um par, em que o elemento da esquerda é a chave e o elemento da direita é o dado associado à chave. Para evitar que a lista, na ausência de registo, se apresente vazia, inclui-se um primeiro elemento fixo, a cabeça da lista, que não faz parte dos elementos da tabela.

Para completar a abstracção tabela consideram-se os seguintes procedimentos básicos:

Construtor

(cria-tabela)

devolve uma tabela vazia.

Selector

(procura-tabela chave tabela)

se chave existir em tabela, devolve o valor associado a chave. Caso contrário, devolve #f.

Modificadores

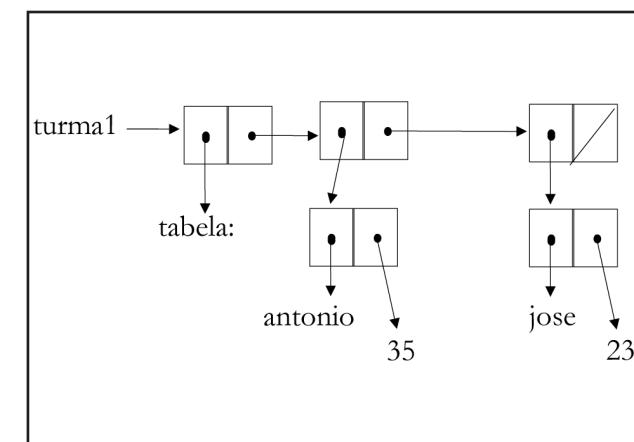
(insere-na-tabela! chave valor tabela)

se chave existir em tabela, actualiza o valor associado a chave com valor. Caso contrário, cria um novo registo com chave e valor e insere-o no início da tabela. Em ambos os casos, devolve o símbolo ok.

(retira-da-tabela! chave tabela)

se chave existir em tabela, retira de tabela o registo composto por chave e valor-associado. Caso contrário, deixa tabela intacta. Em ambos os casos, devolve o símbolo ok.

Observe uma sessão que ilustra a utilização dos procedimentos acabados de apresentar, para que fique claro o respectivo modo de funcionamento.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

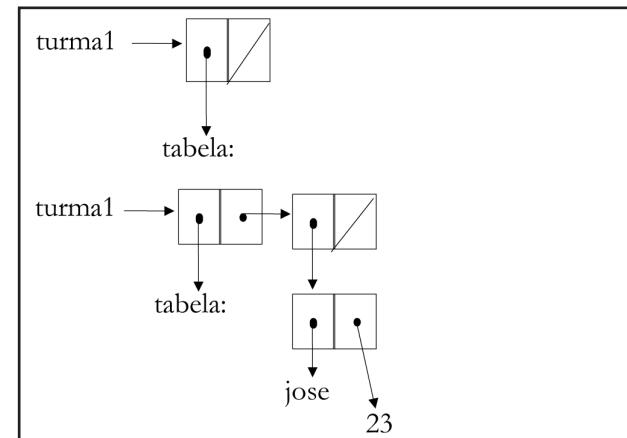
## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

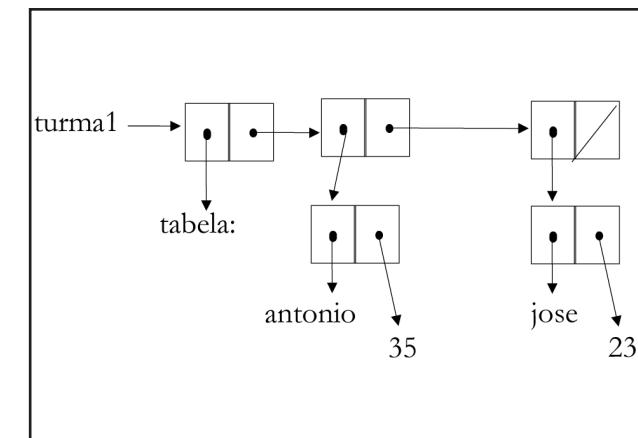
Em primeiro lugar, foi criada a tabela turma1 e, logo de seguida, inseriu-se um registo correspondente à chave jose com o valor 23.

```
> (define turma1 (cria-tabela))
> turma1
(tabela:)
> (insere-na-tabela! 'jose 23 turma1)
ok
> turma1
(tabela: (jose . 23))
```



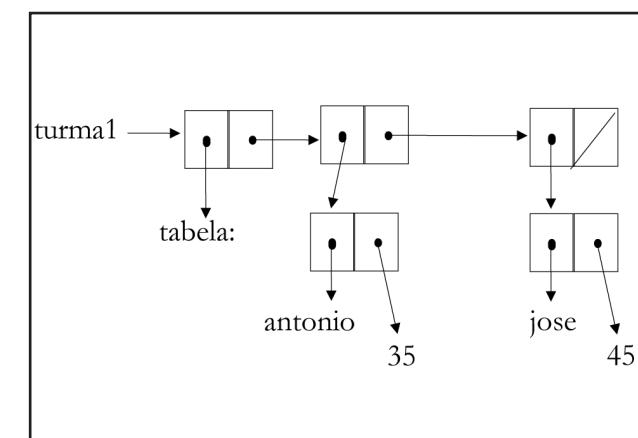
Agora, vai ser inserido um novo registo com a chave antonio e o valor 35.

```
> (insere-na-tabela! 'antonio 35 turma1)
ok
> turma1
(tabela: (antonio . 35) (jose . 23))
```



A mesma operação de inserção, ao encontrar na tabela uma chave com a mesma designação, limitou-se a alterar o valor que lhe estava associado.

```
> (insere-na-tabela! 'jose 45 turma1)
ok
> turma1
(tabela: (antonio . 35) (jose . 45))
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Analise a implementação dos procedimentos para criação e manipulação de tabelas.

```
; construtor: devolve uma tabela vazia
;
(define cria-tabela
  (lambda ()
    (list 'tabela:)))

; selector: procura uma chave numa tabela...
;
(define procura-tabela
  (lambda (chave tabela)
    (let ((registro (assoc chave (cdr tabela))))
      (if registro
          (cdr registro) ; se encontrar a chave, devolve o valor associado
          #f))) ; se não encontrar, devolve #f
```

O procedimento `assoc` é disponibilizado pelo Scheme ([Anexo A](#)). Este procedimento recebe uma chave e uma lista de registos e, se encontrar a chave num dos registos da lista, devolve (um apontador para) esse registo. Caso não encontre a chave, devolve `#f`.

```
> (assoc 'a '((b 556) (a 789) (a 4)))
(a 789)
> (assoc 'k '((b 5) (a 7) (a 4)))
#f
```



Complete e teste o procedimento `assoc-minha-versao`.

```
(define assoc-minha-versao
  (lambda (chave lista-de-registros)
    .... .... .... para completar
```



Agora, depois de conhecer bem o procedimento `assoc-minha-versao`, volte a analisar o selector `procura-tabela` e justifique o uso de `(cdr tabela)` como argumento de `assoc`.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

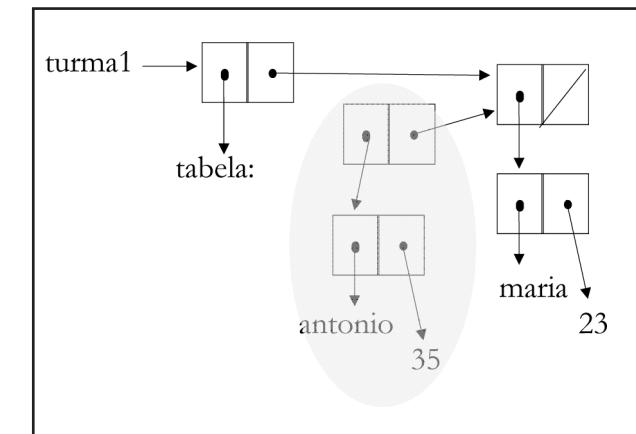
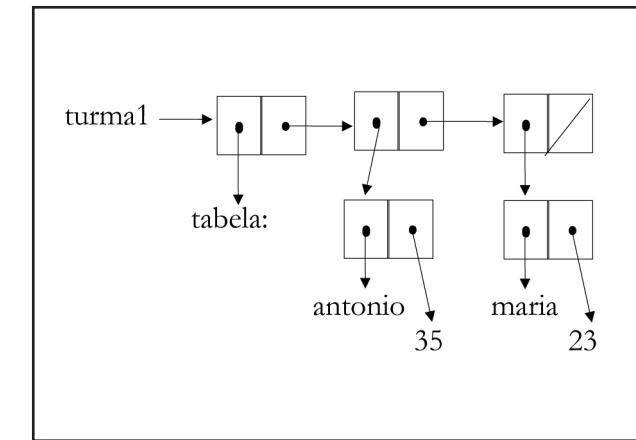
### ANEXOS

```
; modificador:  
;   se chave não existir, insere um novo registo na tabela,  
;   mas se existir, altera o valor associado à chave.  
;  
(define insere-na-tabela!  
  (lambda (chave valor tabela)  
    (let ((registro (assoc chave (cdr tabela))))  
      (if registro  
          ; se registo diferente de #f, mesmo que não  
          ; seja booleano, é considerado #t  
          (set-cdr! registro valor)  
          (set-cdr! tabela  
            (cons (cons chave valor)  
                  (cdr tabela))))  
          'ok)))
```

### Exercício 3

Escreva em Scheme o procedimento `retira-da-tabela!` conforme especificação já apresentada.

```
> turma1  
(tabela: (antonio . 35) (maria . 23))  
  
> (retira-da-tabela! 'antonio turma1)  
ok  
> turma1  
(tabela: (maria . 23))  
  
> (retira-da-tabela! 'jose turma1)  
ok  
> turma1  
(tabela: (maria . 23))
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Antes de começar a completar a abstracção tabela, reserve algum tempo para pensar numa forma de resolver o modificador **retira-da-tabela!**.

O elemento a retirar pode ser o primeiro, o último, um dos elementos intermédios ou até nem existir.  
Trata-se, portanto, de um problema que requer algum cuidado...



Complete e teste a abstracção tabela.

A abstracção tabela, em que se acede aos elementos através de uma chave, é designada por tabela unidimensional. Quando se acede aos elementos da tabela através de duas chaves, estamos perante a tabela bidimensional. No exemplo que se segue, as chaves associadas a cada um dos estudantes são o departamento e o nome do estudante.

	electro	civil
jose	10	carlos 137
maria	273	maria 300
antonio	31	

Com as chaves `electro` e `maria` acede a 273, mas com `civil` e `maria` o valor que alcança é 300.



Verifique, na figura seguinte, que a tabela bidimensional pode ser vista como uma lista de tabelas unidimensionais. As sub-tabelas `electro` e `civil` não apresentarão uma estrutura de uma tabela unidimensional?



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

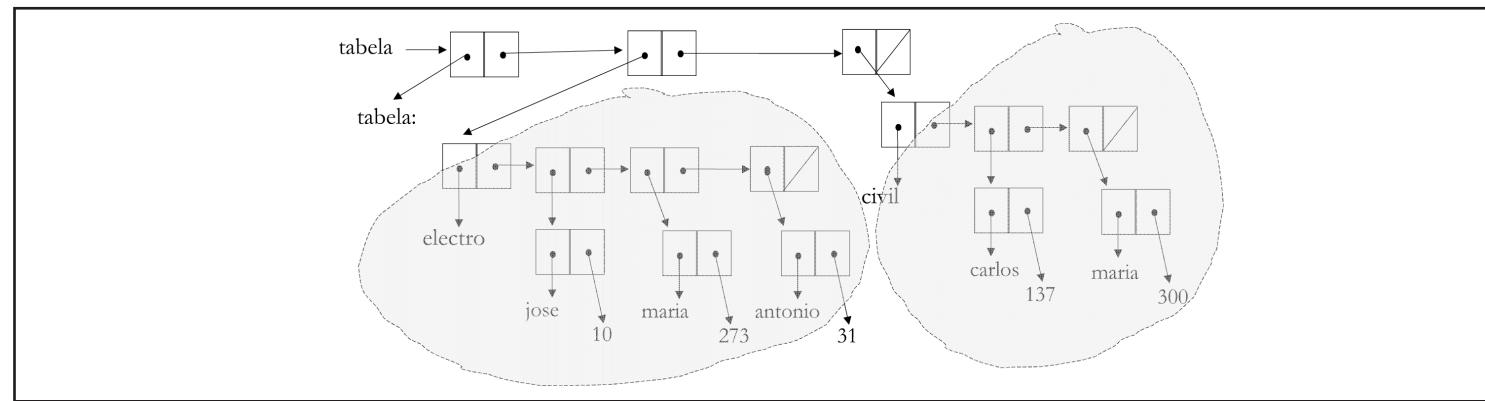
### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



**Exercício 4** - faça este exercício para alcançar um domínio completo dos dados mutáveis

Tomando por base a definição e os procedimentos apresentados para a abstracção tabela unidimensional, defina e desenvolva a abstracção tabela bidimensional.

Teste cuidadosamente a solução encontrada e tenha em conta que está perante um problema com um grau de dificuldade relativamente elevado.



Defina, desenvolva e teste a abstracção tabela bidimensional.

**Guardar resultados para não voltar a calcular** - a tabulação

Guardar resultados de cálculos, para os reutilizar mais tarde, pode fazer muito sentido, sobretudo quando esses cálculos gastam grande quantidade de recursos (tempo e espaço) e são grandes as hipóteses desses cálculos serem repetidos. A esta técnica de guardar resultados, por exemplo, numa tabela local, dá-se o nome de tabulação (*memoization* ou *tabulation*).

O princípio de funcionamento é simples. Se um procedimento é chamado para calcular um valor, consulta-se em primeiro lugar a tabela local, pois o valor poderá já estar na tabela. Se assim não for, o valor é calculado e, antes de ser devolvido, é também colocado na tabela. Da próxima vez já não será necessário voltar a calculá-lo...

# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

A ilustração desta técnica vai ser feita com o problema da geração da sequência de Fibonacci, na qual cada elemento é igual à soma dos dois elementos anteriores, sendo os dois primeiros, por convenção, 0 e 1. Os primeiros elementos são 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...e a sequência pode ser gerada através de uma definição recursiva.

$$fib(n) = \begin{cases} n & \text{se } n < 2 \\ fib(n-1) + fib(n-2) & \text{nos outros casos} \end{cases}$$

Desta definição retira-se o procedimento respectivo, que apresenta duas chamadas a si próprio, no passo recursivo.

```
(define fib
  (lambda (n)
    (if (< n 2)
        n ; caso base
        (+ (fib (- n 1)) ; caso geral ou passo recursivo
            (fib (- n 2))))))

> (fib 5)
5
> (fib 400)
176023680645013966468226945392411250770384383304492191886725992896575345044216019675
```



É melhor não experimentar calcular (fib 400).  
Provavelmente, nem terá paciência para calcular (fib 40). Quer experimentar?  
No meu pequeno computador, desisti ao fim de 2 minutos...



Teste o procedimento fib.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Posso garantir que o resultado de (fib 400) está correcto, mas não foi calculado com o procedimento fib, pois este tem um comportamento  $O(q^n)$ , com  $q \approx 1.6180$  (*golden ratio*) em relação ao tempo e  $O(n)$  em relação ao espaço. O mau comportamento de fib, em relação ao tempo é fácil de entender. A justificação está nas permanentes repetições de cálculos. Basta verificar, por exemplo, que (fib 5) requer (fib 4) e (fib 3), mas depois (fib 4) vai requerer (fib 3) e (fib 2). Vê imediatamente que, só aqui, (fib 3) é calculado 2 vezes.

Numa outra ocasião em que este assunto foi tratado, desenvolvemos uma solução iterativa, **fib-iter**, com um comportamento muito melhor,  $O(n)$  em relação ao tempo e  $O(1)$  em relação ao espaço.

A ideia em que se baseou fib-iter foi manter dois acumuladores, aos quais se associavam, em cada iteração, dois elementos seguidos da sequência de Fibonacci, o elemento corrente (ac-corrente) e o seguinte (ac-seguinte). São inicializados com ac-corrente = fib (0) = 0 e ac-seguinte = fib (1) = 1.

Em cada iteração, desenrolar-se-á, simultaneamente, o seguinte:

- ac-seguinte + ac-corrente vai para ac-seguinte
- ac-seguinte vai para ac-corrente

```
(define fib-iter
  (lambda (contador ac-corrente ac-seguinte)
    (if (zero? contador)
        ac-corrente
        (fib-iter (sub1 contador)
                  ac-seguinte
                  (+ ac-seguinte ac-corrente)))))

(define fib-novo
  (lambda (n)
    (fib-iter n 0 1)))

> (fib-novo 4000)
399094734350044227920812480949609126007925709828202578526288763265230518186413734335491367694241324
422939693065375201182738796280254432353703622509554356541715928979667908648144582231419142725908974
684721803706396953344496626503128747355609262982462494041683090642143510444590777494252367776608092
260951518520527813529754494825658383698091837717874396608251405028243431319117112963924571388674865
939235441778937354286022382122491565646314525076586034000120036853229848384889623514926325777553544
529040492412945656625194172350200498738738786027313792078932123354234848734690830545563298941672628
186925998152095825172779650590682355431394593750282768512214358159573742731438244229094163953751787
392685443681268942409791353221760803747809980106577107756258560415940784954117242365602425977591855
43824798332467919613598667003025993715274875
>
```





Teste o procedimento fib-novo.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



A resposta a (fib-novo 4000) é quase instantânea, mas com (fib-novo 100000) já vai notar uma espera significativa.

Com a técnica de tabulação não é muito diferente e a resposta a (fib-tabelado 100000) é também demorada, mas essa espera só acontecerá da primeira vez. Depois a resposta torna-se instantânea! Acredita?



**Depois da chamada (fib-tabelado 100000), qualquer chamada com argumento igual ou inferior a 100000 é de resposta praticamente instantânea. Justifique.**

Antes de apresentar o procedimento fib-tabelado, convém perceber como é possível associar uma estrutura de dados local a um procedimento, estrutura essa que não morra depois de terminada qualquer chamada do procedimento. Ou seja, a estrutura de dados é criada e inicializada na altura em que o procedimento é criado e pode ser lida ou actualizada sempre que o procedimento é chamado.



**Para a técnica de tabulação precisará certamente de uma solução deste tipo para associar uma tabela local a um procedimento...**

**Concorda? Justifique.**

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

**Exemplo 1** - estrutura de dados associada ao procedimento, que se mantém para além das chamadas

Neste exemplo, o procedimento novo-contador, quando é chamado sem argumentos, devolve um procedimento com um parâmetro, designado por valor. Associado ao procedimento devolvido vai a variável local conta, com o valor zero.

```
(define novo-contador
  (lambda ()
    (let ((conta 0))
      ;
      (lambda (valor)
        (set! conta (+ conta valor))
        conta))))
```

Segue-se a criação de dois contadores novos, que são dois procedimentos, um com a designação conta-a e outro conta-b.

```
> (define conta-a (novo-contador))
> (define conta-b (novo-contador))
```



Analise bem os procedimentos associados aos identificadores conta-a e conta-b e diga como respondem a:

```
> (conta-a 1)
??
> (conta-b 0)
??
> (conta-a (conta-b 4))
??
> (conta-b (conta-b 0))
??
```

Se não conseguiu as respostas, analise o diálogo que se segue...

```
> (conta-a 1)
1
> (conta-a 0)
1
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (conta-b 0)
0
> (conta-a 50)
51
> (conta-a (conta-a 0))
102
> (conta-b 45)
45
> (conta-b (conta-a 0))
147
> (conta-a 0)
102
> (conta-b 0)
147
>
```



Teste o procedimento novo-contador, criando contadores e actuando sobre eles.

### Exercício 5

No contexto do que acaba de ser apresentado, complete o procedimento novo-contador-com-fun, que cria contadores com uma função e um valor inicial associados. Por exemplo, se a função for o operador +, em cada chamada, o contador é actualizado somando o argumento da chamada ao valor actual do contador. Mas se a função for o operador \*, em cada chamada, o contador é actualizado multiplicando o argumento da chamada pelo valor actual do contador.

```
> (define conta-soma (novo-contador-com-func + 100))
> (define conta-subt (novo-contador-com-func - 100))
> (define conta-mult (novo-contador-com-func * 100))
> (define conta-divi (novo-contador-com-func / 100.0))
> (conta-soma 0)
100
> (conta-subt 0)
100
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (conta-mult 1)  
100  
> (conta-divi 1)  
100.0  
> (conta-soma 3)  
103  
> (conta-subt 3)  
97  
> (conta-mult 3)  
300  
> (conta-divi 3)  
33.33333333333336  
>  
  
(define novo-contador-com-func  
  (lambda (func valor-inicial)  
    (let ((conta valor-inicial))  
      ;  
      (lambda (valor)  
        ... .... para completar  
        conta))))
```



Complete e teste o procedimento `novo-contador`, criando contadores e actuando sobre eles.

### Exemplo 2 - técnica de tabulação

Chegou o momento de atacar o problema da geração da sequência de Fibonacci, com uma solução baseada na técnica de tabulação. A ideia a explorar passa pelo desenvolvimento de um procedimento, `memoize`, com um parâmetro único que é um procedimento. O procedimento `memoize` devolve um procedimento a que associa uma lista local, inicialmente vazia, e sobre a qual será implementada uma tabela. Neste caso, a chave de cada registo da tabela é um inteiro positivo e o dado associado é o respectivo valor da sequência de Fibonacci.

```
(define memoize  
  (lambda (proc)  
    (let ((table '()))  
      ; ---- procedimento devolvido por memoize ----
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(lambda (arg)
  (let ((pair-in-table (assoc arg table)))
    (if pair-in-table
        (cdr pair-in-table) ; arg já existe na tabela...
        (let ((val (proc arg))) ; arg ainda não existe na tabela...
            (set! table (cons (cons arg val) table))
            val))))))
```



Explique o modo de funcionamento do procedimento devolvido por `memoize`.

Pista: não esqueça que esse procedimento trabalhará sobre uma tabela que vai crescendo...



O procedimento devolvido por `memoize` não utilizou directamente a abstracção tabela, mas acaba por utilizar mecanismos nela existentes, nomeadamente para procurar e inserir novos registos na tabela.

Aponte uma possível razão para este facto.

```
; alternativa ao procedimento fib, usando agora a técnica de tabulação
(define fib-tabelado
  (memoize (lambda (n)
    (if (< n 2)
        n
        (+ (fib-tabelado (- n 1))
           (fib-tabelado (- n 2)))))))
```



Ao identificador `fib-tabelado` é associado o procedimento devolvido por `memoize`, o procedimento com o parâmetro `arg`...

Ou seja, uma chamada de `fib-tabelado` é equivalente a uma chamada do procedimento devolvido por `memoize`, com consultas e eventuais inserções na tabela a ele associada. Concorda?



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Teste o procedimento fib-tabelado.

Tente imaginar outras situações que justifiquem a utilização da técnica de tabulação.

### Exercício 6

Aplique a técnica de tabulação a um procedimento para calcular o n-ésimo número na sequência de *Mewman-Conway* definida por:

$$P(n) = \begin{cases} 1 & \text{se } n = 1 \text{ ou } n = 2 \\ P(P(n-1)) + P(n - P(n-1)) & \text{se } n > 2 \end{cases}$$



**Calcule, manualmente, os primeiros elementos desta sequência.**



Desenvolva e teste o procedimento pedido.

Experimente aumentar o argumento e tente interpretar o comportamento do procedimento especialmente em termos do tempo de resposta.

Recorra a uma solução que não use a tabulação e compare os resultados em termos de tempo de resposta, sobretudo para argumentos de valor elevado.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



## Exercício 7

Aplique a técnica de tabulação a um procedimento para calcular a função de Ackermann, sabendo que esta é definida para inteiros não negativos, recorrendo a

$$A(m, n) = \begin{cases} n + 1 & \text{se } m = 0 \\ A(m - 1, 1) & \text{se } m > 0 \text{ e } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{se } m > 0 \text{ e } n > 0 \end{cases}$$



Calcule, manualmente, os primeiros elementos da função.

Em relação à aplicação da tabulação, acha que irá encontrar alguma dificuldade adicional em relação aos outros casos tratados? Justifique.



Desenvolva e teste o procedimento pedido.

Experimente aumentar os argumentos, primeiro segundo  $m$  mantendo  $n$  constante e depois fazendo o contrário.

Recorra a uma solução que não use a tabulação e compare os resultados em termos de tempo de resposta, sobretudo para argumentos de valor elevado.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



## Módulo 5.3 - Vectores são dados mutáveis do Scheme

Com os modificadores que o Scheme disponibiliza (`set!`, `set-car!`, `set-cdr!` e `append!`) é possível criar abstracções de dados mutáveis adequadas a situações que delas necessitem. Algumas destas abstracções foram já definidas e implementadas, nomeadamente filas de espera, pilhas e tabelas. Outras são, logo à partida, disponibilizadas pelo Scheme e nelas se incluem os vectores que serão agora considerados.

### Palavras-Chave

Vector, acesso aleatório (*random access*), índice.

# **S C H E M E**

## **na descoberta da programação**

**INTRODUÇÃO**

**1 - O ESSENCIAL DO SCHEME**

**2 - RECURSIVIDADE**

**3 - ABSTRACÇÃO DE DADOS**

**4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE**

**5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS**

R E 1 2 3 4 5

**6 - SCHEME E OUTRAS  
TECNOLOGIAS**

**7-EXERCÍCIOS  
E  
PROJECTOS**

**ANEXOS**



### **Mas afinal, o que é que os vectores trazem de novo?**

Em primeiro lugar, convém que perceba o que é que a abstracção vector traz de novo, relativamente às abstracções já consideradas.

Nos conjuntos, os seus elementos são pura e simplesmente lançados no seu interior e, quando é necessário aceder a algum deles, a procura faz-se elemento a elemento, pois não há, normalmente, uma ordem estabelecida entre eles. Nas filas de espera e nas pilhas os seus elementos já se encontram organizados, normalmente, por ordem de chegada. Nas tabelas, os seus elementos constituem-se em registo, associando cada um deles um valor e uma ou mais chaves, a partir das quais se accede ao valor.

Relembrando o que foi feito, todas estas abstracções se baseavam em listas. Os conjuntos utilizaram listas não mutáveis, mas agora até poderiam ser implementados com listas mutáveis, como aconteceu com as filas de espera e com as tabelas.



**Em que tipo de situação escolheria os conjuntos implementados com listas mutáveis em vez de conjuntos implementados com listas? Justifique.**

Em todas estas abstracções, o acesso aos seus elementos é sequencial, começando no primeiro e avançando na direcção do último. No caso especial da fila de espera em que, fundamentalmente, só interessa acceder ao primeiro e ao último elementos, foi possível utilizar uma solução que minimizava o problema do acesso sequencial, estruturando a fila como se fosse um par, em que o elemento da esquerda apontava para a parte da frente da fila e o elemento da direita para a parte de trás.

A abstracção de dados mutáveis, designada por vector, apresenta uma característica nova que é a possibilidade de acesso aleatório (*random access*) a qualquer um dos seus elementos.

# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

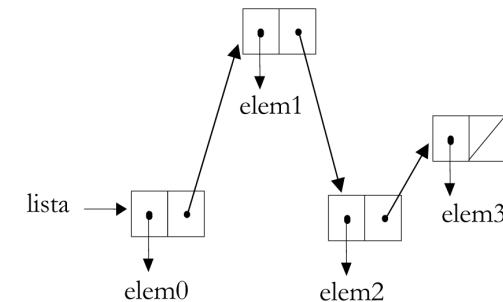
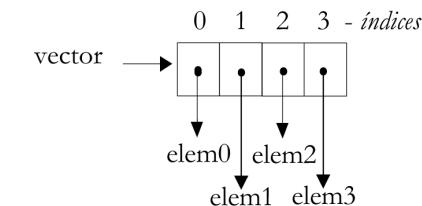


A cada elemento é associado um índice e, através desse índice, o vector dá acesso a esse elemento sem obrigar a passar por qualquer um dos outros. Assim não acontece com as listas, que obrigam a percorrer a sequência de elementos desde o primeiro até ao elemento pretendido, claramente um acesso sequencial.

Por exemplo, se uma lista `ls` contiver 1000 elementos, o acesso ao elemento que se encontra na posição 900, poderá ser conseguido fazendo (`(list-ref ls 900)`). Pode imaginar que esta operação implica 900 `cdr`, seguidos de um `car`.

Se os mesmos 1000 elementos constituíssem um vector `vec`, o elemento colocado na posição 900 seria accedido directamente, fazendo (`(vector-ref vec 900)`), sem ter que passar pelos elementos que o antecedem.

Está perante uma estrutura de dados que se recomenda em situações em que ocorre com grande frequência o acesso não sequencial aos seus elementos, para uma simples leitura ou até mesmo para uma modificação.



**Seria muito interessante se conseguisse identificar pelo menos uma situação em que é conveniente utilizar uma estrutura de dados com acesso aleatório.**

A abstracção vector disponibiliza modificadores, para além de construtores e selectores. Os exemplos que se seguem ilustram a funcionalidade associada a esta abstracção e qualquer dúvida pode ser certamente ultrapassada pela consulta do [Anexo A](#), na parte relativa aos vectores.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Comece pela criação de um vector, enumerando os seus elementos. Para este efeito utilize o construtor vector.

O predicado vector? confirma a criação de um novo vector.

```
> (define v1 (vector 'a 6 'ab 90))  
> (vector? v1)  
#t  
> v1  
#(a 6 ab 90)
```



Não se espante, pois no DrScheme a visualização de vectores indica também o comprimento, logo a seguir ao carácter #, ou seja, o exemplo referido seria visualizado #(a 6 ab 90).

O modificador vector-fill! preenche todas as posições de um vector existente com um mesmo valor, não estando especificado o valor que devolve.

```
> (vector-fill! v1 'abc)  
> v1  
#(abc abc abc abc)
```



Em vez de #(abc abc abc abc), o DrScheme apenas visualizaria #(abc), pois os últimos elementos, quando iguais, são visualizados apenas uma vez.

Por exemplo, o vector #(1 2 3 3) seria visualizado #(1 2 3).

E se fosse o vector #(1 2 2 3)?

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Um novo vector, v2, é criado e determina-se o seu comprimento com `vector-length`. O acesso a um elemento de um vector é conseguido através de `vector-ref`, indicando o índice desse elemento.

```
> (define v2 (vector 'b 8 'ef))  
> v2  
#(b 8 ef)  
> (vector-length v2)  
3  
> (vector-ref v2 1)  
8  
> (vector-ref v2 2)  
ef
```

A criação de um vector também é possível com o construtor `make-vector`, que surge com duas variantes, um ou dois argumentos, como se pode ver de seguida.

```
> (define v3 (make-vector 5))  
> v3  
#(0 0 0 0 0)  
> (define v4 (make-vector 4 'klm))  
> v4  
#(klm klm klm klm)
```



**Na criação de um novo vector...**

- 1- indique pelo menos uma situação em que usará `make-vector` em vez de `vector`.
- 2- e, agora ao contrário, indique pelo menos uma situação em que usará `vector` em vez de `make-vector`.

Esta é mais uma maneira de criar um vector, agora através dos elementos de uma lista. A operação inversa, a criação de uma lista a partir dos elementos de um vector, também é possível.

```
> (define v5 (list->vector '(1 2 a 7)))  
> v5  
#(1 2 a 7)  
> (define lista-1 (vector->list '#(1 3 4 5)))  
> lista-1  
(1 3 4 5)
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Finalmente, a modificação de um elemento de um vector através do modificador `vector-set!`..

```
> (define v6 (vector 0 2 4 8))  
> v6  
#(0 2 4 8)  
> (vector-set! v6 2 'aaa)  
> v6  
#(0 2 aaa 8)
```



Teste a funcionalidade associada aos vectores.

**Exemplo 1** - para familiarização com a funcionalidade mais importante da abstracção `vector`

Um programa simula vários lançamentos de um dado e faz a estatística respectiva. O utilizador do programa é interrogado sobre o número de lançamentos que pretende simular. Se este número for superior a zero, passa à simulação de um número equivalente de lançamentos, sendo anotadas as ocorrências de cada uma das suas 6 faces.

No final, são visualizadas as frequências de ocorrência dessas faces.

```
> (estatistica-dos-lancamentos)  
Quantos lançamentos (se < 1, termina)?  
200  
Resultados:  
1- 40  
2- 35  
3- 33  
4- 33  
5- 35  
6- 24  
  
Quantos lançamentos (se < 1, termina)?  
2000  
Resultados:  
1- 346  
2- 339
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

- 3- 327
- 4- 329
- 5- 339
- 6- 320

Quantos lançamentos (se < 1, termina) ?

0

Acabou...

A primeira decisão que se tomou refere-se à forma de representar computacionalmente as ocorrências das faces do dado, durante a simulação dos vários lançamentos. Optou-se por um vector de 6 posições, inicialmente todas elas com o valor zero, e à medida que a simulação progride, as ocorrências das faces vão sendo contadas e registadas no citado vector. As ocorrências da face 1 são contabilizadas na posição 0, as da face 2 na posição 1, ..., e as da face 6 na posição 5.



**Indique duas vantagens do vector sobre a lista mutável, neste caso da estatística dos resultados do lançamento do dado.**

O programa `estatistica-dos-lancamentos` pode ser decomposto em várias tarefas.

- Pergunta inicial sobre o número de lançamentos, incluindo a condição de finalização;
- Leitura do número de lançamentos;
- Sobre o número de lançamentos:
  - Se se verificar a condição de finalização, visualização da mensagem respectiva, ou seja, `Acabou...`;
  - Caso contrário, simulação dos lançamentos, visualização do resultado e novo lançamento de `estatistica-dos-lancamentos`.

Das tarefas do programa `estatistica-dos-lancamentos`, a mais complexa e que exige um cuidado especial tem a ver com a simulação dos lançamentos do dado, realizada pelo procedimento `lancar-dado`.



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1ª CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
(define estatistica-dos-lancamentos
  (lambda()
    (display "Quantos lançamentos (se < 1, termina)? ")
    (let ((numero-lancamentos (read)))
      (cond ((< numero-lancamentos 1)
             (newline)
             (display "Acabou..."))
            (else
              (let ((contador-faces (lancar-dado numero-lancamentos)))
                (newline)
                (display "Resultados:")
                (newline)
                (visu-vector contador-faces)
                (estatistica-dos-lancamentos)))))))
```

No procedimento `lancar-dado` é definida a variável local `conta-faces`, um vector de seis posições, utilizado para ir acumulando o número de ocorrências das faces do dado, nos vários lançamentos simulados. Este procedimento apoia-se num procedimento local `aux` que executa os lançamentos e actualiza o vector, que será devolvido no final.

```
(define lancar-dado
  (lambda (num-vezes)
    (let ((conta-faces (make-vector 6 0)))
      (letrec ((aux
                (lambda (n-vezes)
                  (if (zero? n-vezes)
                      conta-faces
                      (let ((face-menos-1 (sub1 (roleta-1-6))))
                        (vector-set! conta-faces
                                      face-menos-1
                                      (add1 (vector-ref conta-faces
                                                       face-menos-1)))
                        (aux (sub1 n-vezes)))))))
        ;
        (aux num-vezes)))))

(define roleta-1-6      ; gera e devolve um número aleatório entre 1 e 6
  (lambda()
    (add1 (random 6))))
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Experimente o procedimento `lancar-dado`.

No programa `estatistica-dos-lancamentos` ainda se distingue a visualização do resultado, mais especificamente a visualização do vector de seis posições, que representa as ocorrências das faces durante os lançamentos. Esta visualização baseia-se numa solução recursiva, em que a redução do problema no passo recursivo assume uma forma diferente da utilizada para as listas e que é característica dos vectores. Inicialmente determina-se o comprimento do vector a processar e define-se um procedimento local, neste caso designado por `aux`, com o parâmetro `indice`. A chamada inicial de `aux` apresenta-se com o argumento zero, que vai sendo incrementado em cada nova chamada. A condição de terminação corresponde a verificar-se a igualdade entre este argumento e o comprimento do vector, pois sendo este comprimento igual a 6, os índices válidos vão de 0 a 5.

```
(define visu-vector
  (lambda (vec)
    (let ((comprim (vector-length vec)))
      (letrec ((aux
                (lambda (indice)
                  (if (= indice comprim)
                      (newline)
                      (begin
                        (display (add1 indice))
                        (display "- ")
                        (display (vector-ref vec indice))
                        (newline)
                        (aux (add1 indice)))))))
        (aux 0)))))
```



Experimente, em primeiro lugar, o procedimento `visu-vector`, criando, para tal, alguns vectores. Seguidamente, experimente o programa `estatistica-dos-lancamentos`.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 1

Em relação ao programa `estatistica-dos-lancamentos`, anteriormente apresentado, como o índice da primeira posição dos vectores é 0, associou-se a posição genérica `i` do vector à face `i+1`. Há quem prefira, em situações análogas, usar um vector com mais uma posição do que o necessário, desprezar a posição de índice 0 e fazer a associação da posição genérica `i` à face `i`, que é muito mais legível. Para seguir esta via, introduza as alterações necessárias à solução apresentada anteriormente.



Desenvolva e teste a nova versão do programa `estatistica-dos-lancamentos`.

Agora que a abstracção vector começa a estar bem entendida, propomos que faça algumas medições do tempo de acesso a vectores e a listas. Aproveitaremos para lhe dar a conhecer o procedimento `time` do DrScheme que lhe permite determinar o tempo gasto no cálculo de uma expressão ([Anexo A](#)).



`(time exp)`

calcula `exp`, visualiza em `ms` o tempo de cpu gasto no cálculo, o tempo real e o tempo de recolha de lixo (*garbage collection*) e, finalmente, devolve o resultado do cálculo.

```
(define fib
  (lambda (n)
    (if (< n 2)
        n
        (+ (fib (- n 1))
            (fib (- n 2))))))

> (time (fib 31))
cpu time: 3922 real time: 3953 gc time: 0
1346269
>
```



Teste o procedimento `time`, com o procedimento `fib`.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



### Exemplo 2 - medição do tempo de acesso a vectores e a listas

Neste exemplo, são criados um vector e uma lista, ambos com 1000 elementos iguais a 0. O vector designado por `vector-1000` é criado com o construtor `make-vector` e a lista designada por `lista-1000` é criada com o procedimento `cria-lista`.

```
; cria vector-1000
(define vector-1000 (make-vector 1000 0))

; cria lista com um comprimento dado
; e com todos os elementos iguais a valor
;
(define cria-lista
  (lambda (comprimento valor)
    (letrec((aux
              (lambda (posicao)
                (if (= posicao comprimento)
                    ()
                    (cons valor
                          (aux (add1 posicao)))))))
      ;
      (aux 0)))))

; cria lista-1000
(define lista-1000 (cria-lista 1000 0))

; acede à posição pos de uma estrutura (vector ou lista)
; representada pelo parâmetro estrut.
; A função de acesso é representada pelo parâmetro func-acesso
; e será vector-ref ou list-ref, conforme a estrutura.
; O número de acessos a realizar é representado pelo parâmetro por n-vezes.
;
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(define acesso-estrutura
  (lambda (estrut func-acesso pos n-vezes)
    (letrec ((ciclo
              (lambda (vezes)
                (if (> vezes 0)
                    (begin
                      (func-acesso estrut pos)
                      (ciclo (sub1 vezes)))
                    'ok))))
      (ciclo n-vezes)))))
```



Primeiro para 10 mil e depois para 1 milhão de acessos à lista-1000, veja como evolui o tempo gasto em função da posição em jogo.

```
> (time (acesso-estrutura lista-1000 list-ref 0 10000))
cpu time: 16 real time: 16 gc time: 0
ok
> (time (acesso-estrutura lista-1000 list-ref 500 10000))
cpu time: 63 real time: 63 gc time: 0
ok
> (time (acesso-estrutura lista-1000 list-ref 999 10000))
cpu time: 125 real time: 125 gc time: 0
ok
> (time (acesso-estrutura lista-1000 list-ref 0 1000000))
cpu time: 438 real time: 438 gc time: 0
ok
> (time (acesso-estrutura lista-1000 list-ref 500 1000000))
cpu time: 3078 real time: 3079 gc time: 0
ok
> (time (acesso-estrutura lista-1000 list-ref 999 1000000))
cpu time: 5812 real time: 5845 gc time: 0
ok
```



E agora para o vector-1000...



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
> (time (acesso-estrutura vector-1000 vector-ref 0 10000))
cpu time: 15 real time: 16 gc time: 0
ok
> (time (acesso-estrutura vector-1000 vector-ref 500 10000))
cpu time: 0 real time: 0 gc time: 0
ok
> (time (acesso-estrutura vector-1000 vector-ref 999 10000))
cpu time: 16 real time: 15 gc time: 0
ok
> (time (acesso-estrutura vector-1000 vector-ref 0 1000000))
cpu time: 406 real time: 406 gc time: 0
ok
> (time (acesso-estrutura vector-1000 vector-ref 500 1000000))
cpu time: 469 real time: 468 gc time: 0
ok
> (time (acesso-estrutura vector-1000 vector-ref 999 1000000))
cpu time: 359 real time: 360 gc time: 0
ok
>
```



Experimente o procedimento acesso-estrutura e meça tempos de acesso a listas e a vectores.



O desempenho das listas acompanha o dos vectores para acessos às suas posições iniciais. Justifique.

**Exemplo 3** - criação de vectores com valores definidos por uma função

Para criar um vector de comprimento 5, com os 5 primeiros pares, podemos escrever:

```
> (vector 0 2 4 6 8)
#(0 2 4 6 8)
```

Para criar um vector de comprimento 40, com os primeiros 40 pares, a tarefa já se complica um pouco mais.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (vector 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62  
64 66 68 70 72 74 76 78)  
#(0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68  
70 72 74 76 78)
```

Deve reconhecer que, em situações como esta última, será de tentar uma forma diferente para criar vectores. Assim, a ideia a explorar passa pela definição de um procedimento, designado por `gera-vector-com-proc`, que recebe como argumentos um procedimento, `proc`, e um valor inteiro, `comp`, e devolve um vector de comprimento `comp`. O elemento genérico `i` do vector gerado é igual ao valor de `proc` em `i`.

Nos exemplos que se seguem, criam-se vários vectores, os dois primeiros com 5 e 40 inteiros pares, seguidos de outro com 15 potências de 2 e o último com uma sequência de inteiros a começar em 1.

```
> (gera-vector-com-proc (lambda (i) (* 2 i)) 5)  
#(0 2 4 6 8)  
> (gera-vector-com-proc (lambda (i) (* 2 i)) 40)  
#(0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66  
68 70 72 74 76 78)  
> (gera-vector-com-proc (lambda (i) (* i i)) 15)  
#(0 1 4 9 16 25 36 49 64 81 100 121 144 169 196)  
> (gera-vector-com-proc add1 40)  
#(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35  
36 37 38 39 40)
```

Analise a solução que se apresenta para o procedimento `gera-vector-com-proc`, sobretudo no que respeita à forma característica como se implementa a recursividade com vectores, como se mostrou anteriormente ao desenvolver o programa `estatista-dos-lancamentos`.

```
(define gera-vector-com-proc  
  (lambda (proc comprimento)  
    (letrec((vec (make-vector comprimento))  
           ;  
           (aux  
             (lambda (indice)  
               (if (= indice comprimento)  
                   vec  
                   (begin  
                     (vector-set! vec indice (proc indice))  
                     (aux (add1 indice)))))))  
           ;  
           (aux 0))))
```



INTRODUÇÃO
1 - O ESSENCIAL DO SCHEME
2 - RECURSIVIDADE
3 - ABSTRACÇÃO DE DADOS
4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE
5 - ABSTRACÇÕES COM DADOS MUTÁVEIS
R E 1 2 3 4 5
6 - SCHEME E OUTRAS TECNOLOGIAS
7-EXERCÍCIOS E PROJECTOS
ANEXOS



Experimente o procedimento gera-vector-com-proc.

## Exercício 2

Escreva o procedimento estica-vector que toma um vector de comprimento `comp` e devolve um novo vector de comprimento `novo-comp`, com `novo-comp > comp`, em que os `comp` primeiros elementos são os elementos do vector dado e os restantes elementos são iguais a 0.

```
> (estica-vector '#(a 1 b 2 c 3 d 4) 13)
#(a 1 b 2 c 3 d 4 0 0 0 0 0)
```



Desenvolva e teste o procedimento `estica-vector`.

## Exemplo 4 - processamento de dois vectores como um padrão de computação

Imagine uma situação em que se pretende processar simultaneamente dois vectores de igual comprimento. Os primeiros elementos dos dois vectores são processados, sendo registado o resultado, depois são processados os segundos elementos dos dois vectores, sendo também registado o resultado e assim sucessivamente até aos últimos elementos. No final, é devolvido um vector composto pelos resultados referidos.

Na interacção que se segue podemos ver três exemplos do processamento referido.

```
> (multiplica-2-vectores (vector 1 2 3) (vector 4 5 6))
#(4 10 18)
> (soma-2-vectores (vector 1 2 3) (vector 4 5 6))
#(5 7 9)
> (cons-2-vectores (vector 1 2 3) (vector 4 5 6))
#((1 . 4) (2 . 5) (3 . 6))
```



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

Estes três procedimentos apresentam um padrão semelhante, permitindo antever uma solução que passa por um procedimento de ordem mais elevada, como se indica.

```
(define soma-2-vectores
  (lambda (vec1 vec2)
    ((processa-2-vectores +) vec1 vec2)))

(define multiplica-2-vectores
  (lambda (vec1 vec2)
    ((processa-2-vectores *) vec1 vec2)))

(define cons-2-vectores
  (lambda (vec1 vec2)
    ((processa-2-vectores cons) vec1 vec2)))
```



Ainda se recorda do conceito **procedimento de ordem mais elevada**?  
Por que razão **processa-2-vectores** é um procedimento de ordem mais elevada?

O procedimento **processa-2-vectores** toma um procedimento como argumento e devolve um procedimento com dois parâmetros, que são dois vectores, sobre os quais aquele procedimento deve actuar.

```
(define processa-2-vectores
  (lambda (proc)

    (lambda (vec1 vec2)
      (let ((aux
              (lambda (indice)
                (proc (vector-ref vec1 indice)
                      (vector-ref vec2 indice)))))

        ; (gera-vector-com-proc aux (vector-length vec1))))))
```



Experimente o procedimento **processa-2-vectores**.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 3

Depois de estudar o exemplo anterior, concluirá certamente que se trata de uma solução muitíssimo engenhosa, que aproveita muito bem o procedimento `gera-vector-com-proc`. Nem sempre é fácil chegar a uma solução como esta... Como desafio, escreva uma solução alternativa para o procedimento `processa-2-vectores`, mas que não utilize o procedimento `gera-vector-com-proc`.



Desenvolva e teste uma nova versão do procedimento `processa-2-vectores`.

### Exercício 4

Escreva o procedimento `processa-n-vectores` que é uma generalização de `processa-2-vectores`, pois funciona com um ou mais vectores e não apenas com dois.



Desenvolva e teste o procedimento `processa-n-vectores`.



Tente identificar as analogias e diferenças do procedimento `processa-n-vectores` com `map` do Scheme, este utilizado com listas.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Módulo 5.4 - Cadeias de caracteres também são dados mutáveis do Scheme

Com os modificadores disponibilizados pelo Scheme (`set!`, `set-car!`, `set-cdr!` e `append!`) foram definidas e implementadas algumas abstracções, nomeadamente filas de espera, pilhas e tabelas. Outras são, logo à partida, disponibilizadas pelo Scheme e nelas se incluem as cadeias de caracteres que serão agora consideradas.

A propósito das cadeias de caracteres e da necessidade de guardar dados de uma sessão para outra sessão de trabalho, introduzem-se os ficheiros de texto.

### Palavras-Chave

Carácter, código ASCII, cadeia de caracteres (*string*), ficheiros de texto.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Não é fácil passar sem as cadeias de caracteres

Os caracteres representam mais um tipo de dados que o Scheme disponibiliza, para além dos outros já considerados, como sejam os números, booleanos, símbolos e vectores. É um carácter que se gera ao actuar uma tecla de um teclado, corresponda essa tecla a uma letra ou a um dígito numérico, a um carácter de pontuação ou de controlo. A cadeia de caracteres (*string*) é, numa primeira aproximação, um vector de caracteres e, como verá, os procedimentos primitivos associados aos vectores e às cadeias de caracteres apresentam grandes semelhanças. Com o procedimento `display`, as cadeias de caracteres têm sido largamente utilizadas na visualização de mensagens no ecrã.

A cada carácter está associado um valor numérico, que nem sempre foi o mesmo em todos os computadores, situação que originava enormes problemas quando se transferiam dados entre computadores que não adoptavam a mesma codificação. Para evitar tal confusão, a partir da década de 60, os fabricantes de computadores começaram a adoptar o código ASCII - *American Standard Code for Information Interchange* - que estabelece a codificação para todas as letras, dígitos numéricos, caracteres de pontuação e de controlo, num total de 128 caracteres. Uma tabela com o código ASCII é seguidamente apresentada.

A 1<sup>a</sup> e 2<sup>a</sup> colunas dessa tabela, ou seja, as duas colunas mais à esquerda, com os códigos de 00 a 31, correspondem aos caracteres de controlo como, por exemplo, *LineFeed* ou *Newline* (código 10), *Carriage Return* (código 13), enquanto as restantes colunas, com algumas excepções, estão essencialmente associadas aos caracteres de pontuação (3<sup>a</sup> coluna), aos dígitos decimais (4<sup>a</sup> coluna), às letras maiúsculas (5<sup>a</sup> e 6<sup>a</sup> colunas) e às letras minúsculas (7<sup>a</sup> e 8<sup>a</sup> colunas).

Tabela do Código ASCII

00 a 15	16 a 31	32 a 47	48 a 63	64 a 79	80 a 95	96 a 111	112 a 127
NUL	DLE	Space	0	@	P	‘	p
SOH	DC1	!	1	A	Q	a	q
STX	DC2	”	2	B	R	b	r
ETX	DC3	#	3	C	S	c	s
EOT	DC4	\$	4	D	T	d	t
ENQ	NAK	%	5	E	U	e	u
ACK	SYN	&	6	F	V	f	v
BEL	ETB	,	7	G	W	g	w
BS	CAN	(	8	H	X	h	x
HT	EM	)	9	I	Y	i	y
LF	SUB	*	:	J	Z	j	z
VT	ESC	+	;	K	[	k	{
FF	FS	,	<	L	\	l	
CR	GS	-	=	M	]	m	}
SO	RS	.	>	N	^	n	~
SI	US	/	?	O	_	o	DEL



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Para já, os caracteres vistos isoladamente

Em Scheme, cada carácter é representado pelo seu símbolo antecedido por `#\`, como, por exemplo, `#\a` para o carácter a, ou `#\5` para o carácter 5. O predicado `char?` reconhece os caracteres.

```
> (char? #\5)
#t
> (char? 5)
#f
```

O Scheme disponibiliza o procedimento `char->integer`, que devolve o código associado ao carácter que recebe como argumento.

```
> (char->integer #\5)
53
> (char->integer #\a)
97
> (char->integer #\A)
65
```

O Scheme aceita representações especiais para alguns caracteres de controlo como, por exemplo, `#\newline` e `#\space`.

```
> (char->integer #\newline)
10
> (char->integer #\space)
32
```

Por seu lado, `integer->char` realiza a operação inversa de `char->integer`.

```
> (integer->char 37)
#\%
> (integer->char 10)
#\newline
```

Para comparação do código dos caracteres, o Scheme oferece uma gama completa de predicados: `char=?`, `char>?`, `char<?`, `char>=?`, `char<=?`.

```
> (char=? #\a #\A)
#f
> (char>? #\a #\A)
#t
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (char<? #\9 #\1)
#f
> (char<? #\9 #\a)
#t
```

Quando não pretender distinguir as letras maiúsculas das minúsculas, os predicados deverão ser insensíveis a esta característica (*ci*, ou seja, *case-insensitive*), como acontece com `char-ci=?`, `char-ci>?`, `char-ci<?`, `char-ci>=?`, `char-ci<=?`.

```
> (char-ci=? #\a #\A)
#t
> (char-ci>? #\a #\A)
#f
> (char-ci>=? #\a #\A)
#t
```

No contexto das letras, ainda são de salientar os predicados `char-upper-case?` e `char-down-case?`, que determinam, respectivamente, se a letra é maiúscula ou minúscula e os procedimentos `char-upcase` e `char-downcase`, que devolvem, respectivamente, uma letra maiúscula ou minúscula.

```
> (char-upper-case? #\a)
#f
> (char-upper-case? #\A)
#t
> (char-upcase #\a)
#\A
> (char-upcase #\A)
#\A
> (char-upcase (char-downcase #\A))
#\A
```

Os procedimentos `char-upcase` e `char-downcase` devolvem um carácter igual ao que recebem, se este não for letra.

```
> (char-upcase #\5)
#\5
```



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

O Scheme também disponibiliza os predicados `char-alphabetic?`, `char-numeric?` e `char-whitespace?`.

O primeiro verifica se o argumento é um carácter alfabetico ou seja se é uma das letras, o segundo se é um dos dígitos decimais e o terceiro se é um carácter whitespace (whitespace inclui o space e o newline, mas algumas implementações do Scheme poderão considerar ainda outros caracteres).



Experimente a funcionalidade do Scheme associada aos caracteres.

Neste contexto, aproveite para escrever em Scheme versões pessoais para os predicados `char-alphabetic?`, `char-numeric?` e `char-whitespace?`, acabados de referir, que deverão responder como se indica.

```
> (pessoal-char-alpha? #\a)
#t
> (pessoal-char-alpha? #\4)
#f
> (pessoal-char-alpha? #\newline)
#f
> (pessoal-char-num? #\5)
#t
> (pessoal-char-num? #\newline)
#f
> (pessoal-char-whitesp? #\a)
#f
> (pessoal-char-whitesp? #\newline)
#t
```

### Os caracteres vistos não como elementos isolados, mas como cadeias de caracteres

Depois de observada e experimentada a forma como o Scheme trata os caracteres como elementos isolados, vai passar a ver como ele trata cadeias de caracteres.

Como já se referiu, a cadeia de caracteres é um vector de caracteres, o que até se evidencia fazendo o paralelismo entre os procedimentos que manipulam vectores e os que manipulam cadeias de caracteres.





Tente relembrar os procedimentos dos vectores equivalentes aos seguintes procedimentos para as cadeias de caracteres e, a partir deles, confirme as respostas dadas pelo Scheme na interacção que se segue.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
> (define c1 (string #\a #\6 #\b))  
> c1  
"a6b"  
> (string? c1)  
#t  
> (define c2 (string-copy c1))  
(Algumas implementações do Scheme disponibilizam  
vector-copy, o que não acontece com o DrScheme)  
  
> c2  
"a6b"  
> (string-fill! c2 #\z)  
> c2  
"zzz"  
> (string-length c1)  
3  
> (string-ref c1 1)  
#\6  
> (make-string 3 #\a)  
"aaa"  
> (make-string 3)  
"  
> (list->string '(#\1 #\2 #\a #\7))  
"12a7"  
> (string->list "abcd")  
(#\a #\b #\c #\d)  
> (string->list (string #\1 #\2 #\3 #\4))  
(#\1 #\2 #\3 #\4)  
> (define c51 (string #\0 #\2 #\4 #\6 #\8))  
> c51  
"02468"  
> (string-set! c51 2 #\a)  
> c51  
"02a68"
```





Experimene a funcionalidade do Scheme associada às cadeias de caracteres.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Com estes procedimentos, a implementação de soluções recursivas com cadeias de caracteres é semelhante à que foi utilizada com os vectores e também aqui assume uma forma diferente da utilizada para as listas. Inicialmente determina-se o comprimento da cadeia a processar e define-se um procedimento auxiliar, com um parâmetro que represente o índice ou a posição de um carácter na cadeia. A chamada inicial do procedimento auxiliar apresenta-se com o argumento zero, que vai sendo incrementado em cada nova chamada. A condição de terminação corresponde a verificar a igualdade entre este argumento e o comprimento da cadeia.

### Exemplo 1 - solução recursiva com cadeias de caracteres

Escreva em Scheme o procedimento `cadeia-de-maiusculas!`, que recebe uma cadeia de caracteres como argumento e altera a cadeia recebida, transformando as letras minúsculas em letras maiúsculas. Como se pode verificar, o procedimento pedido é um modificador e não um construtor, pois altera uma entidade existente. Devolve o símbolo `ok`.

```
(define cadeia-de-maiusculas!
  (lambda (cadeia)
    (let ((comprim (string-length cadeia)))
      (letrec ((aux
                (lambda (indice)
                  (if (= indice comprim)
                      'ok
                      (begin
                        (string-set! cadeia
                                      .... .... para completar
                        (aux (add1 indice)))))))
              ;
              (aux 0)))))

> (define cadeia (string #\a #\b))
> cadeia
"ab"
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
> (cadeia-de-maiusculas! cadeia)
ok
> cadeia
"AB"
```



Complete e teste o procedimento `cadeia-de-maiusculas!`.

### Exercício 1

O Scheme disponibiliza o predicado `string=?` que aceita dois argumentos do tipo cadeia de caracteres e devolve `#t` se e só se os argumentos forem iguais, comparando carácter a carácter.

```
> (string=? "abc" "ab")
#f
> (string=? "abc" "aBc")
#f
> (string=? "abc" (string #\a #\b #\c))
#t
```

Escreva em Scheme o procedimento `pessoal-cadeia=?`, uma versão pessoal do procedimento `string=?`.



Desenvolva e teste o procedimento `pessoal-cadeia=?`.

Para além dos procedimentos com paralelismo nos vectores, a abstracção cadeia de caracteres admite ainda outros procedimentos, próprios deste tipo de dados.

Os quatro procedimentos que se seguem funcionam como construtores e devolvem, todos eles, uma nova cadeia de caracteres.

# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



O construtor `substring` requer uma explicação adicional, uma vez que os outros têm uma funcionalidade fácil de entender. A cadeia devolvida por `substring` é uma cópia da cadeia dada desde o índice representado pelo segundo argumento até ao índice representado pelo terceiro argumento, o qual já não será incluído.

```
> (string-append "123" "abc" "!!!")
"123abc!!!"
> (substring "12abc345" 2 6)
"abc3"
> (symbol->string 'bom-dia)
"bom-dia"
> (number->string 123)
"123"
```

Os predicados comparam as cadeias carácter a carácter, tendo em consideração os respectivos códigos ASCII. O Scheme oferece uma gama completa de predicados: `string=?`, `string>?`, `string<?`, `string>=?`, `string<=?`.

```
> (string=? "abc" "ab3")
#f
> (string-ci=? "abc" "aBc")
#t
> (string<? "abc" "123")
#f
> (string<? "ABC" "abc")
#t
```

Finalmente, um procedimento que devolve um número representado por uma cadeia de caracteres.

```
> (string->number "123")
123
> (string->number "1ab")
#f
```

Por analogia com as listas, vamos desenvolver o procedimento construtor `string-cdr`, que recebe uma cadeia de caracteres como argumento e devolve uma cadeia equivalente à recebida, mas sem o primeiro carácter.

```
> (string-cdr "123456")
"23456"
> (string-cdr (string-cdr "123456"))
"3456"
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(define string-cdr
  (lambda (cadeia)
    (substring cadeia
               1
               (string-length cadeia))))
```

Encontrou-se para `string-cdr` uma solução relativamente simples, com a ajuda do procedimento primitivo `substring`. Agora, com este construtor, a recursividade nas cadeias de caracteres poderá também assumir uma forma idêntica à praticada com as listas, para as quais existe o `cdr`.

#### Exercício 2

Desenvolva o procedimento `cadeia-de-maiusculas-com-string-cdr` que, não sendo um modificador como `cadeia-de-maiusculas!`, mas um construtor, cria uma nova cadeia de caracteres a partir da cadeia que recebe como argumento, transformando as letras minúsculas em maiúsculas.



Desenvolva e teste o procedimento `cadeia-de-maiusculas-com-string-cdr`.

Pista: Sugere-se uma solução recursiva, em que esta assuma a forma de recursividade característica das listas, recorrendo ao procedimento `string-cdr`.

#### Exercício 3

O procedimento `cadeia-maius-minus-e-vice-versa` transforma as letras maiúsculas em minúsculas e as minúsculas em maiúsculas, da cadeia que recebe como argumento.

Desenvolva duas versões deste procedimento, uma como modificador da cadeia recebida e outra como construtor de uma nova cadeia.



Desenvolva e teste as duas versões do procedimento `cadeia-maius-minus-e-vice-versa`.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Dados guardados em ficheiros para reutilização futura

Até ao momento, alguns dos programas desenvolvidos esperavam dados vindos do teclado com o procedimento `read` e visualizavam os resultados no ecrã com o procedimento `display`. Esta forma de interactuar com o utilizador nem sempre é aceitável, sobretudo quando os dados a fornecer são em grande quantidade, o que inviabiliza o fornecimento repetido de dados através do teclado, ou então quando se pretende guardar resultados de uma sessão de trabalho para outra sessão. Tome como exemplo um programa de análise estatística sobre o aproveitamento escolar dos estudantes de um curso. Se considerar que são 200 os estudantes desse curso e que durante um ano cada um deles está inscrito em 10 disciplinas, o cálculo da classificação média de todo o curso envolveria 2000 dados. Não é razoável aceitar que o director desse curso, ao querer conhecer essa classificação média, tenha que fornecer 2000 dados pelo teclado. É óbvio que outras análises se poderão fazer sobre os resultados obtidos pelos estudantes do curso, estudante a estudante ou englobando todos os estudantes. Estes e outros estudos referem-se aos mesmos dados que deverão estar organizados e disponíveis sempre que forem necessários. Isto é possível através de ficheiros, implementados sobre memória não volátil, memória que mantém os dados registados mesmo quando é dada por encerrada uma sessão do programa e o computador é desligado completamente. Como suporte de memória não volátil temos, como exemplo, os discos existentes em praticamente todos os computadores.

Em Scheme, para abrir um ficheiro para escrita, ou seja, para guardar alguns dados, é necessário:

- associar uma porta de saída (*output port*) ao ficheiro a abrir, sendo este especificado através do respectivo nome, que é uma cadeia de caracteres.

```
(define porta-saida (open-output-file nome-ficheiro-a-abrir))
```



O procedimento `open-output-file` devolve uma porta de saída que fica associada ao ficheiro cujo nome é recebido como argumento.

Se já existe um ficheiro com este nome, o que acontece depende da implementação do Scheme, mas o mais usual é limpar o conteúdo do ficheiro.

Se o ficheiro não existe, é criado um novo com o nome especificado.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Até aqui, sempre que utilizou **display** e **newline** sem especificar a porta de saída, significava que pretendia aceder ao ecrã (*standard-output*).

- Fechar o ficheiro, quando não há mais dados para nele guardar

(**close-output-port** *porta-saida*)

#### Exemplo 2 - Como criar ficheiros em modo de escrita?

Um programa designado por **cria-ficheiro-ano-de-nascimento** não tem parâmetros e começa por interrogar o utilizador sobre o nome do ficheiro que pretende criar para registar o nome e o ano de nascimento de vários amigos.

O programa mantém um diálogo com o utilizador, para que este lhe transmita os dados a registar.

```
> (cria-ficheiro-ano-de-nascimento)
nome do ficheiro: "anoNasc.txt"
```



O ficheiro especificado com o nome **anoNasc.txt** é criado no mesmo directório onde se encontra o programa **cria-ficheiro-ano-de-nascimento**.

Para o colocar noutra local, deve ser especificado o respectivo caminho.

Por exemplo, "**d:\\\Scheme\\\Trabalhos\\aulas\\anoNasc.txt**".

A duplicação do carácter \ é necessária, pois o Scheme elimina um deles.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
Nome (-1 = fim) : "joao antunes"
Ano de nascimento (-1 = fim) : 1949
Nome (-1 = fim) : "manuel antonio"
Ano de nascimento (-1 = fim) : 1952
Nome (-1 = fim) : "antonio jose"
Ano de nascimento (-1 = fim) : 1950
Nome (-1 = fim) : "maria costa"
Ano de nascimento (-1 = fim) : 1954
Nome (-1 = fim) : "joaquim sousa"
Ano de nascimento (-1 = fim) : 1943
Nome (-1 = fim) : -1
Ano de nascimento (-1 = fim) : -1
```

Após este diálogo, o conteúdo do ficheiro anoNasc.txt deverá apresentar o conteúdo que se segue.

```
"joao antunes"1949
manuel antonio"1952
antonio jose"1950
maria costa"1954
joaquim sousa"1943
```

Como pode verificar, os dados respeitantes a cada uma das pessoas foram registados numa linha de texto, com dois elementos, em que o primeiro elemento é uma cadeia de caracteres (o nome) e o segundo é um inteiro (o ano de nascimento).

O programa `cria-ficheiro-ano-de-nascimento` mostra claramente as tarefas identificadas anteriormente para abrir um ficheiro para escrita.

```
(define cria-ficheiro-ano-de-nascimento
  (lambda ()
    (display "nome do ficheiro: ")
    (let ((nome-fich (read)))
      (let ((porta-out (open-output-file nome-fich)))
        (pedir-dados porta-out)
        (close-output-port porta-out)))))
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



De cada vez que o procedimento pedir-dados chama ler-nome-ano, é estabelecido um diálogo com o utilizador que conduz à devolução dos dados relativos a uma pessoa, ou seja, um par cuja parte esquerda é o nome e parte direita é o ano de nascimento, ou então ao retorno do símbolo fim, indicando que não há mais dados a fornecer.

```
(define pedir-dados
  (lambda (porta-fich)
    (let ((nome-ano (ler-nome-ano)))
      (if (equal? nome-ano 'fim)
          'fim
          (begin
            (display #\" porta-fich) ; registo de um carácter "
            (display (car nome-ano) porta-fich) ; registo do nome
            (display #\" porta-fich) ; registo de um carácter "
            (display (cdr nome-ano) porta-fich)
            (newline porta-fich)
            (pedir-dados porta-fich))))))
```



Foi necessário colocar o carácter aspas em torno do nome, para o manter como uma cadeia de caracteres.  
Não esquecer que o display não visualiza as aspas.

```
(define ler-nome-ano
  (lambda ()
    (newline)
    (display "Nome (-1 = fim): ")
    (let ((nome (read)))
      (newline)
      (display "Ano de nascimento (-1 = fim): ")
      (let ((ano (read)))
        (if (or (equal? nome -1)
                (equal? ano -1))
            'fim
            (cons nome ano))))))
```

<b>INTRODUÇÃO</b>
<b>1 - O ESSENCIAL DO SCHEME</b>
<b>2 - RECURSIVIDADE</b>
<b>3 - ABSTRACÇÃO DE DADOS</b>
<b>4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE</b>
<b>5 - ABSTRACÇÕES COM DADOS MUTÁVEIS</b>
R E 1 2 3 4 5
<b>6 - SCHEME E OUTRAS TECNOLOGIAS</b>
<b>7-EXERCÍCIOS E PROJECTOS</b>
<b>ANEXOS</b>



Teste o programa `cria-ficheiro-ano-de-nascimento`.

Experimente criar vários ficheiros, sendo alguns casos, com nome de um ficheiro existente.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

- Sempre que um ficheiro é acedido em leitura, o elemento lido deverá ser testado a fim de se verificar se já se atingiu o fim do ficheiro (`eof - end of file`).  
`(eof-object? ultimo-elemento-resultante-da-operacao-de-read)`
- Fechar o ficheiro, logo que deixe de fazer sentido mantê-lo aberto, por exemplo, quando a operação anterior devolver `#t`  
`(close-input-port porta-entrada)`

Um programa designado por `le-ficheiro-e-faz-relatorio` não tem parâmetros e começa por interrogar o utilizador sobre o nome do ficheiro onde foram registados o nome e o ano de nascimento de vários amigos, com a ajuda do programa `cria-ficheiro-ano-de-nascimento`, anteriormente apresentado. Em seguida, o programa abre o ficheiro, pede ao utilizador a indicação de um ano de referência e apresenta um relatório com o nome de todas as pessoas registadas no ficheiro e a respectiva idade, no ano de referência.

```
> (le-ficheiro-e-faz-relatorio)
nome do ficheiro: "anoNasc.txt"

ano de referencia: 1999

Relatorio de idades relativo a 1999
joao antunes      50
manuel antonio    47
antonio jose      49
maria costa       45
joaquim sousa     56

Fim do relatorio

> (le-ficheiro-e-faz-relatorio)
nome do ficheiro: "anoNasc.txt"

ano de referencia: 1950

Relatorio de idades relativo a 1950
joao antunes      1
manuel antonio    -2
antonio jose      0
maria costa       -4
joaquim sousa     7

Fim do relatorio
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



A implementação do programa `le-ficheiro-e-faz-relatorio` mostra as tarefas identificadas no enunciado.

- pedido do nome do ficheiro a processar;
- pedido do ano em relação ao qual são calculadas as idades;
- visualização do relatório respectivo
  - cabeçalho do relatório
  - corpo do relatório
- Fecho do ficheiro

Para a visualização do relatório, tarefa um pouco mais complexa que as restantes, recorreu-se ao procedimento designado por `relatorio`.

```
(define le-ficheiro-e-faz-relatorio
  (lambda ()
    (display "nome do ficheiro: ")
    (let ((nome-fich (read)))
      (let ((porta-in (open-input-file nome-fich)))
        (newline)
        (display "ano de referencia: ")
        (let ((ano-ref (read)))
          (newline)
          (display "Relatorio de idades relativo a ")
          (display ano-ref)
          (newline)
          (relatorio porta-in ano-ref)
          (close-input-port porta-in))))))
```

O procedimento `relatorio` utiliza o procedimento local `percorre-ficheiro` para ler os dados associados a cada pessoa e para verificar quando é atingido o fim do ficheiro. Deve ter em atenção que, sempre que se lê um valor do ficheiro (na leitura do nome e na leitura do ano), verifica-se se já se atingiu o fim desse ficheiro, aplicando o predicado `eof-object?` ao valor lido.



**Este cuidado é fundamental quando se trabalha com ficheiros.**

**Fazendo analogia, um cuidado semelhante também era tido, antes de aceder a uma lista, verificando se era vazia, com o predicado `null?`. No caso da lista, o teste era feito antes do acesso, mas no caso do ficheiro é feito depois.**

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Por seu turno, o procedimento `percorre-ficheiro` recorre ao procedimento auxiliar `visu-registo`. Este procedimento auxiliar toma como argumentos os dados associados a uma pessoa (um par, com o nome na parte esquerda e o ano de nascimento na parte direita) e o ano de referência e visualiza a linha respectiva.

```
(define relatorio
  (lambda (porta-fich ano-refer)
    ;
    (letrec ((percorre-ficheiro
              (lambda ()
                (let ((nome (read porta-fich)))
                  (cond ((eof-object? nome)
                           (newline)
                           (display "Fim do relatorio"))
                        (else
                          (let ((ano (read porta-fich)))
                            (cond ((eof-object? ano)
                                     (newline)
                                     (display "Fim do relatorio"))
                                  (else
                                    (visu-registo (cons nome ano) ano-refer)
                                    (percorre-ficheiro))))))))
                    (percorre-ficheiro)))))

(define visu-registo
  (lambda (reg-pessoa ano-rf)
    .... .... para completar
```



Complete e teste o programa `le-ficheiro-e-faz-relatorio`.

Experimente ler vários ficheiros, em alguns casos, com o nome de ficheiro não existente.

# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



### Exercício 4

Relativamente ao procedimento `le-ficheiro-e-faz-relatorio` anteriormente apresentado, desenvolva uma versão melhorada, em que o relatório apresenta as idades alinhadas, a partir da coluna 25.

```
> (le-ficheiro-e-faz-relatorio-v2)
nome do ficheiro: "anoNasc.txt"
```

```
ano de referencia: 1999
```

```
Relatorio de idades relativo a 1999
```

joao antunes	50
manuel antonio	47
antonio jose	49
maria costa	45
joaquim sousa	56

```
Fim do relatorio
```



Desenvolva e teste o programa `le-ficheiro-e-faz-relatorio-v2`.

**Exercício 5** - Para sistematizar e aprender mais alguma coisa sobre ficheiros em Scheme

Em primeiro lugar, analise com atenção o diálogo e as imagens que se seguem.

Criação do ficheiro de texto "c:\\expfnf\\lixo111.txt".

```
> (define f1 (open-output-file "c:\\expfnf\\lixo111.txt"))
> (display "lin 1" f1)
> (newline f1)
> (display "lin 2" f1)
> (close-output-port f1)
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

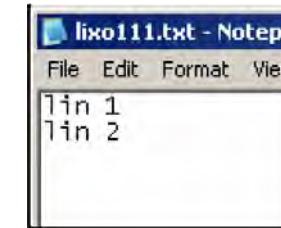
### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

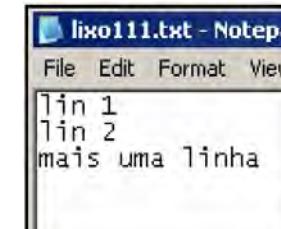
Leitura do ficheiro de texto "c:\\expfnf\\lixo111.txt".

```
> (define f1 (open-input-file "c:\\expfnf\\lixo111.txt"))
> (read f1)
lin
> (read f1)
1
> (read f1)
lin
> (read f1)
2
> (read f1)
#<eof>
>(close-input-port f1)
```



Reabertura do ficheiro de texto "c:\\expfnf\\lixo111.txt", em modo append (para acrescentar mais elementos, a partir do final do ficheiro).

```
> (define f10 (open-output-file "c:\\expfnf\\lixo111.txt" 'append)) ← novo
> (newline f10)
> (display "mais uma linha" f10)
> (close-output-port f10)
```



Reabertura do ficheiro de texto "c:\\expfnf\\lixo111.txt", em modo replace (para refazer o conteúdo do ficheiro, limpando-o previamente).



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

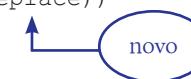
R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (define fx (open-output-file "c:\\expfnf\\lixo111.txt" 'replace))
> (newline fx)
> (display "mais uma linha" fx)
> (close-output-port fx)
```



Leitura do ficheiro de texto "c:\\expfnf\\lixo111.txt", depois de refeito em modo *replace*.

```
> (define f1 (open-input-file "c:\\expfnf\\lixo111.txt"))
> (read f1)
mais
> (read f1)
uma
> (read f1)
linha
> (read f1)
#<eof>
> (close-input-port f1)
```



Leitura do ficheiro de texto "c:\\expfnf\\lixo111.txt", depois de refeito em modo *replace*, mas agora para observar o predicado *eof-object?*.

```
> (define f1 (open-input-file "c:\\expfnf\\lixo111.txt"))
> (define x (read f1))
> (eof-object? x)
#f
> x
mais
> (define x (read f1))
> (eof-object? x)
#f
> x
uma
> (define x (read f1))
> (eof-object? x)
#f
> x
linha
> (define x (read f1))
> (eof-object? x)
#t
> x
#<eof>
> (close-input-port f1)
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Observar agora, o efeito dos procedimentos `file-exists?` e `delete-file`.

```
> (file-exists? "c:\\expfnf\\lixo111.txt")
#t
> (delete-file "c:\\expfnf\\lixo111.txt")
> (file-exists? "c:\\expfnf\\lixo111.txt") ← novo
#f
>
```

### Exercício 5

O problema que agora se coloca é o seguinte: como o Scheme não nos oferece funcionalidades para alterar elementos já registados em ficheiros de texto, propõe-se que desenvolva o procedimento `escrever-ficheiro!` que tem como parâmetros `novo-elemento`, `ordem` e `nome-ficheiro`.

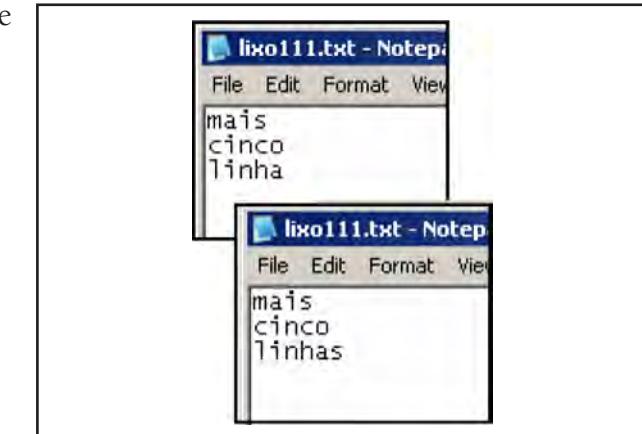
O argumento correspondente ao parâmetro `novo-elemento` irá substituir o elemento especificado por `ordem` (que é um inteiro; se 0, indica o primeiro elemento do ficheiro, se 1, indica o segundo...) no ficheiro especificado por `nome-ficheiro`.

No diálogo que se segue, imagine que o ficheiro `lixo111.txt` não tinha sido eliminado e continha `mais cinco linhas`.

```
> (escrever-ficheiro! "linhas" 2 "c:\\expfnf\\lixo111.txt")
```

O procedimento `escrever-ficheiro!` deve detectar dois tipos de erro:

```
> (escrever-ficheiro! "linhas" 2 "c:\\expfnf\\lixo88888.txt")
erro- ficheiro nao existe
> (escrever-ficheiro! "linhas" 8 "c:\\expfnf\\lixo111.txt")
erro- elemento nao existe
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Desenvolva e teste o procedimento `escrever-ficheiro!`.

Pista 1: o procedimento a desenvolver recebe como primeiro argumento o mesmo que `display`.

Pista 2: ler o ficheiro para uma lista, reabrir o ficheiro em modo `replace` para o reescrever a partir da estrutura intermédia, com o cuidado de alterar um certo elemento...

Para treinar, crie e leia ficheiros com os procedimentos Scheme apresentados e experimente modificá-los com o procedimento desenvolvido.



Para aceder à funcionalidade especificada para o procedimento `escrever-ficheiro!` poderá fazer:

```
(require (lib "ficheiros.scm" "user-feup"))
```



## INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

## Módulo 5.5 - Pesquisas, Ordenações e Árvores binárias

As operações de pesquisa e ordenação são as operações em destaque neste módulo, incidindo especialmente sobre árvores binárias e vectores. Das árvores binárias, distinguiu-se a de pesquisa binária pelo excelente desempenho  $O(\lg_2 n)$  que oferece quando se encontra equilibrada ou balanceada.

### Palavras-Chave

Pesquisa sequencial, pesquisa binária, árvore binária, árvore de pesquisa binária, árvore equilibrada ou balanceada, ordenação de listas por inserção, ordenação de vectores por selecção (*selection sort*), por bolha (*bubble sort*) e por balde (*bucket*), ordenação rápida de vectores (*quick sort*), ordenação por mistura de vectores (*merge sort*).



# **S C H E M E**

## **na descoberta da programação**

### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS**

R E 1 2 3 4 5

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

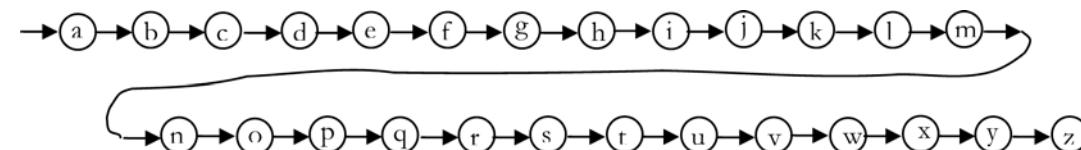
### **7-EXERCÍCIOS E PROJECTOS**

### **ANEXOS**

## **A importância de pesquisar um elemento numa estrutura de dados**

O comportamento das operações sobre entidades de uma abstracção está normalmente muito dependente da forma como essas entidades se estruturam. Com a lista, aquelas operações assumem frequentemente um carácter sequencial e, por exemplo, a pesquisa de um elemento começa num determinado elemento, vai percorrendo sequencialmente os outros elementos e só termina quando encontra o elemento a pesquisar ou quando chegou ao fim. E com estas entidades implementadas com listas, é de esperar que as pesquisas apresentem comportamentos  $O(n)$ . Este tipo de comportamento na pesquisa não é aceitável, normalmente em situações que envolvam um grande número de elementos, como, por exemplo, uma longa lista telefónica - para chegar a um dos últimos nomes da lista seria necessário passar por todos os que estão antes dele...

A figura esboça uma lista telefónica imaginária, onde cada círculo representa, simbolicamente, todos os nomes iniciados pela letra respectiva. A estrutura de base poderia ser uma tabela (uma lista de registos), com um registo por entidade existente na lista telefónica. Os primeiros elementos seriam os registos associados às entidades cuja letra inicial é a. Mas para chegar a um manuel, a um pedro ou, pior ainda, a um zeferino, seria necessário percorrer um longo caminho...



**Nesta lista telefónica imaginária, que estrutura escolheria para cada um dos seus registos? Justifique.**



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

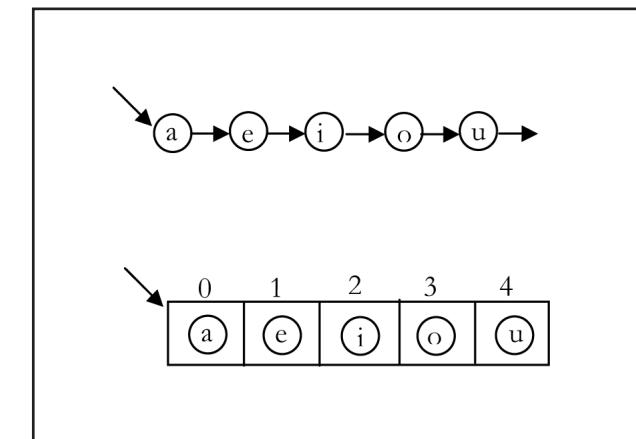
### ANEXOS



## Vectores em vez de listas, abrindo caminho a pesquisas com comportamento $O(\log_2 n)$

A utilização de vectores, em vez de listas, pode transformar completamente o cenário da pesquisa. Por uma questão de simplificação do que se segue, vamos imaginar que a lista telefónica só tem nomes começados por vogais. A figura esboça a estruturação dos dados, com uma lista e com um vector, em ambos os casos com os elementos ordenados alfabeticamente do menor para o maior.

Para encontrar `o` de `e`, pode começar por pesquisar o vector de uma forma sequencial, o que não é boa opção, partindo da posição de índice 0, até chegar a `o` de `e`, passando pelo `a`, `e` e `i`, ou seja, ao fim de 4 passos.



### Exercício 1 - pesquise sequencial em vectores e medição do tempo de resposta

Escreva em Scheme o procedimento `pesquisa-seq-vec` com os parâmetros `vec` e `elem`, respectivamente, um vector de valores numéricos (ordenados ou não) e um elemento numérico a pesquisar naquele vector. O procedimento utiliza a pesquisa sequencial e devolve o índice da posição do vector onde se encontra o elemento a pesquisar ou `-1` se o elemento não existir no vector.



Desenvolva e teste o procedimento `pesquisa-seq-vec`.

Crie (por ex., com `make-vector`) vectores de comprimento cada vez maior (vectores de milhares de elementos) para medir, com o procedimento `time` ([Anexo A](#)), o tempo de resposta do procedimento `pesquisa-seq-vec`.



Indique a ordem de complexidade do procedimento **pesquisa-seq-vec** desenvolvido, em relação ao tempo e ao espaço. Justifique.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Com a pesquisa sequencial do vector, é natural que tenha obtido para o procedimento **pesquisa-seq-vec** um comportamento  $O(n)$ . Mas tem à sua disposição uma solução bem melhor... pois pode encontrar **odete** de uma forma muito diferente, explorando outro tipo de acesso que o vector oferece, o designado acesso aleatório (*random access*).

Agora a pesquisa começa pela posição central do vector, ou seja, na posição de índice 2. É nesta posição que se encontram os nomes iniciados por **i**, valor alfabeticamente inferior ao **o** de **odete**. Sendo **i** alfabeticamente menor que **o**, **odete** só poderá estar na metade superior do vector. Continue, por isso, pela posição central desta metade superior, ou seja pela posição de índice 3... e com dois passos encontra o espaço dos nomes começados por **o**, onde estará **odete**.

0	1	2	3	4
(a)	(e)	(i)	(o)	(u)



Antes de avançar para o parágrafo que se segue, indique e justifique o comportamento deste tipo de pesquisa.

O que ocorre, em cada uma das decisões desta pesquisa, é a eliminação de metade dos elementos que ainda estão por pesquisar, o que corresponde a um comportamento  $O(\log_2 n)$ , muito melhor que  $O(n)$ . Está perante um exemplo de uma pesquisa binária.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 2 - pesquisa binária em vectores e medição do tempo de resposta

Escreva em Scheme o procedimento `pesquisa-binaria-vec` com os parâmetros `vec` e `elem`, respectivamente, um vector numérico ordenado do menor para o maior e um elemento numérico a pesquisar naquele vector. O procedimento utiliza a pesquisa binária e devolve o índice da posição do vector onde se encontra o elemento a pesquisar ou `-1` se o elemento não existir no vector.



Desenvolva e teste o procedimento `pesquisa-binaria-vec`.

Crie vectores de comprimento cada vez maior (vectores de milhares de elementos) com valores numéricos ordenados do menor para o maior (se necessário, escreva um procedimento para criar estes vectores...) e meça com `time` os tempos de resposta obtidos com os procedimentos `pesquisa-seq-vec` e `pesquisa-binaria-vec`.



Indique a ordem de complexidade do procedimento `pesquisa-binaria-vec` desenvolvido, em relação ao tempo e ao espaço. Justifique.

Compare a ordem de complexidade, em relação ao tempo e ao espaço, deste procedimento e do procedimento `pesquisa-seq-vec`.

Em termos de pesquisa, uma solução baseada num vector carregado com elementos ordenados parece ser imbatível ou, pelo menos, muito melhor do que outra que utilize a lista. No entanto, enquanto a lista é uma estrutura dinâmica (que pode crescer ou diminuir, acrescentando-lhe ou retirando-lhe um elemento, em qualquer um dos seus pontos), o vector surge como uma estrutura estática (o seu comprimento é definido quando é criado). E mesmo que crie um vector com mais elementos do que os necessários no início, surgem algumas dificuldades, que não ocorriam com as listas, quando se pretende acrescentar ou retirar um elemento. Retirar um elemento obriga a deslocar, uma posição para a esquerda, todos os elementos à direita do elemento retirado.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Acrescentar um elemento obriga a deslocar, uma posição para a direita, todos os elementos a partir desse ponto. Esta operação só será possível se ainda houver espaço disponível no vector.



**Através de um desenho, mostre como acrescentaria ou retiraria um elemento num vector, numa posição mais ou menos central.**

**Identifique a situação que exigiria mais tempo. Justifique.**

**Verifique que, esgotado um eventual espaço de reserva, já não é possível acrescentar mais um elemento.**

Na lista mutável, sendo esta uma estrutura dinâmica, para lhe retirar ou acrescentar um elemento bastará procurá-lo, ou procurar o ponto de inserção, e abrir a lista nesse ponto para realizar a operação desejada.



**Através de um desenho, mostre como acrescentaria ou retiraria um elemento numa lista, numa posição mais ou menos central.**

**Identifique a situação que exigiria mais tempo. Justifique.**

**Mostre que não há impedimento ao acrescentar mais um elemento.**

A estrutura ideal parece ser a que reúna o melhor dos dois mundos, que suporte naturalmente a pesquisa binária (como o vector), mas com a facilidade de crescer ou diminuir que caracteriza uma estrutura dinâmica (como a lista mutável).

**A solução pode estar na árvore de pesquisa binária!!!**

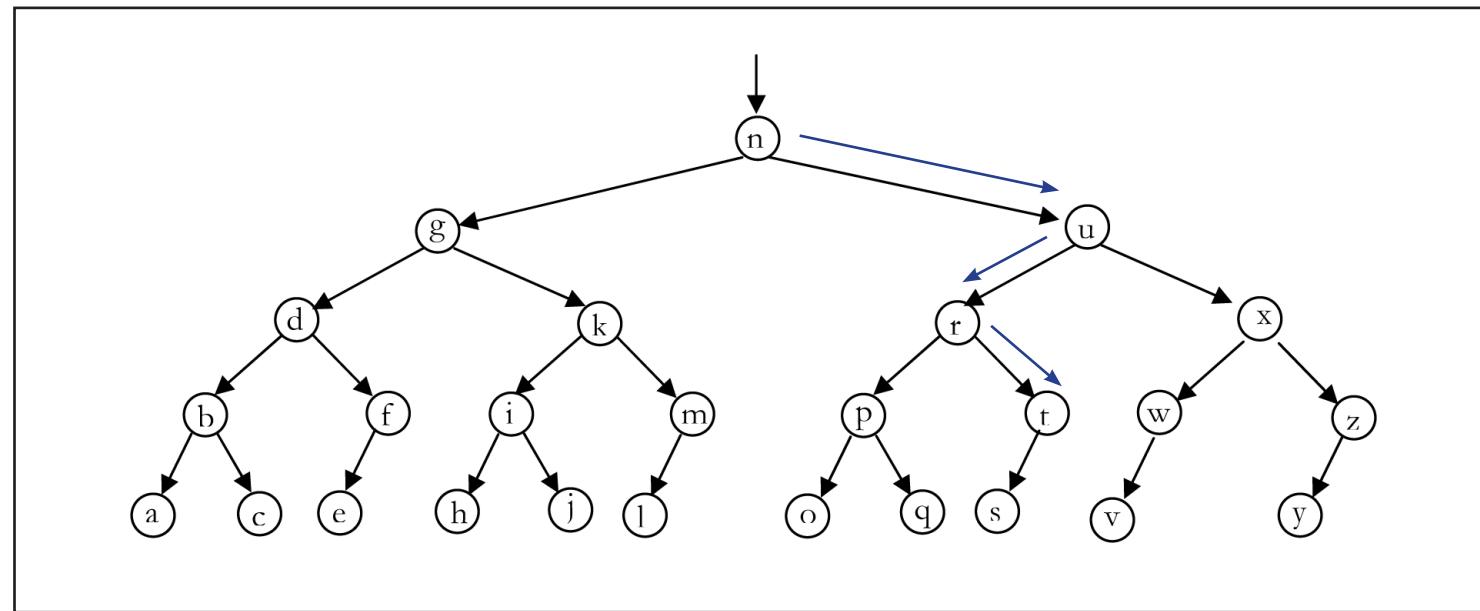
Imagine que a lista telefónica se organiza de uma maneira diferente, não em sequência, mas sim como se indica na figura, que esboça uma lista telefónica tomando como base uma árvore binária (árvore que não terá mais de dois ramos por nó).



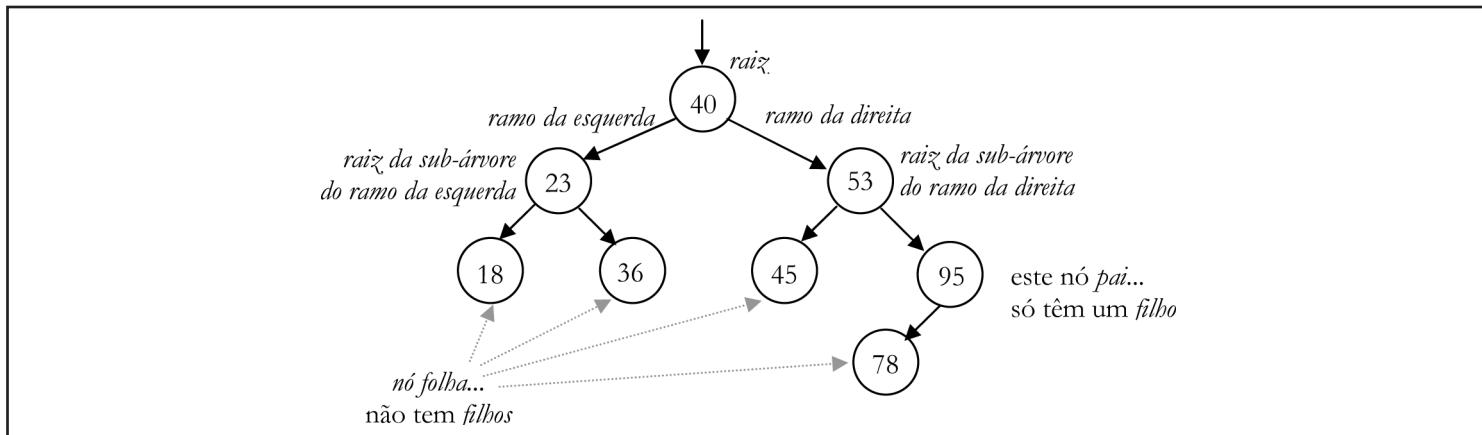
Suponha que quer encontrar **tomás**. O acesso à lista telefónica inicia-se pela letra **n**, não pela letra **a**, e a inicial do nome a encontrar é **t**.

- $t$  não é igual a  $n$ , alfabeticamente é maior. Então, o acesso continua pelo ramo da direita até ao nó  $u$ ;
  - $t$  não é igual a  $u$ , alfabeticamente é menor. Então, o acesso continua pelo ramo da esquerda até ao nó  $r$ .
  - $t$  não é igual a  $r$ , alfabeticamente é maior. Então, o acesso continua pelo ramo da direita até ao nó  $t$ .

E assim chegava à zona correspondente aos nomes iniciados por t, onde se encontra o tomás, sem passar por todos os nomes que lhe são anteriores, segundo a ordenação alfabética.



Já reparou que, em cada decisão, são eliminados cerca de metade dos nomes que ainda estão por pesquisar, o que corresponde a um comportamento  $O(\log_2 n)$ ?



- uma árvore é composta por nós, sendo o nó de entrada designado por raiz da árvore. Uma árvore com zero nós é uma árvore vazia.
  - numa árvore binária, partem normalmente dois ramos de um nó, o nó pai: o ramo da esquerda e o ramo da direita, mas também pode partir apenas um só ramo (ou da esquerda ou da direita) ou zero, situação em que o nó é designado por folha, por ser terminal. Assim, o nó pai pode ter dois, um ou zero filhos.



**Um nó filho não terá mais do que um nó pai. Concorda?**

- qualquer ramo de uma árvore apresenta-se com a estrutura de uma árvore, ou seja, é uma sub-árvore também com a sua raiz e com os seus eventuais ramos.
  - o peso de uma árvore é o comprimento do mais longo caminho desde a raiz a uma folha.



Na figura anterior, indo pela sub-árvore da esquerda, o caminho mais longo tem comprimento 2. E pela sub-árvore da direita, qual é o comprimento do caminho mais longo? Qual é então o peso desta árvore?

- uma árvore binária A diz-se equilibrada ou balanceada se for uma árvore vazia ou então se
    - o peso da sub-árvore da esquerda de A diferir no máximo de 1 do peso da sub-árvore da direita de A;
    - e as sub-árvores da esquerda e da direita forem árvores equilibradas ou balanceadas.



**Em relação à árvore da figura anterior, diga se é ou não uma árvore binária equilibrada. Justifique.**

- uma árvore binária A é de pesquisa binária se for uma árvore vazia ou então se
    - todos os elementos da sub-árvore da esquerda de A forem menores que a raiz de A;
    - todos os elementos da sub-árvore da direita de A forem maiores que a raiz de A;
    - as sub-árvore da esquerda e da direita forem árvores de pesquisa binária.

A relação maior ou menor referida poderá ser numérica, alfabetica ou outra qualquer que permita uma ordenação.



A árvore da figura anterior é uma árvore de pesquisa binária? Justifique.



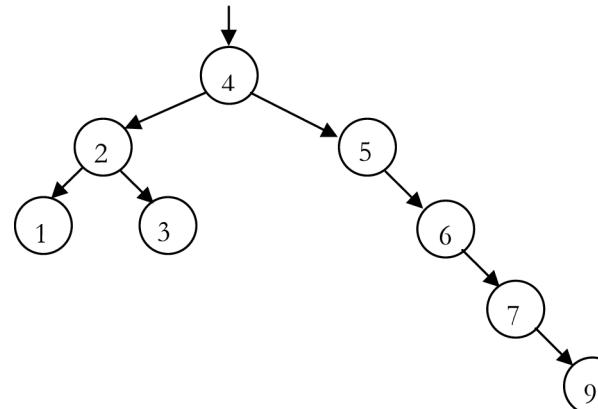


Considerando a árvore binária apresentada na figura seguinte

Trata-se de uma árvore equilibrada ou balanceada? Justifique.

Trata-se de uma árvore de pesquisa binária? Justifique.

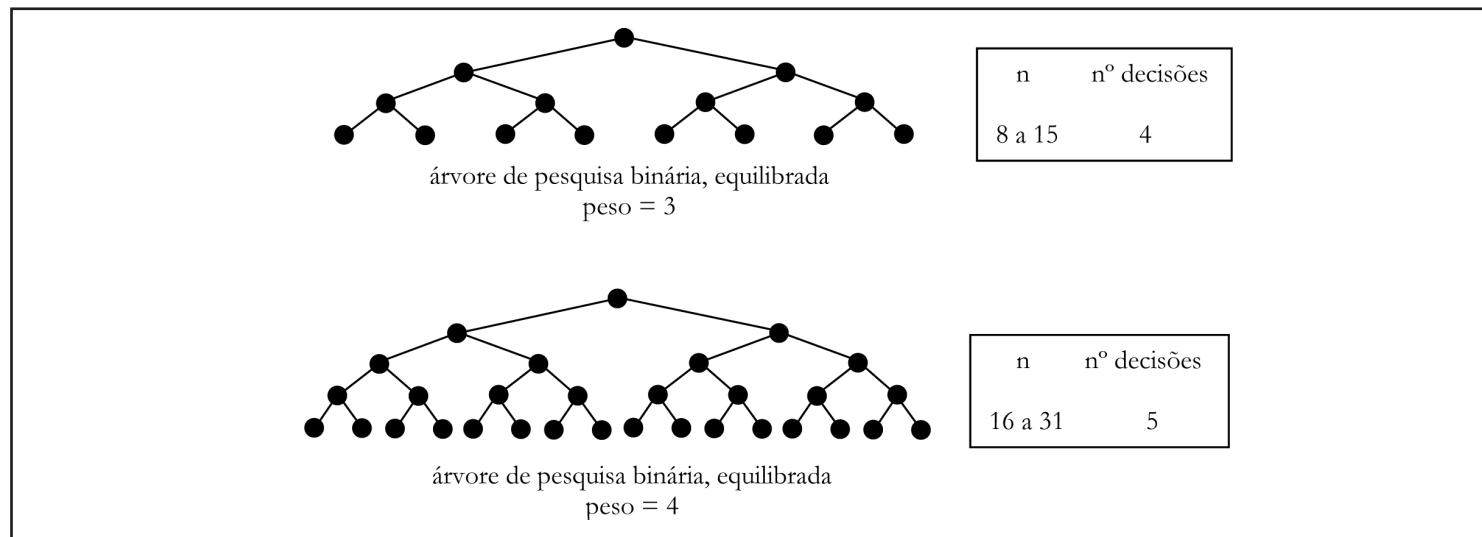
**Relacione a pior situação de acesso a um elemento desta árvore com o seu peso. Justifique**



E se a árvore de pesquisa binária for equilibrada, onde se situam os nós a que correspondem as piores situações de acesso? Justifique.

Relativamente ao tempo, indique a ordem de complexidade do acesso a uma árvore de pesquisa binária equilibrada, em função de  $n$ , o número de nós da árvore.

A figura que se segue mostra a relação entre o número de nós da árvore de pesquisa binária equilibrada, o seu peso e o número de decisões tomadas para aceder às folhas.



**Apresente uma explicação para o facto de o número de decisões ser igual 5, para um número de nós que vai de 16 a 31.**

**Numa árvore de pesquisa binária não equilibrada de 31 nós, qual seria o maior número possível de decisões que poderia ocorrer? Justifique.**

## A abstracção árvore de pesquisa binária

A importância da árvore de pesquisa binária, com um comportamento  $O(\log_2 n)$ , motiva a definição e o desenvolvimento de uma abstração, com a funcionalidade que se segue.

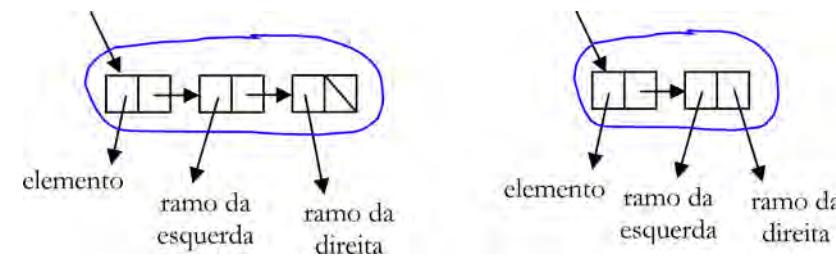
- $(faz-arvore\ elem\ r-esq\ r-dir)$  - devolve uma árvore binária com um elemento, `elem`, um ramo da esquerda, `r-esq`, e um ramo da direira, `r-dir`. Trata-se de um construtor.
  - $(raiz\ arv)$ , devolve a raiz de `arv`, em que este parâmetro é do tipo árvore binária. Trata-se de um selector.
  - $(ramo-esq\ arv)$  - devolve o ramo da esquerda de `arv`, em que este parâmetro é do tipo árvore binária. Trata-se de um selector.

- (`ramo-dir` `arv`) - devolve o ramo da direita de `arv`, em que este parâmetro é do tipo árvore binária. Trata-se de um selector.
  - (`arv-vazia?` `arv`) - devolve `#t`, se `arv` for uma árvore binária vazia, caso contrário devolve `#f`. Trata-se de um selector.
  - (`na-arv?` `elem` `arv`) - se `elem` for um elemento de `arv`, em que este parâmetro é do tipo árvore de pesquisa binária, devolve o nó onde se encontra `elem`, caso contrário devolve `#f`. Trata-se de um selector.
  - (`poe-na-arv!` `elem` `arv`) - se `elem` não for elemento de `arv`, em que este parâmetro é do tipo árvore de pesquisa binária, põe `elem` em `arv` garantindo que esta continuará a ser uma árvore de pesquisa binária. Se `elem` já for elemento de `arv`, então nada modifica. Em ambos os casos devolve o símbolo `ok`. Trata-se de um modificador.



Na funcionalidade indicada, são sempre referidas árvores binárias, enquanto a especificação árvore de pesquisa binária é apenas reservada para `na-arv?` e `poe-na-arv!`.  
Estará bem ou terá sido uma falha? Justifique.

Antes de passar à implementação da abstracção árvore binária, coloca-se à sua apreciação a forma de representar computacionalmente um nó da árvore binária.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Das duas formas indicadas, qual escolheria para representar os nós da árvore binária? Justifique.

### Exercício 3

Complete o desenvolvimento da abstracção árvore binária.

```
(define faz-arvore
  (lambda (elem r-esq r-dir)
    (list elem (cons r-esq r-dir))))
(define raiz
  .... .... .... para completar
(define ramo-esq
  .... .... .... para completar
(define ramo-dir
  .... .... .... para completar
(define arv-vazia?
  (lambda (arv)
    (null? arv)))
(define na-arv?
  (lambda (elem arv)
    .... .... .... para completar
```



Um algoritmo para o procedimento `na-arv?`.

se a árvore for vazia, a chave não está na árvore e devolve #f.

se não,

    se a chave for a raiz, encontrou a chave e devolve o nó (elemento e ramos) raiz.

    se não,

        se a chave for menor que a raiz,

            procura na sub-árvore do ramo da esquerda.

        se não,

            procura na sub-árvore do ramo da direita.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(define poe-na-arv!
  (lambda (elem arv)
    ... ... ... para completar
```



Um algoritmo para o procedimento **poe-na-arv!**.  
se a árvore for vazia, põe o elemento na raiz da árvore.  
se não,  
  se a chave for a raiz, não põe o elemento na árvore (para evitar duplicações).  
  se não,  
    se a chave for menor que a raiz,  
      põe o elemento na sub-árvore do ramo da esquerda.  
    se não,  
      põe o elemento na sub-árvore do ramo da direita.  
  
devolve ok.



Complete e teste a abstracção árvore binária.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

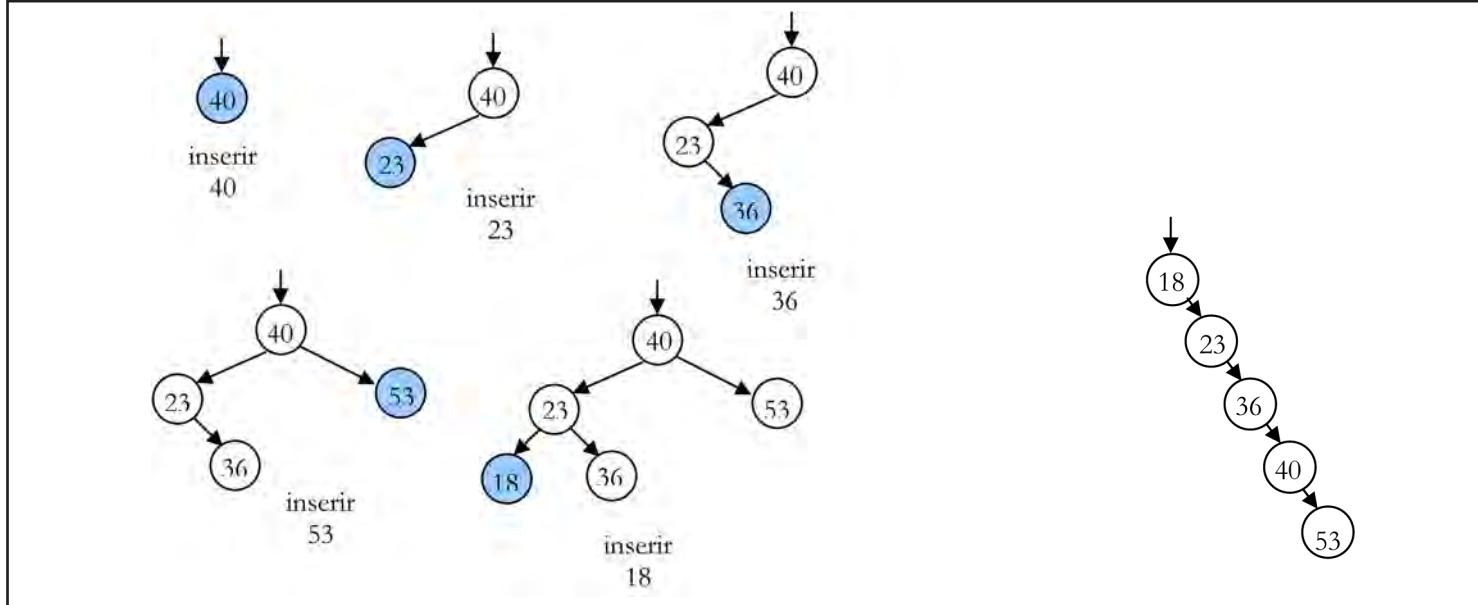
## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Analise as duas árvores da figura, ambas constituídas com os mesmos elementos: 18, 23, 36, 40 e 53.  
A diferença entre as duas reside apenas na ordem de inserção dos vários elementos.  
Indique a ordem de inserção dos elementos da árvore da direita.



Indique o peso das duas árvores de pesquisa binária representadas na figura.  
São ambas árvores de pesquisa binária equilibradas?  
Qual o tipo de comportamento da pesquisa em cada uma delas?

Ao inserir novos elementos numa árvore de pesquisa binária, só por sorte ela se manterá equilibrada, o que afasta o comportamento  $O(\log_2 n)$  que caracteriza este tipo de árvore. Este problema fica, para já, por resolver.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Atravessar árvores binárias... e se forem de pesquisa binária, o resultado é surpreendente!...

Para atravessar uma árvore binária, seja de pesquisa binária ou não, segue-se normalmente o seguinte algoritmo.

se a árvore for vazia

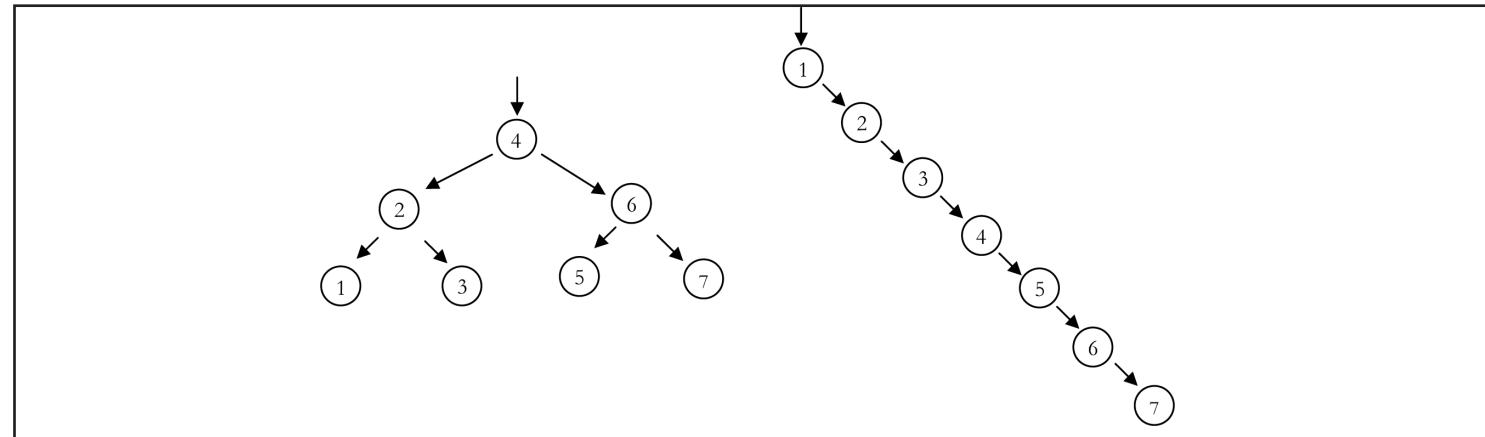
termina a visita

se não

atravessa a sub-árvore da esquerda

visita o nó da raiz

atravessa a sub-árvore da direita



Aplique manualmente este algoritmo às duas árvores de pesquisa binária indicadas na figura e analise o resultado. Provavelmente vai ficar surpreendido...

Pois é, em ambos os casos, vai chegar ao mesmo resultado. Todos os elementos aparecem ordenados!



Este tipo de atravessamento, pelo qual optámos, é designado por *in-order*, pois a visita ao nó da raiz encontra-se no meio do atravessamento das duas sub-árvores. Mas a visita daquele nó poderia ser, como alternativa, antes do atravessamento das duas sub-árvores (*pre-order*) ou depois (*post-order*)...

Sem desviar a sua atenção do que estava a fazer, e se tiver curiosidade, veja o que acontece no atravessamento das duas árvores quando segue a visita *pre-order* ou *post-order*...



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



### Exercício 4 - atravessar árvores binárias

Escreva em Scheme o procedimento `atravessa-arv`, com o parâmetro `arv`, uma árvore binária, baseado no algoritmo para atravessar árvores binárias (*in-order*).



Desenvolva e teste o procedimento `atravessa-arv`, que vai visualizando os elementos do nó por onde passa.



Se atravessar uma árvore de pesquisa binária, seja ou não equilibrada, vai obter uma sequência ordenada. Esta sequência ordenada parece ser a justificação para o tema que se segue.

**A árvore de pesquisa binária para ordenar listas... e listas ordenadas para equilibrar árvores de pesquisa binária!**

A ideia agora a explorar é o facto de, ao atravessar uma árvore de pesquisa, resultar uma sequência ordenada.

### Exercício 5 - ordenar listas através de árvores de pesquisa binária

Uma ideia para ordenar listas parece agora mais do que óbvia...

- criar uma árvore de pesquisa binária a partir da lista a ordenar (procedimento `lista->arv-pesq`)



Que funcionalidade da abstracção árvore binária vai necessitar para esta conversão?

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



- criar uma lista ordenada a partir da árvore de pesquisa binária (procedimento arv-pesq->lista, que deverá evidenciar alguma semelhança com o procedimento do exercício anterior atravessa-arv).

Desenvolva o procedimento ordena-lista-com-arvore com o parâmetro lis, uma lista a ordenar, baseado na ideia exposta.



Desenvolva e teste o procedimento ordena-lista-com-arvore e os respectivos procedimentos auxiliares.

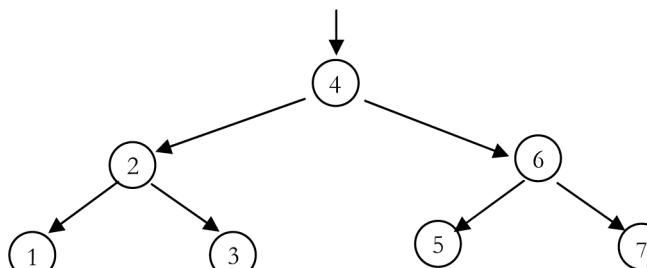
#### Exercício 6 - Equilibrar uma árvore de pesquisa binária através de uma lista ordenada

Uma ideia para equilibrar uma árvore de pesquisa binária não surge de uma maneira muito directa. Poderá começar por tentar alguma coisa do seguinte tipo...

- criar uma lista (que surge ordenada) a partir da árvore de pesquisa binária (procedimento arv-pesq->lista)



- a partir da lista ordenada obtida, criar uma árvore de pesquisa binária equilibrada



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Mas como realizar esta operação garantindo, no final, uma árvore de pesquisa binária equilibrada? Por que razão a árvore de pesquisa binária, que se deseja equilibrada, não será obtida por inserção dos elementos da lista ordenada, do primeiro até ao último?



Verifique que uma ordem de inserção dos elementos da lista ordenada numa árvore de pesquisa binária, para garantir o seu equilíbrio, começará pelo elemento central da lista (4), depois seguirá pelo elemento central da metade inicial da lista (2), depois pelo elemento central da metade final (6)...

Desenvolva o procedimento `equilibra-arvore` com o parâmetro `arv`, uma árvore de pesquisa binária, que devolve uma árvore de pesquisa binária equilibrada com os elementos da árvore `arv`.



Desenvolva e teste o procedimento `equilibra-arvore` e os respectivos procedimentos auxiliares.

Variante

A estrutura intermédia, em vez de ser uma lista ordenada, será agora um vector ordenado... Concorda com esta variante?

### Uma expansão à abstracção árvore binária

Sob a forma de um exercício, aparece mais alguma funcionalidade para juntar à abstracção árvore binária.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

R E 1 2 3 4 5

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



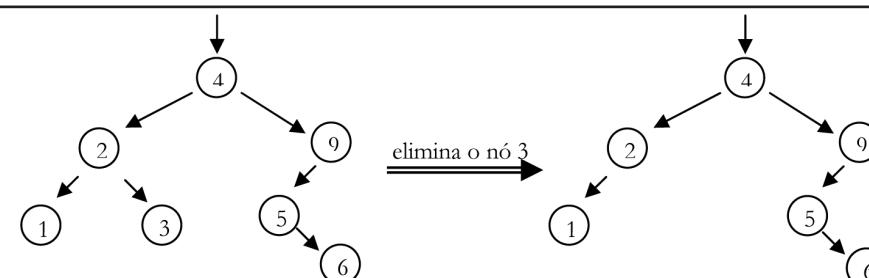
**Na funcionalidade indicada, apenas `tira-da-arv!` tem um parâmetro do tipo árvore de pesquisa binária. Estará bem ou terá sido uma falha? Justifique.**



Complete a abstracção árvore binária, desenvolvendo os procedimentos `numero-de-nos`, `peso` e `tira-da-arv!`.

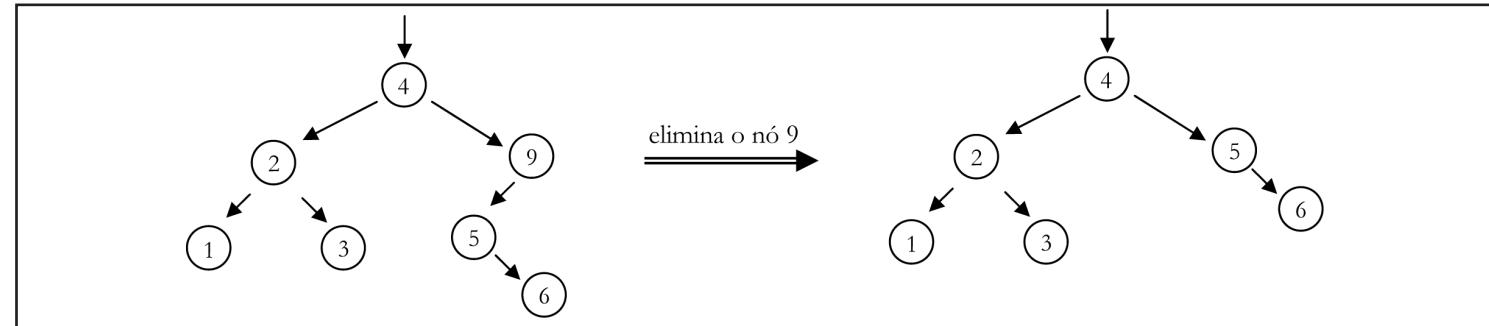
Pistas para eliminar um nó de uma árvore de pesquisa binária:

1- Se o nó a eliminar for uma folha, basta ir ao nó que é pai desta folha e colocar no ramo respectivo uma árvore vazia.

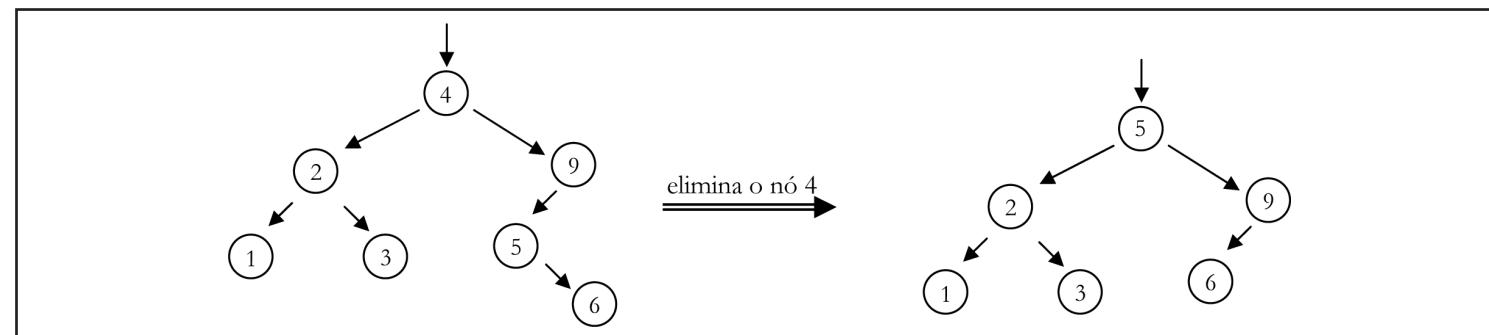


- 2- Se o nó a eliminar não for uma folha...

- se tiver apenas um filho... o nó é eliminado e o filho único é adoptado pelo avô (o pai do nó que foi eliminado).



- se tiver dois filhos... procura na sub-árvore da direita o menor nó, que vai tomar o lugar do nó a eliminar. Para isso toma o ramo da direita e depois sempre para a esquerda até ...
    - parar numa folha, que será o menor nó... e colocar árvore vazia na sub-árvore da esquerda do pai desta folha.
    - ou parar num nó que só tem um filho e é do lado direito... o nó onde parou é o menor nó e o seu filho do lado direito ocupará a sua posição (ver figura seguinte).



**Será que também funcionaria procurando no ramo da esquerda o maior nó? Justifique.**

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Mas também se pode ordenar sem utilizar árvores de pesquisa binária...

A literatura da especialidade é rica em algoritmos de ordenação, pois este é o tipo de problema que tem atraído sempre muitas pessoas com interesses nesta área. Alguns desses algoritmos são agora apresentados, por vezes através de uma simples ideia, ou então de uma forma um pouco mais rigorosa. As actividades que se propõem surgem como exercícios ou como exemplos para completar.

### Exemplo 1 - ordenação de listas por inserção

Considere uma lista de valores a ordenar e, a partir desta, deve ser criada uma nova lista com os seus elementos ordenados do menor para o maior, segundo a ideia que se segue:

- enquanto a lista a ordenar tiver elementos, toma o seu primeiro elemento, que é inserido na lista em construção, inicialmente vazia.
  - a inserção de um elemento na lista em construção deve ser feita na posição certa, para que a lista continue ordenada.
- repetir o passo anterior, mas a lista a ordenar será considerada já sem o primeiro elemento.

Implemente em Scheme esta ideia de ordenação em que a lista ordenada vai sendo criada por inserção dos elementos da lista não ordenada, um a um, até que esta lista fique completamente vazia.

```
> (ordena-lista (list 4 6 7 1 0 9 8 5 3 2))  
(0 1 2 3 4 5 6 7 8 9)  
> (ordena-lista (list 34 56 -3 4 57 56 33))  
(-3 4 33 34 56 56 57)  
>
```

Pista:

O procedimento `ordena-lista` tem uma lista como parâmetro, a lista a ordenar, e deve devolver uma nova lista correspondente à lista dada, mas devidamente ordenada.

O procedimento recursivo `ordena-lista-aux` tem duas listas como parâmetros, uma é a lista a ordenar e outra é a lista em construção, inicialmente vazia... A ideia em que se baseia este procedimento é simples: toma um elemento da lista não ordenada e coloca-o na posição certa na lista ordenada, ainda em construção. Depois toma outro colocando-o sempre na posição certa na lista em construção. Assim, quando se esgotar a lista não ordenada, a outra estará ordenada.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Da descrição anterior identifica-se o interesse do procedimento `poe-um-ordenado`, com dois parâmetros, novo e lista-ord. O parâmetro novo é um elemento que se pretende colocar na posição certa da lista ordenada em construção, aqui designada por lista-ord.

```
> (poe-um-ordenado 5 '(1 8 15))  
(1 5 8 15)  
> (poe-um-ordenado 5 '())  
(5)  
>  
  
(define poe-um-ordenado  
  (lambda (novo lista-ord)  
    ... ... ... para completar  
  );  
(define ordena-lista-aux  
  (lambda (lis lis-ord)  
    (if (null? lis)  
        lis-ord  
        (ordena-lista-aux (cdr lis)  
          (poe-um-ordenado (car lis)  
            lis-ord))))  
  );  
(define ordena-lista  
  (lambda (lista)  
    (ordena-lista-aux lista '())))
```



Complete e teste os procedimentos apresentados.



O procedimento `ordena-lista`, conjuntamente com os seus procedimentos auxiliares, apresenta-se como um construtor ou como um modificador? Justifique.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Os vectores suportam muito bem operações de modificação (com acesso através de índice e operações de modificação) e motivam o desenvolvimento de soluções de ordenação de carácter não construtivo, ou seja, a estrutura com os dados a ordenar é, ela própria, modificada e devolvida depois de ordenada.

### Exercício 8 - ordenação de vectores por inserção

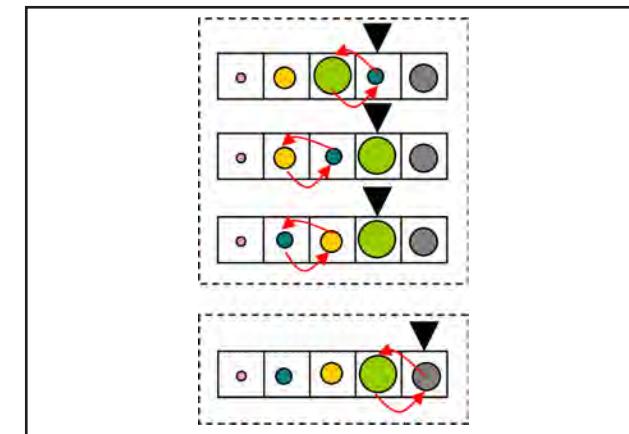
Temos, novamente, a ordenação por inserção, mas agora de uma forma diferente da apresentada anteriormente, pois os dados a ordenar são fornecidos num vector e o resultado é apresentado nesse mesmo vector, depois de ordenado. Trata-se portanto de uma solução modificadora.

A figura mostra já a parte final da ordenação.

A seta preta indica a posição onde se encontra a ordenação. Na parte superior da figura, o elemento do vector identificado pela seta preta vai ser comparado com o elemento que está à sua esquerda e, se este for maior, trocam entre si de posição.

E a comparação e troca repetem-se até que surja um elemento mais pequeno...

Repare que, no início das trocas, a parte do vector à esquerda da seta preta está ordenada...



Depois disso, a seta preta avança para a posição seguinte, indicando onde se encontra a ordenação.

1. defina um algoritmo de ordenação baseado na ideia esboçada;
2. escreva em Scheme o procedimento `ordena-por-insercao!` que se baseia no algoritmo definido em 1. Este procedimento aceita como argumento um vector, cujo conteúdo vai ser ordenado. Trata-se, portanto, de um procedimento modificador.
3. indique e justifique a ordem de crescimento em termos de tempo e de espaço da solução apresentada.

Pista para a ordem de crescimento em relação ao tempo:

$$1 + 2 + \dots + (n-2) + (n-1) = (n-1) * [(n-1) + 1]/2 = (n-1) * n/2$$



Desenvolva e teste o procedimento `ordena-por-insercao!`.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

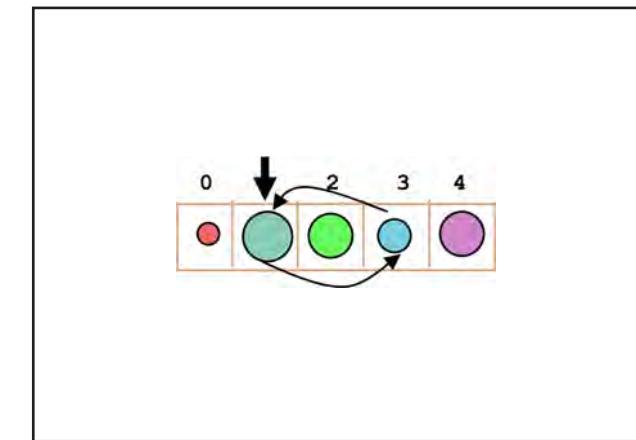


### Exercício 9 - ordenação de vectores por selecção (*selection sort*)

Uma forma muito simples, mas muito ineficaz para ordenar um vector, do menor para o maior elemento, pode definir-se sobre a ideia que se segue.

Supondo um vector v de n posições, com índices de 0 a n-1:

- procura o menor valor de v, desde o índice 0 ao índice n-1;
- troca esse valor com o que se encontra na posição de índice 0;
- procura o menor valor de v, desde o índice 1 ao índice n-1;
- troca esse valor com o que se encontra na posição de índice 1;
- procura o menor valor de v, desde o índice 2 ao índice n-1;
- ...
- termina quando a procura se reduzir ao valor situado na posição de índice n-1.



1. defina um algoritmo de ordenação baseado na ideia esboçada;
2. escreva em Scheme o procedimento `ordena-por-seleccao!` que se baseia no algoritmo definido em 1. Este procedimento aceita como argumento um vector, cujo conteúdo vai ser ordenado. Trata-se, portanto, de um procedimento modificador.
3. indique e justifique a ordem de crescimento em termos de tempo e de espaço da solução apresentada.



Desenvolva e teste o procedimento `ordena-por-seleccao!`.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



### Exercício 10 - ordenação de vectores por bolha (*bubble sort*)

Uma forma simples, mas também muito ineficaz para ordenar, é a ordenação por bolha. O vector a ordenar (em ordem crescente) vai ser percorrido tantas vezes quantos os seus elementos menos 1 e, em cada percurso, o maior valor encontrado é arrastado para a posição de índice mais elevado, ainda não ordenada...

- no primeiro percurso, do índice 0 até ao índice  $n-1$ , os respectivos valores são comparados aos pares e são trocados entre si se o de menor índice for maior.

Assim, começam por ser comparados os valores situados nas posições de índice 0 e 1 e o maior deles é puxado para a posição 1.

Na próxima comparação estarão presentes os valores agora situados nas posições de índice 1 e 2, e o maior deles vai ser puxado para a posição de índice 2...

Quando se compararem os dois valores, situados nas posições de índice  $n-2$  e  $n-1$ , um deles será o maior valor do vector. E o maior será puxado para a posição de índice  $n-1$ .

- no segundo percurso, as comparações começam novamente com os valores situados nas posições de índice 0 e 1 e terminam com o par  $n-3$  e  $n-2$ , pois na posição  $n-1$  já está o maior valor do vector. Da comparação do par  $n-3$  e  $n-2$  resulta o segundo maior valor do vector que será puxado para a posição  $n-2$ ...
- Termina a ordenação, após  $n-1$  percursos do vector.

1. defina um algoritmo de ordenação baseado na ideia esboçada;
2. escreva em Scheme o procedimento `ordena-por-bolha!` que se baseia no algoritmo definido em 1. Este procedimento aceita como argumento um vector, cujo conteúdo é devolvido depois de ordenado.  
Trata-se, portanto, de um procedimento modificador;
3. indique e justifique a ordem de crescimento em termos de tempo e de espaço da solução apresentada.





Se continuasse, o que aconteceria a todos os elementos?

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



1. defina um algoritmo recursivo para ordenar um vector de números inteiros positivos, do menor para o maior, de acordo com a ideia apresentada.
2. escreva em Scheme o procedimento `ordena-por-balde!` que se baseia no algoritmo definido em 1. Este procedimento aceita como argumento um vector, cujo conteúdo é devolvido depois de ordenado. Trata-se, portanto, de um procedimento modificador.
3. indique e justifique a ordem de crescimento em termos de tempo e de espaço da solução apresentada.



Desenvolva e teste o procedimento `ordena-por-balde!`.

### Exercício 12 - Ordenação rápida de vectores (*quick sort*)

Mais um algoritmo de ordenação, mas este muito eficiente.

- Passo de partição
  - Se o vector a ordenar só tem um elemento, o problema está resolvido... Trata-se do caso base.
  - Se o vector tem mais do que um elemento, toma o seu primeiro elemento e encontra a respectiva posição no vector final.

Essa posição encontra-se quando todos os valores à esquerda (ordenados ou não) são menores do que o elemento a ordenar, e todos os elementos à direita (ordenados ou não) são maiores do que ele. Neste momento temos um elemento correctamente colocado e dois sub-vectores (um à esquerda e outro à direita) normalmente não ordenados.

- Passo recursivo
    - Aplica recursivamente o passo de partição a cada um dos sub-vectores.

Pista: Considere o vector `n0rd` carregado com os valores 8, 9, 2, 5, 4, 6, 1, 3 e 7.

Como colocar o primeiro elemento, o valor 8, na posição correcta, deixando à esquerda os valores menores e à direita os maiores?

```

        i           s
        8   9   2   5   4   6   1   3   7
enquanto i < s
    1- se nOrd[i] > nOrd[i + 1] trocam entre si nOrd[i] e nOrd[i+1], e i = i + 1
    2- se nOrd[i] < nOrd[i + 1] trocam entre si nOrd[i+1] e nOrd[s], e s = s - 1

```



Aplique estas regras ao valor 8, no vector nOrd, enquanto  $i < s..$

1. o procedimento `particao!` tem como parâmetros um vector `vec` e os índices `inf` e `sup`, que definem um sub-vector de `vec`. Este procedimento implementa a ideia indicada na pista e ainda devolve o índice da posição onde fica o primeiro elemento do sub-vector. Escreva `particao!`.
  2. indique e justifique a ordem de crescimento em termos de tempo e de espaço da solução apresentada.

```
(define ordena-rapido!
  (lambda (vec inf sup)
    (if (< inf sup)
        (let ((posicao (particao! vec inf sup))
              (ordena-rapido! vec inf (sub1 posicao))
              (ordena-rapido! vec (add1 posicao) sup)))
          (ordena-rapido! vec inf sup)))
  (define particao!
    (lambda (vec i s)
      ... ... ... para completar.
```



Complete e teste o código relativo à ordenação rápida de vectores

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

### Exercício 13 - Ordenação por mistura de vectores (*merge sort*)

Mais um algoritmo de ordenação de uma tão longa série de algoritmos e um dos mais eficientes.

A implementação que se sugere é recursiva.

A ordenação recursiva de um vector tem como:

- Caso base

Um vector com um só elemento está ordenado.

- Passo recursivo

- Dividir o vector em dois sub-vectores de igual comprimento (se o comprimento do vector inicial for ímpar, um dos sub-vectores terá um elemento a mais que o outro);
- ordenar recursivamente cada um dos sub-vectores;
- misturar os dois sub-vectores ordenados, fundindo-os num único vector ordenado.

A mistura de 2 sub-vectores ordenados, fundindo-os num vector único, consiste no seguinte:

- comparam-se repetidamente os 2 elementos mais pequenos, um de cada sub-vector;
- o mais pequeno é colocado no vector em constituição.

Exemplo

Vector A: 37 45 98 103

Vector B: 23 145 234 765 1022

Os 2 elementos mais pequenos são 37 e 23, portanto o primeiro elemento do vector em constituição será 23. Os 2 elementos mais pequenos que se seguem serão 37 e 145...

1. escreva em Scheme o procedimento `merge-sort!` que se baseia na solução recursiva apresentada.

```
> (define vec3 (vector 245 34 3 21 -4 32 4 34))  
> vec3  
#(245 34 3 21 -4 32 4 34)  
> (merge-sort! vec3)  
> vec3  
#(-4 3 4 21 32 34 34 245)  
>
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



O procedimento `merge-sort!` utiliza o procedimento `merge-sort-subvector!`, para ordenar um subvector. Por seu turno, `merge-sort-subvector!` utiliza os procedimentos auxiliares `vector-copy` e `merge!`, como pode ver mais à frente.

2. indique e justifique a ordem de crescimento em termos de tempo e de espaço da solução apresentada.

```
; ordena o vector vec em ordem crescente
; segundo o algoritmo merge-sort
;
(define merge-sort!
  (lambda (vec)
    (merge-sort-subvect! vec 0 (sub1 (vector-length vec)))))

; ordena o vector v, das posições inf a sup
;
(define merge-sort-subvect!
  (lambda (v inf sup)
    (if (> (- sup inf) 0)
        (let* ((v-aux (vector-copy v inf sup))
               (comprimento-um (sub1 (vector-length v-aux)))
               (meio (quotient comprimento-um 2)))
          (merge-sort-subvect! v-aux 0 meio)
          (merge-sort-subvect! ... ... ... para completar )
          (merge! v inf v-aux 0 meio ... ... ... para completar ))
        ;
        ; ----- procedimentos auxiliares -----
        ;
        ; devolve um vector cujo conteúdo é
        ; o vector vec-source das posições de inf a sup
        ;
(define vector-copy
  (lambda (vec-source inf sup)
    (letrec((vec (make-vector (add1 (- sup inf)) 0))
           ;
           (ciclo
             (lambda (i-s i-d)
               ... ... ... para completar.
           ;
           (ciclo inf 0)))
    ;
    ; faz o merge de 2 vectores ordenados:
    ; vector v1 (das posições i1 a s1) e vector v2 (das posições i2 a s2)
    ; num vector único ordenado vcomb a partir da posição icom
    ;
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

R E 1 2 3 4 5

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
(define merge!
  (lambda (vcomb icomb v1 i1 s1 v2 i2 s2)
    (cond ((and (>= (- s1 i1) 0) (>= (- s2 i2) 0))
            (if (<= (vector-ref v1 i1) (vector-ref v2 i2))
                (begin (vector-set! vcomb icomb (vector-ref v1 i1))
                      (merge! vcomb (add1 icomb) v1 (add1 i1) s1 v2 i2 s2)))
                (begin (vector-set! vcomb icomb (vector-ref v2 i2))
                      (merge! vcomb (add1 icomb) v1 i1 s1 v2 (add1 i2) s2))))
            ((>= (- s1 i1) 0)
             ... ... ... para completar.
            ((>= (- s2 i2) 0)
             ... ... ... para completar.

> (define vec1 (vector 37 45 98 103))
> (define vec2 (vector 23 145 234 765 1022))
> (define vec10 (make-vector 9 0))
> vec10
(0 0 0 0 0 0 0 0 0)
> (merge! vec10 0 vec1 0 3 vec2 0 4)
> vec10
#(23 37 45 98 103 145 234 765 1022)
>
```



Complete e teste o código relacionado com a ordenação por mistura.  
Sugere-se que comece por completar e testar os procedimentos `merge!` e `vector-copy!`.

Para efeitos de comparação das suas respostas sobre ordem de crescimento dos algoritmos de ordenação, apresenta-se essa grandeza, em relação ao tempo, para os principais algoritmos tratados.

Inserção	$O(n^2)$
Selecção	$O(n^2)$
Borbulha	$O(n^2)$
Rápido	$O(n^2)$ - pior situação; $O(n * \log_2 n)$ - em média
Mistura	$O(n * \log_2 n)$ .

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



## Resumo dos módulos 6.1 a 6.5

Resumo dos assuntos abordados nos módulos

Módulo 6.1 - Interface gráfica - GUI (Graphical User Interface)

Módulo 6.2 - Criar jogos... em Scheme

Módulo 6.3 - Testes unitários

Módulo 6.4 - Código executável criado a partir de programas Scheme

Módulo 6.5 - A programação OO com o Scheme

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Resumo dos assuntos abordados nos módulos 6.1 a 6.5

Neste conjunto de módulos, o objectivo principal é mostrar que o Scheme não é só aquela linguagem essencialmente usada para aprender a programar. Isso fica provado com: o desenvolvimento de interfaces gráficas (GUI) para os programas Scheme; o desenvolvimento de programas multimédia, em especial jogos; o teste ao funcionamento do código desenvolvido, através de testes unitários, sempre prontos a ajudar na identificação de falhas; a geração de um código executável a partir de programas Scheme, permitindo a execução desse código em computadores que não tenham Scheme instalado; e a demonstração da flexibilidade do Scheme para introduzir conceitos do paradigma de programação orientada por objectos, tanto do ponto de vista da implementação desses conceitos como do respectivo uso.

### Módulo 6.1

A utilização de programas em Scheme por pessoas que não conheçam esta linguagem, pode ser facilitada através de interfaces gráficas. A construção de interfaces gráficas é o objectivo deste módulo.

Numa primeira aproximação à construção de interfaces gráficas, utilizaremos uma biblioteca muito simples, `gui.ss`, que é a base do *teachpack GUI* do DrScheme.

Numa segunda aproximação, encontraremos uma plataforma de desenvolvimento um pouco mais sofisticada, MrEd Designer, que permite definir e construir interfaces através de um editor gráfico. É também apresentada uma forma de ultrapassar algumas limitações desta plataforma.

### Módulo 6.2

Os jogos de computador são um dos produtos informáticos com maior sucesso. Ao longo deste módulo será feita uma introdução ao desenvolvimento de jogos de computador, utilizando recursos disponibilizados pela linguagem Scheme. Com base num dos jogos dos primórdios da indústria informática - o Pong - será desenvolvido um caso de estudo que permitirá descobrir as diversas fases da concepção de um jogo de computador. Tome pois atenção, pois, no final deste módulo, como desafio, ser-lhe-á pedido o desenvolvimento de um jogo de computador....

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Quase todos os jogos de computador são aplicações multimédia. O Scheme permite o desenvolvimento de aplicações multimédia através do módulo Allegro.plt, parte integrante da distribuição PLT Scheme. Paralelamente à abordagem ao desenvolvimento dos jogos electrónicos, será feita uma descrição das principais funcionalidades disponibilizadas por este módulo, que se encontra documentado em [Allegro: Multimedia library and game utilities](#).

### Módulo 6.3

Testar o código que se produz, de uma forma sistemática, é uma actividade que nem sempre tem merecido a atenção que lhe é devida. No decurso dos nossos trabalhos, temos recomendado que, ao analisar um problema, deve ser preparado, manualmente, um conjunto de entradas de teste que o ajudarão a melhor entender esse problema, servindo também, mais tarde, para testar o código desenvolvido. Muitas vezes, por comodismo, nem se dispensa um pouco de tempo para experimentar todas as entradas de teste anteriormente preparadas. Um dia, se esse código sofrer alguma alteração, aquele conjunto de entradas de teste ou já foi esquecido ou a paciência não é suficiente para o aproveitar.

Os testes unitários são programas preparados normalmente antes de ser desenvolvido o código que irão testar, e guardados num ficheiro cujo nome mostre bem que é o teste unitário desse código. E depois, quando o código acaba de ser desenvolvido pela primeira vez, ou após qualquer actualização, grande ou pequena, bastará executar o teste unitário que lhe está associado. Da execução do teste unitário resulta informação que permite localizar rapidamente os pontos do código em teste que exijam intervenção...

O Scheme permite o desenvolvimento de testes unitários através, por exemplo, da biblioteca SchemeUnit.plt, parte integrante da distribuição PLT Scheme, documentada em [SchemeUnit: Unit Testing in Scheme](#).

Neste Módulo é feita uma abordagem à utilização da biblioteca SchemeUnit, ilustrada com vários exemplos. Haverá também espaço quer para uma breve incursão ao código que preveja o lançamento de exceções, quer para os testes unitários respectivos.

### Módulo 6.4

Os programas que tem encontrado, sejam eles muito simples ou mais complexos, com interface textual ou gráfica, com ou sem interacção, têm sido sempre experimentados no ambiente do Scheme. Tudo se tem passado, necessariamente, num computador onde o Scheme esteja instalado.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Certamente que já passou por alguma situação em que pretendeu enviar um dos seus programas mais interessantes a um amigo ou a um familiar mas, apesar do acesso ao Scheme ser livre, o seu amigo ou familiar pode não ter instalado, nem querer instalar, o Scheme no computador.

A ideia deste pequeno módulo é apresentar a forma de ultrapassar este tipo de situação e mostrar como o DrScheme permite produzir um código executável que correrá em quaisquer computadores, mesmo nos que não tenham o Scheme instalado. Esta possibilidade, como verá, aplica-se não só a pequenos programas, mas também a programas mais complexos, com interface gráfica, como a navegação em labirintos e até jogos...

Neste pequeno módulo é indicada a forma de criar código executável, a partir do DrScheme, sempre ilustrada com vários exemplos de complexidade muito variada.

### Módulo 6.5

O grande interesse da programação OO reside na modularidade que disponibiliza, pois cada objecto surge como um bloco, o qual protege ou encapsula os seus atributos, condicionando o respectivo acesso através das suas operações preparadas para esse efeito. Outro interesse da programação OO está na facilidade do reaproveitamento do seu código, provavelmente até desenvolvido por equipas ou programadores diversos.

O Scheme não é uma linguagem vocacionada para a programação orientada por objectos, como acontece com linguagens como o Java, C++, Smalltalk ou Simula, mas permite, com alguma facilidade, a implementação dos principais conceitos deste importante paradigma de programação. Terá assim a oportunidade de contactar com a implementação dos conceitos de classe, objecto, mensagem, operação, herança, entre outros, isto numa primeira fase. Só depois passará à respectiva utilização em vários exemplos e exercícios. Ora, não é isso que acontece com aquelas linguagens, onde apenas se limitará a usar esses conceitos, pois já os encontrará implementados...

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Módulo 6.1 - Interface gráfica - GUI (Graphical User Interface)

A utilização de programas em Scheme por pessoas que não conheçam esta linguagem pode ser facilitada através de interfaces gráficas. A construção de interfaces gráficas é o objectivo deste módulo.

Numa primeira aproximação à construção de interfaces gráficas, utilizaremos uma biblioteca muito simples, `gui.ss`, que é a base do *teachpack GUI* do DrScheme.

Numa segunda aproximação, encontraremos uma plataforma de desenvolvimento um pouco mais sofisticada, MrEd Designer, que permite definir e construir interfaces através de um editor gráfico. É também apresentada uma forma de ultrapassar algumas limitações desta plataforma.

### Palavras-Chave

GUI, interface gráfica.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Uma primeira abordagem às interfaces gráficas

No exemplo do procedimento factorial, surge, mais uma vez, a interface textual que nos tem acompanhado até aqui.

```
(define factorial
  (lambda (n)
    (if (zero? n)
        1
        (* n (factorial (sub1 n))))))
```

```
> (factorial 2)
2
> (factorial 5)
120
>
```

interface textual

A utilização deste procedimento pode ser facilitada, sobretudo a quem não conheça a linguagem Scheme, quando se recorre a uma interface gráfica. Para este efeito, vamos começar por utilizar a biblioteca `gui.ss` que constitui a base do *teachpack GUI* do DrScheme.



A sequência de figuras, da esquerda para a direita, mostra a evolução de uma interface gráfica, desde o estado inicial até ao estado final.

Em cada uma destas figuras, a primeira linha representa um campo de texto que sugere ao utilizador a entrada de um valor e a segunda mostra uma mensagem onde vai surgir o factorial desse valor. Na parte inferior, dois botões mostram claramente a funcionalidade associada a cada um deles.





Teste a interface gráfica que dá um acesso fácil ao procedimento factorial.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

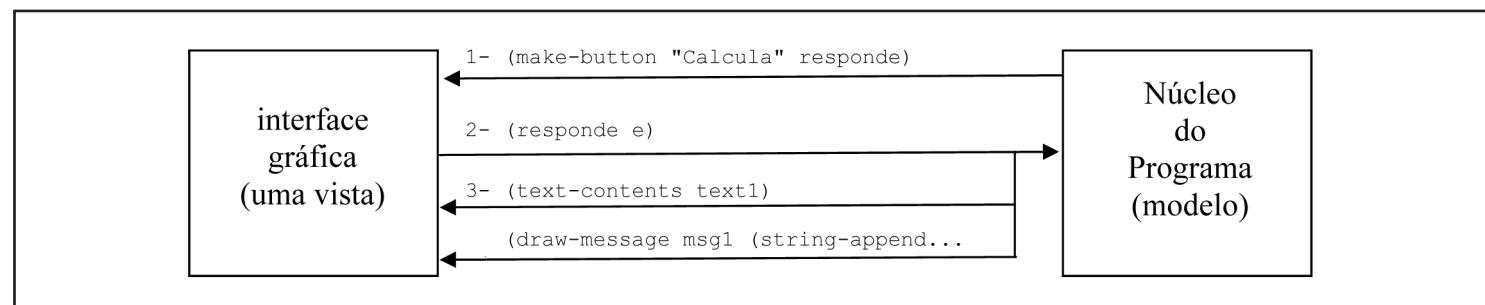
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Neste exemplo, o procedimento `factorial` é o núcleo do programa e a interface apresentada é apenas uma das possíveis interfaces para este programa. De facto, um programa poderá ter várias interfaces. Podemos dizer que o programa pode ter várias vistas.

A interface cabe a visualização da informação e a gestão do rato e do teclado, dispositivos que o utilizador actuará para exprimir, da forma mais natural possível, o que pretende.



No essencial, a figura mostra a funcionalidade associada ao processo de gestão da interface gráfica:

- 1- a construção de um elemento da interface (botão `Calcula`) e a associação da respectiva função de resposta ou *call-back* (função `responde`);
- 2- a actuação, pelo utilizador, de um elemento da interface (botão `Calcula`), evento que provoca a chamada da respectiva função de resposta ou *call-back* (função `responde`);
- 3- a função de resposta (função `responde`) que requer informação a outro elemento da interface (campo de texto `text1`) e envia informação através de um outro elemento da interface (campo de mensagem `msg1`).

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Analise com cuidado a implementação que se segue.

Tome atenção aos comentários e, no final, prepare-se para responder a algumas perguntas...

```
; inclusão de gui.ss do teachpack GUI do DrScheme
(require (lib "gui.ss" "htdp"))

; núcleo do programa: neste caso, resume-se ao procedimento factorial
(define factorial
  (lambda (n)
    (if (zero? n)
        1
        (* n (factorial (sub1 n))))))

; campo de texto que recebe o próximo argumento do procedimento factorial
(define text1
  (make-text "Factorial de "))

; mensagem onde se coloca o resultado
(define msg1
  (make-message (string-append "é " (make-string 20 #\SPACE)))))

; função de resposta/call-back do botão Calcular
; a mensagem é previamente formada apenas por "é " e por 20 caracteres #\SPACE...
; Estes 20 caracteres são substituídos, na resposta, por uma cadeia de caracteres
; Esta cadeia de caracteres é obtida convertendo o valor numérico resultante
; da chamada de factorial.
; O argumento usado na chamada de factorial é procurado no campo de texto...
(define responde
  (lambda (e)
    (draw-message msg1 (string-append "é "
                                      (number->string
                                        (factorial (string->number
                                          (text-contents text1)))))))

; criação e visualização de uma janela, composta por três linhas:
; - um campo de texto
; - uma mensagem
; - dois botões
;
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MÚTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
(define w
  (create-window
    (list
      (list text1) ; a primeira linha,
      (list msg1) ; a segunda linha ... e agora
      ; ; a terceira linha, composta por 2 elementos
      (list (make-button "Calcula" responde)
            (make-button "Termina" (lambda (e) (hide-window w)))))))
```

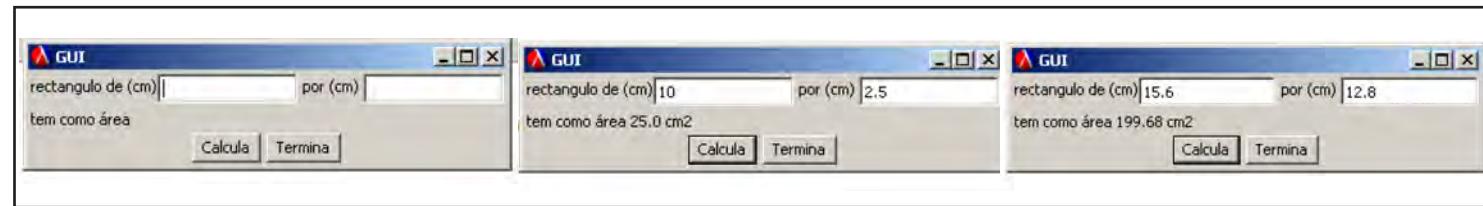


### Perguntas sobre esta implementação:

- 1- na definição da mensagem `msg1` o que garantirá a cadeia de caracteres formada por 20 `\SPACE?`
- 2- na definição da janela, como se apresentaria a interface se os elementos `text1` e `msg1` fossem colocados na mesma lista, ou seja, `(list text1 msg1)`?
- 3- o que tem a dizer sobre o segundo argumento de `make-button`?

### Exercício 1

Desenvolva uma interface gráfica para calcular a área de rectângulos, sugerindo-se que ela se apresente conforme é indicado na figura.



Desenvolva e teste a interface gráfica que dá acesso ao cálculo da área de rectângulos.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Os vários tipos de elementos da interface gráfica `gui.ss`

A biblioteca `gui.ss` do *teachpack GUI* do DrScheme, para além de

- botão (*button*), criado com

`(make-button a-string a-function) -> button-gui-item`

chama a função de resposta associada ao botão quando o utilizador o actua com o rato. A função de resposta tem um parâmetro e devolve um booleano;

- campo de texto (*text field*), criado com

`(make-text a-string) -> text-gui-item`

permite a entrada de texto numa determinada área da janela;

- campo de mensagem (*message field*), criado com

`(make-message a-string) -> message-gui-item`

permite a visualização de um texto, numa determinada área da janela;

disponibiliza ainda o

- menu de escolha (*choice menu*), criado com

`(make-choice a-list-of-strings) -> choice-gui-item`

permite a escolha de uma opção entre várias.

Outra funcionalidade importante disponibilizada por `gui.ss`:

- `(create-window (list (list gui-item) (list gui-item) ...))`

`-> window-gui-item`

cria uma janela gráfica com elementos de interface e visualiza a janela criada.

Cada `(list gui-item)` define uma linha de elementos de interface na janela;



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

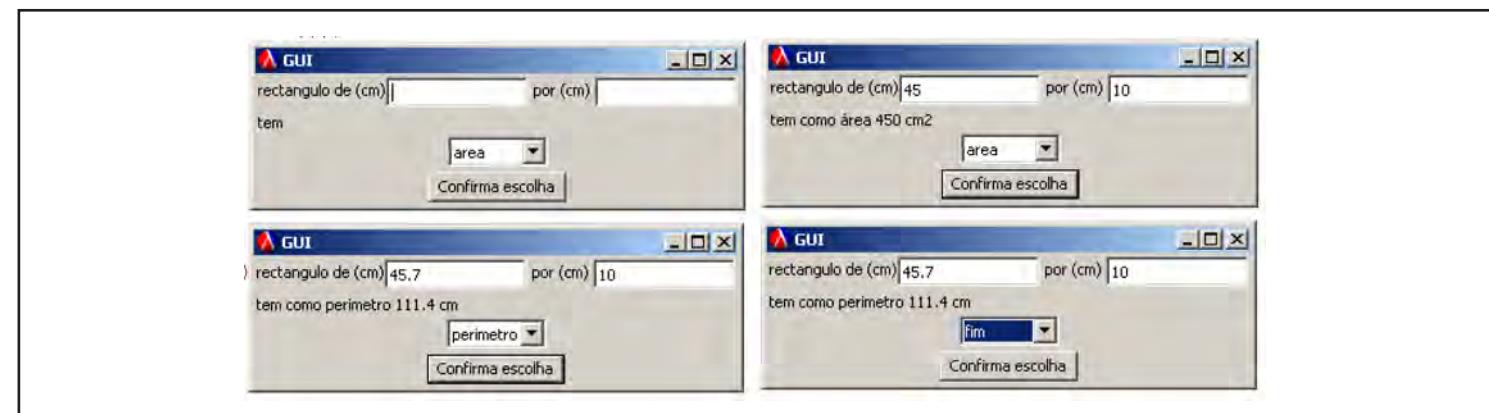
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

- (show-window window-gui-item) -> #t  
mostra a janela;
- (hide-window window-gui-item) -> #t  
esconde a janela;
- (draw-message message-gui-item a-string) -> #t  
visualiza a cadeia de caracteres na mensagem especificada;
- (text-contents text-gui-item) -> a-string  
devolve o conteúdo do campo texto especificado;
- (choice-index choice-gui-item) -> number  
devolve a escolha correntemente seleccionada no menu, correspondendo à primeira escolha o valor 0, à segunda o valor 1, ... e assim sucessivamente até à última escolha.

### Exemplo 1

Desenvolva uma interface gráfica para calcular a área e o perímetro de rectângulos.



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(require (lib "gui.ss" "htdp"))

(define area-rectangulo
  (lambda (n m)
    (* n m)))

(define perimetro-rectangulo
  (lambda (n m)
    (* 2 (+ n m)))))

; text : GUI-ITEM
(define text1
  (make-text "rectangulo de (cm)"))
(define text2
  (make-text "por (cm)"))

; msg1 : GUI-ITEM
(define msg1
  (make-message (string-append "tem " (make-string 33 #\SPACE))))

; Event -> true
(define respond-1
  (lambda ()
    (draw-message
      msg1 (string-append
        "tem como área "
        (number->string (area-rectangulo (string->number (text-contents text1))
                                         (string->number (text-contents text2))))
        " cm2"))))

(define respond-2
  (lambda ()
    (draw-message
      msg1 (string-append
        "tem como perimetro "
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
(number->string
  (perimetro-rectangulo (string->number (text-contents text1))
    (string->number (text-contents text2))))
  " cm")))
(define respond-3
  (lambda ()
    (hide-window w)))

(define as-escolhas
  (list "area" "perimetro" "fim"))

(define as-operacoes
  (list respond-1 respond-2 respond-3))

(define uma-escolha
  (make-choice as-escolhas))

(define uma-mensagem
  (make-message (car as-escolhas)))

(define responde
  (lambda (e)
    ((list-ref as-operacoes (choice-index uma-escolha)))))

;

(define w
  (create-window
    (list
      (list text1 text2)
      (list msg1)
      (list uma-escolha)
      (list (make-button "Confirma escolha" responde)))))
```



Teste a interface gráfica que dá acesso ao cálculo da área e perímetro de rectângulos.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exemplo 2

Desenvolva uma interface gráfica que permita simular o lançamento de um dado (de 6 faces) um número de vezes à escolha, a fim de determinar a frequência da ocorrência de cada uma das suas faces.



```
; inclusão da biblioteca gui
(require (lib "gui.ss" "htdp"))

; campo de texto que recebe o número de lançamentos
(define text1
  (make-text "Quantos lançamentos"))

; 6 mensagens independentes onde se coloca o resultado
(define msg1
  (make-message (string-append "1 - " (make-string 8 #\SPACE))))
(define msg2
  (make-message (string-append "2 - " (make-string 8 #\SPACE))))
(define msg3
  (make-message (string-append "3 - " (make-string 8 #\SPACE))))
(define msg4
  (make-message (string-append "4 - " (make-string 8 #\SPACE))))
(define msg5
  (make-message (string-append "5 - " (make-string 8 #\SPACE))))
(define msg6
  (make-message (string-append "6 - " (make-string 8 #\SPACE)))))

; resposta dada através de 6 mensagens...
```

# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



```
(define responde
  (lambda (e)
    (let* ((num-lancamentos (string->number (text-contents text1)))
           (vec (lancar-dado (if (and (number? num-lancamentos)
                                         (> num-lancamentos 0))
                                 num-lancamentos
                                 0))))
        ;
        (draw-message msg1 (string-append "1 - "
                                         (number->string (vector-ref vec 0)))))
        (draw-message msg2 (string-append "2 - "
                                         (number->string (vector-ref vec 1)))))
        (draw-message msg3 (string-append "3 - "
                                         (number->string (vector-ref vec 2))))
        (draw-message msg4 (string-append "4 - "
                                         (number->string (vector-ref vec 3))))
        (draw-message msg5 (string-append "5 - "
                                         (number->string (vector-ref vec 4))))
        (draw-message msg6 (string-append "6 - "
                                         (number->string (vector-ref vec 5))))))
    ;
    (define w
      (create-window
        (list
          (list text1)
          (list (make-button "Lança" responde)
                (make-button "Termina" (lambda (e) (hide-window w)))))
        (list msg1 msg4)
        (list msg2 msg5)
        (list msg3 msg6)))))

(define lancar-dado
  (lambda (num-vezes)
    (let ((conta-faces (make-vector 6 0)))
      (letrec ((aux
                (lambda (n-vezes)
                  (if (zero? n-vezes)
                      conta-faces
                      (let ((face-menos-1 (sub1 (roleta-1-6))))
                        (vector-set! conta-faces
                                      face-menos-1
                                      (add1 (vector-ref conta-faces
                                                       face-menos-1)))
                        (aux (sub1 n-vezes)))))))
        ;
        (aux num-vezes)))))

(define roleta-1-6      ; gera e devolve um número aleatório entre 1 e 6
  (lambda()
    (add1 (random 6))))
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Teste a interface gráfica que permite a simulação do lançamento de um dado.

Experimente substituir partes do código testado pelo código que se segue e verifique se tudo continua a funcionar cor-  
rectamente. Compare as duas soluções e indique a que prefere.

```
; uma lista de 6 mensagens...
(define msgs
  (list (make-message (string-append "1 - " (make-string 8 #\SPACE)))
        (make-message (string-append "2 - " (make-string 8 #\SPACE)))
        (make-message (string-append "3 - " (make-string 8 #\SPACE)))
        (make-message (string-append "4 - " (make-string 8 #\SPACE)))
        (make-message (string-append "5 - " (make-string 8 #\SPACE)))
        (make-message (string-append "6 - " (make-string 8 #\SPACE)))))

(define responde
  (lambda (e)
    (let* ((num-lancamentos (string->number (text-contents text1)))
           (vec (lancar-dado (if (and (number? num-lancamentos)
                                       (> num-lancamentos 0))
                                 num-lancamentos
                                 0))))
           (letrec ((ciclo
                     (lambda (x)
                       (cond ((< x 6)
                             (draw-message (list-ref msgs x)
                                         (string-append
                                           (number->string (add1 x))
                                           " - "
                                           (number->string (vector-ref vec x))))
                             (ciclo (add1 x)))
                           (else #t))))
                     (ciclo 0))))
         (ciclo w)
         (create-window
           (list
             (list text1)
             (list (make-button "Lança" responde)
                   (make-button "Termina" (lambda (e) (hide-window w)))))
             (list (list-ref msgs 0) (list-ref msgs 3))
             (list (list-ref msgs 1) (list-ref msgs 4))
             (list (list-ref msgs 2) (list-ref msgs 5)))))))
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

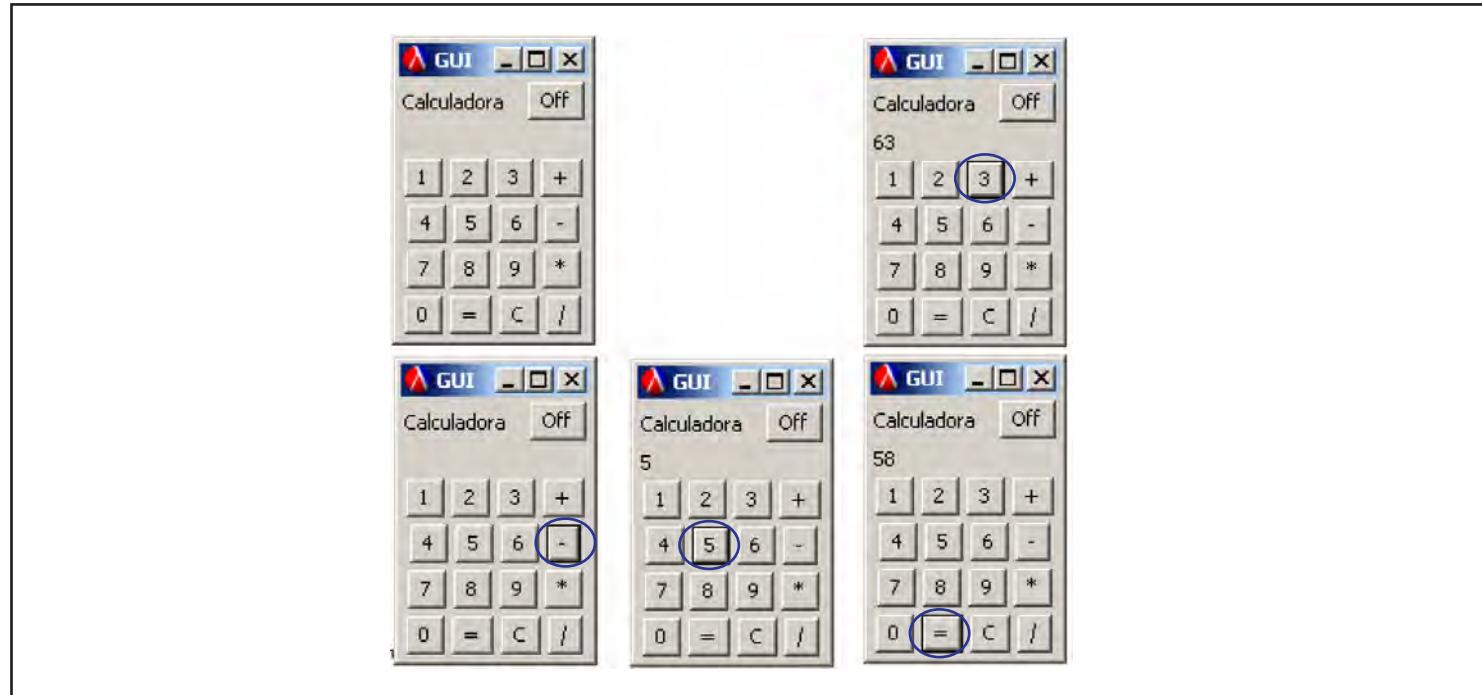
R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Exercício 2

Desenvolva uma interface gráfica que permita simular uma calculadora simples, como se indica na sequência de figuras.



Normalmente, é indicado um primeiro número seguido de uma operação. Nesse momento, tanto o número como a operação são memorizados e o visualizador volta a estar limpo. Um segundo número seguido da tecla = fará aparecer o resultado no visualizador. O botão Off termina a visualização da calculadora.



Desenvolva e teste a interface gráfica que simula a calculadora.



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Construtores de interfaces gráficas

Certamente verificou, nos exemplos e exercícios que envolviam a biblioteca `gui.ss`, a necessidade de um esforço não desprezável, e até um tanto ou quanto repetitivo, exigido pela implementação das interfaces. Há, no entanto, algumas plataformas cujo objectivo é minimizar este esforço de programação. Com estas plataformas de construção de interfaces gráficas, as *gui builders*, o custo do desenvolvimento de uma interface pode resumir-se, quase exclusivamente, ao esforço da sua especificação. Em algumas destas plataformas, a especificação pode até beneficiar do apoio de um editor gráfico para estabelecer a relação entre os vários elementos da interface em construção. É este o caso do MrEd Designer que pode encontrar em:

<http://hexahedron.hu/personal/peteri/mreddesigner/index.html>

Depois de descomprimir o ficheiro disponível, copie o directório `mreddesigner` para o directório `\PLT\` do DrScheme.

Agora, em `\PLT\mreddesigner`, desloque o subdirectório `images` para o directório `\PLT\` do DrScheme.

A partir deste momento, com

```
(require (file "mreddesigner\\mreddesigner.ss"))
```

passa a ter acesso ao MrEd Designer...

O ambiente disponibilizado pelo MrEd Designer é relativamente intuitivo e quase dispensa a consulta do respectivo manual.



Para utilizar o MrEd Designer, o DrScheme deve ser inicialmente aberto através do sistema operativo do seu computador. Se tiver dúvidas sobre esta situação, o melhor será seguir a sequência que se descreve:

- 1- Faça **Exit** no DrScheme, se este estiver activo, garantindo que não fica qualquer sessão aberta;
- 2- Inicie o DrScheme através do sistema operativo que está a utilizar;
- 3- Agora, já pode utilizar o MrEd Designer, mesmo que seja através de um dos laboratórios Scheme.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

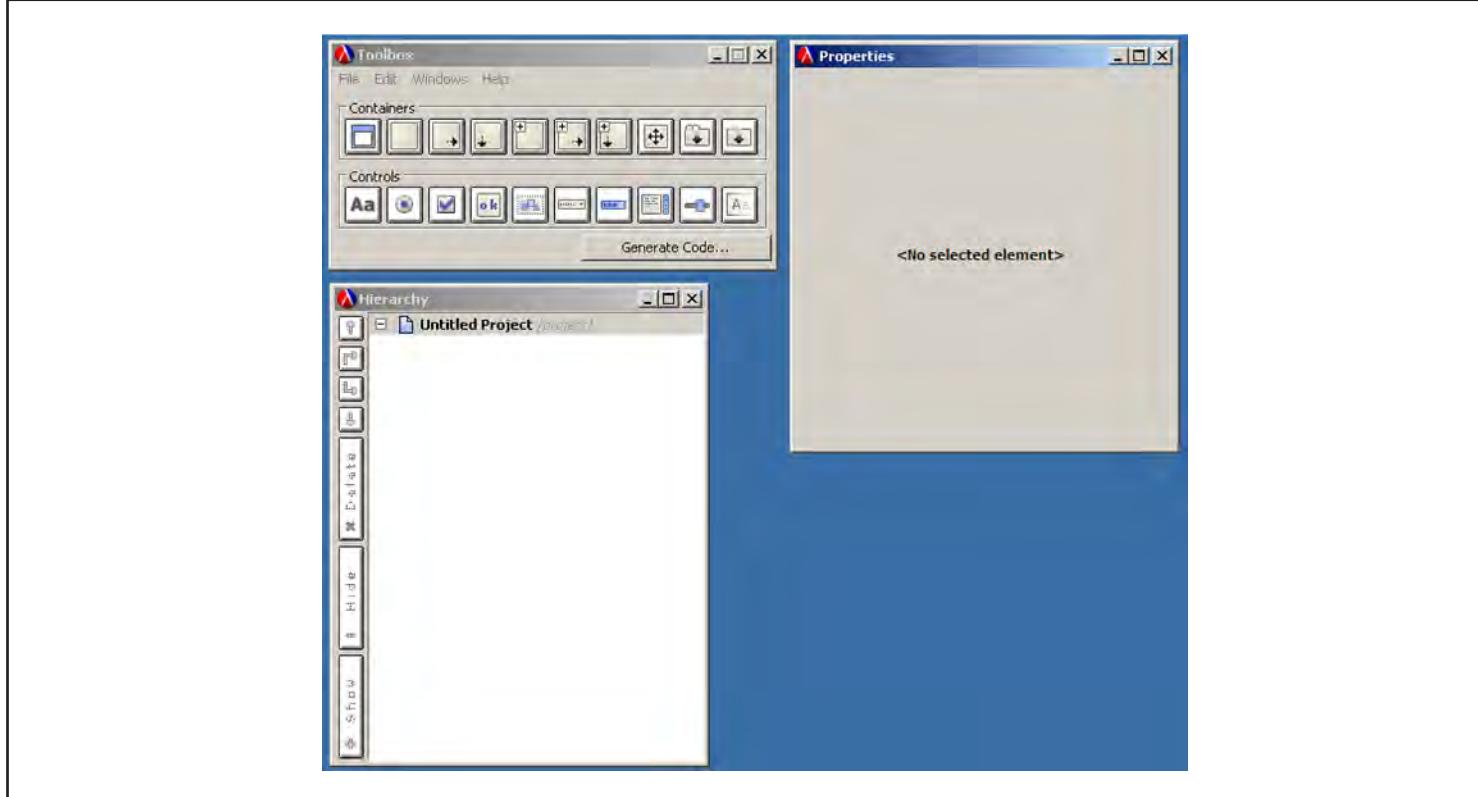
### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Familiarize-se com o ambiente disponibilizado pelo MrEd Designer.

Sem grande preocupação, dispense uns minutos neste ambiente, começando por seleccionar, na janela *Toolbox*, na barra *Containers*, o contentor mais à esquerda. Surge imediatamente uma nova janela... Na janela *Hierarchy* também aparece o contentor seleccionado.

Na janela *Properties*, pode ver e até alterar as propriedades do elemento seleccionado em cada momento.

Volte à janela *Toolbox* e, na barra *Controls*, vá seleccionando elementos, altere as respectivas características e posições...

## INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

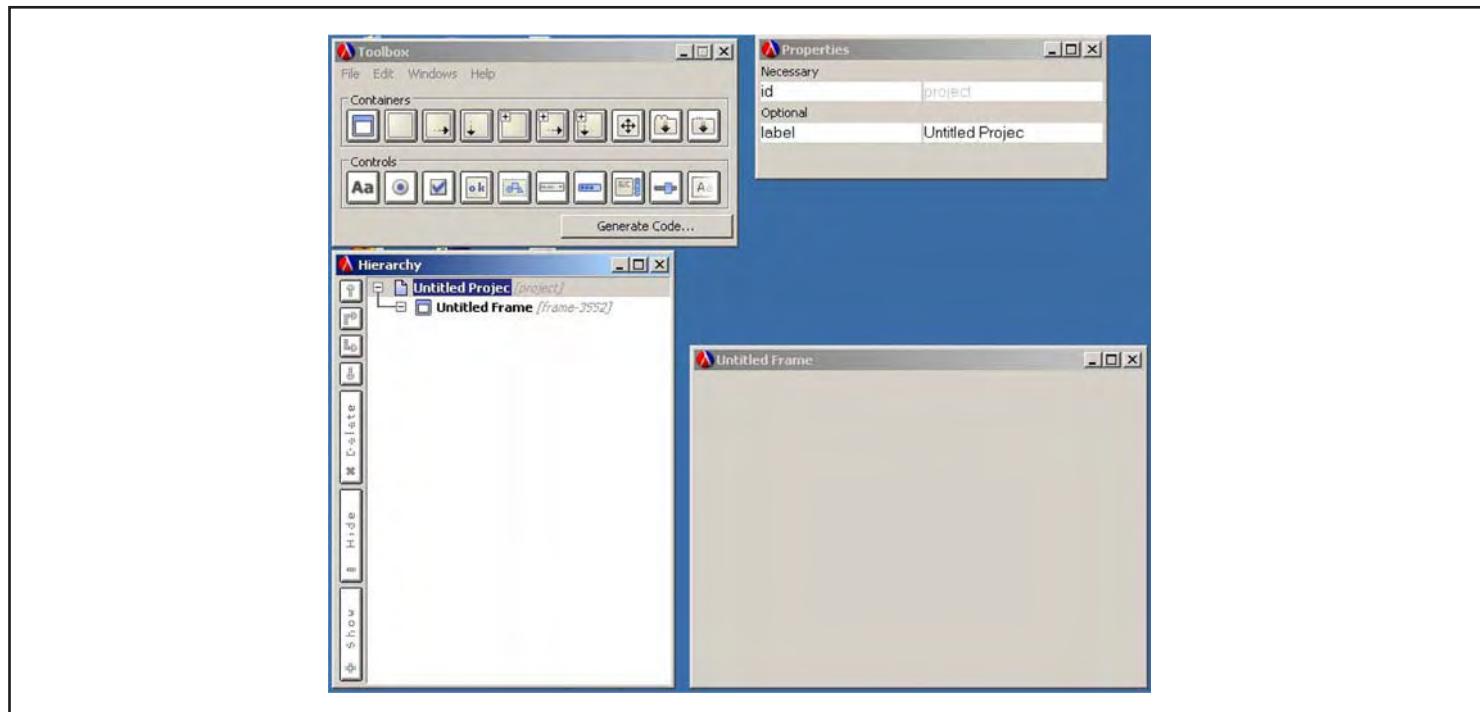
### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exemplo 3

Pretende-se construir, com a ajuda do MrEd Designer, uma interface gráfica que inclua 4 botões (Norte, Sul, Este e Oeste) e um campo de texto, como se mostra na figura. Conforme o botão actuado, no campo de texto surgirá "Vai para Norte", "Vai para Sul", etc.

Neste exemplo, aproveita-se a oportunidade para indicar como se grava o código, gerado automaticamente, que implementa a interface e como se guarda o respectivo projecto. É importante guardar este projecto, pois viabilizará futuras alterações à interface.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

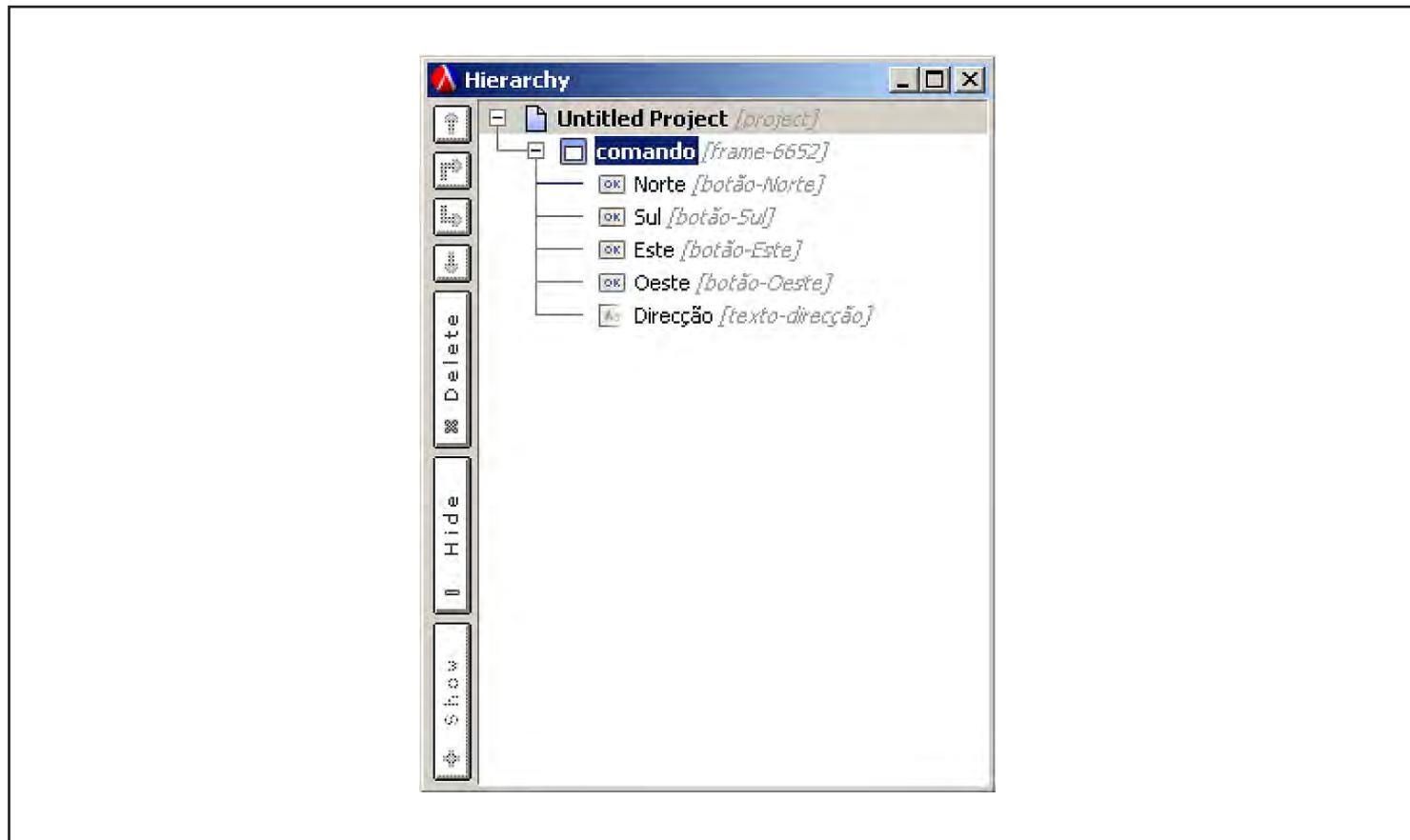
R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Esta interface foi construída criando um contentor, ao qual foi dado o nome comando. Sobre esse contentor foram colocados 4 botões e um campo de texto. Os nomes também foram adequados à respectiva função. Tudo isto se reflecte na janela *Hierarchy*. Na barra vertical desta janela, do lado esquerdo, é possível verificar a existência de funcionalidades para deslocar, apagar, esconder e mostrar elementos existentes na interface.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

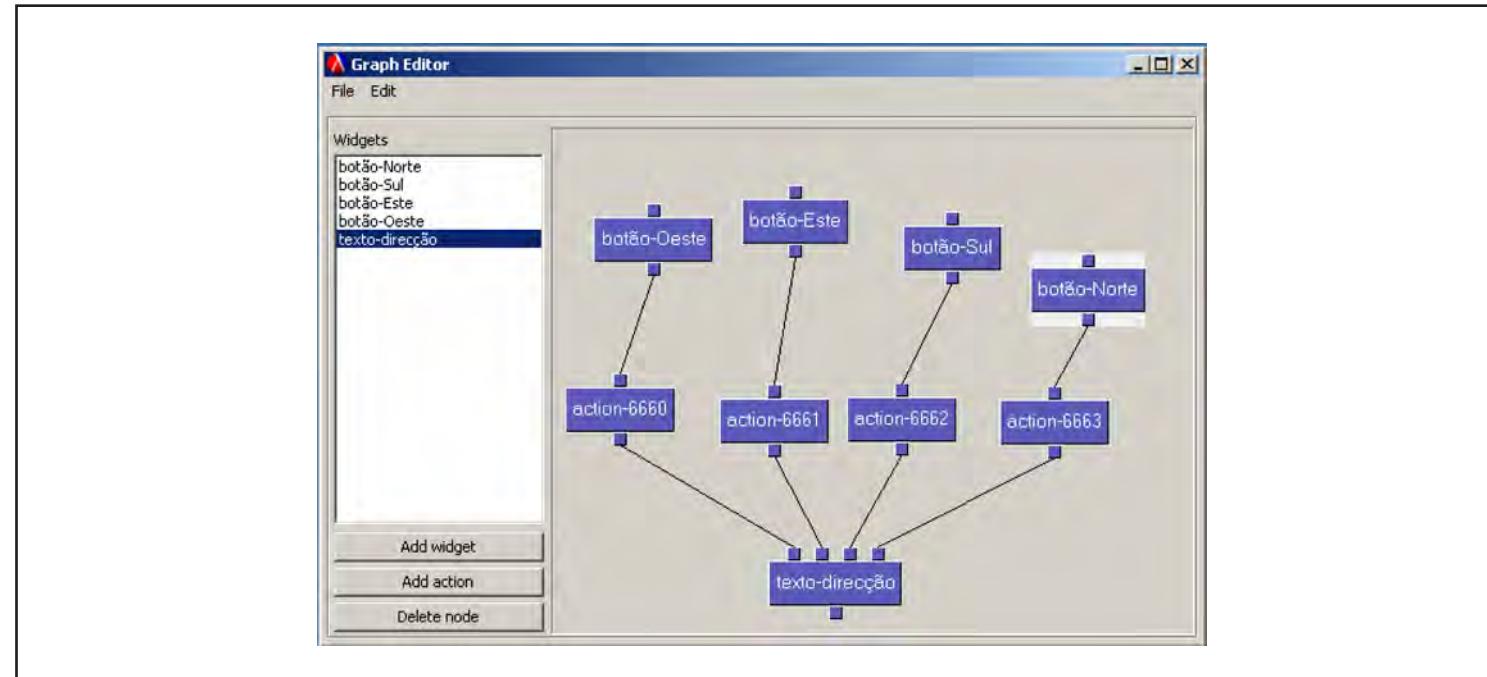
R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

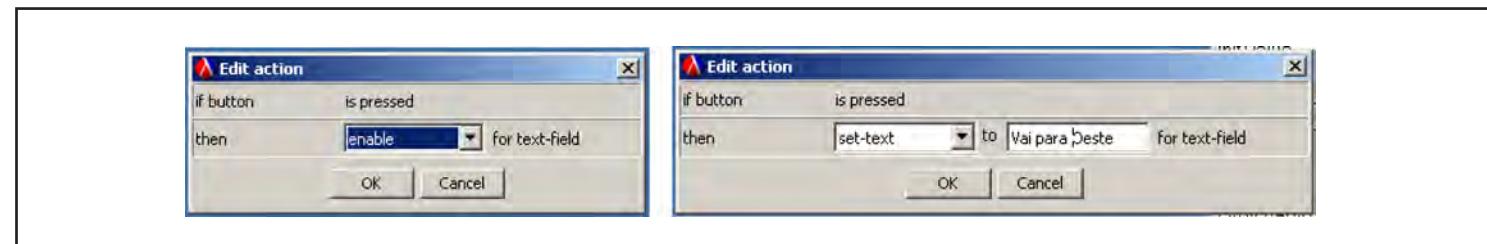


No menu Windows da caixa Toolbox, a escolha da opção *Callback Graph frame*, faz surgir a janela do editor gráfico, que permite estabelecer a ligação entre os elementos da interface, recorrendo a *Add widget* e *Add action*.



Para adicionar mais entradas à caixa de texto, para poder ligar mais botões, basta seleccionar a opção *Add Input tab* do menu *Edit*, tendo a caixa de texto seleccionada.

Actuando com o cursor numa acção é possível programar a funcionalidade pretendida. A figura mostra a programação da acção que liga o botão Oeste ao campo de texto.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

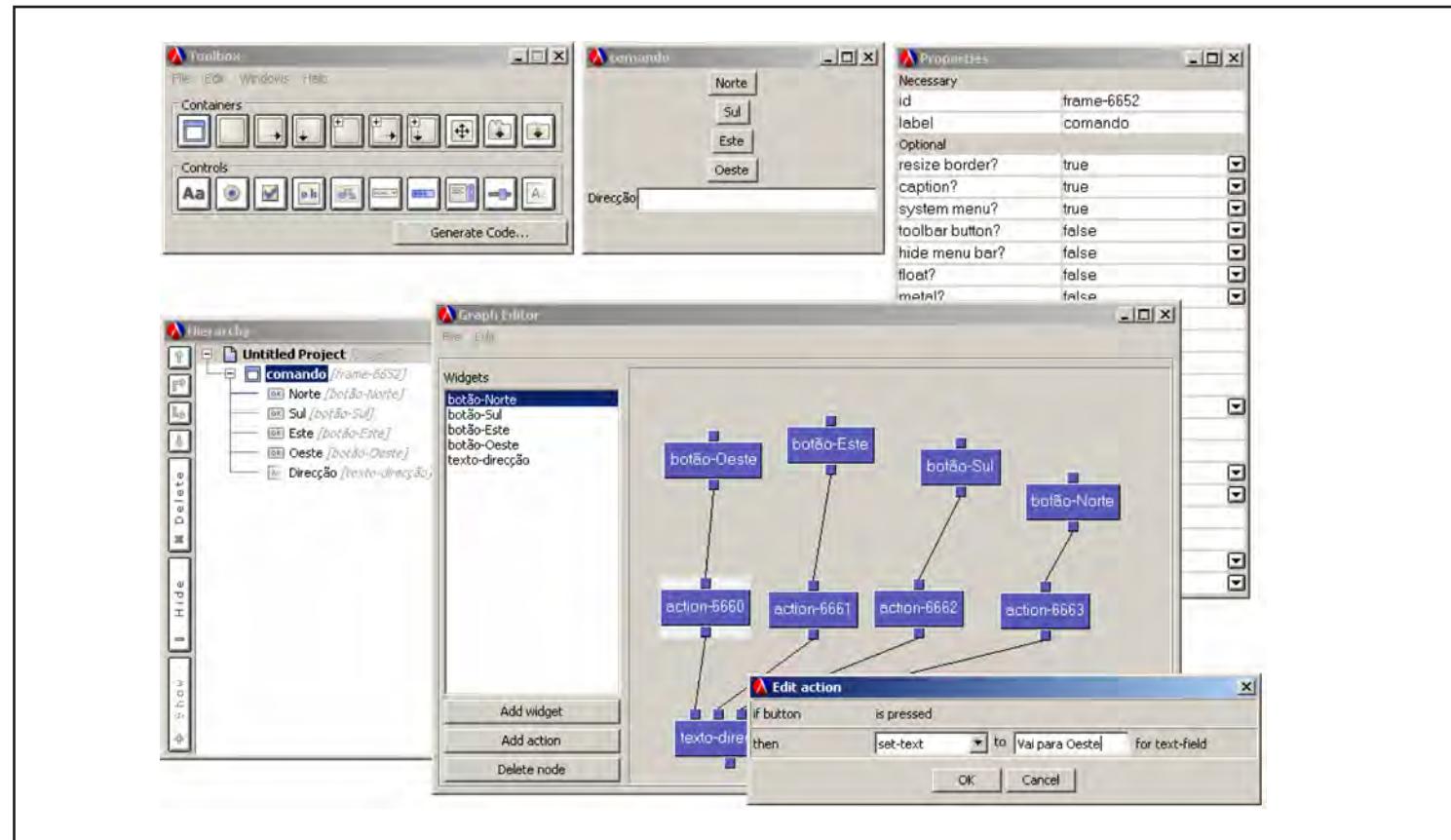
### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Pode agora ter uma perspectiva que agrupa as várias janelas do projecto corrente.



Na janela *Toolbox*, no menu *File*, é possível salvar o projecto da interface, por exemplo, no ficheiro `comando.med`. Mais tarde, este ficheiro será o melhor caminho para introduzir possíveis alterações à interface.

Também nesta janela *Toolbox*, o botão *Generate Code* gera o código da interface que é guardado, por exemplo, no ficheiro `comando.scm`.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



No código que se segue, gerado automaticamente pelo MrEd Designer, verifique a existência de duas partes distintas:

- 1- A parte inicial, a mais interessante, onde encontrará os elementos da interface e as respectivas funções de resposta (*call-backs*)
- 2- A parte restante essencialmente constituída pela descrição daqueles elementos.



Outros aspectos a considerar no código que se segue.

Tome em atenção, por exemplo, a porção de código relativa a **botão-Norte-callback**, e verifique nele também duas partes:

- 1- (`send texto-direcção set-value "Vai para Norte"`)  
envia para **texto-direcção** o valor "Vai para Norte"
- 2- A parte restante do código, identificada por uma linha poligonal, pode pô-la como comentário, já que isso não terá qualquer consequência no funcionamento da interface! E, para já, não se preocupe com este pormenor, aliás, um pouco estranho.



O código da interface é criado como um módulo. Mas um módulo tem o nome do ficheiro, sem a extensão. Neste caso, o ficheiro é comando.scm e o módulo tem o nome comando.

Portanto, tenha cuidado e não altere o nome do ficheiro...

Mas se precisar de alterar o nome do ficheiro, altere também o nome do módulo, para que se mantenham iguais.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
; =====
; === Code generated with MrEd Designer 2.0 ===
; === http://www.hexahedron.hu/personal/peteri/mreddesigner/index.html ===
; =====

; File: ...gui\comando.scm
(module comando mzscheme
  (require (lib "class.ss") (lib "mred.ss" "mred"))
  (define
    (main)
    (letrec ((botão-Norte-callback
              (lambda (w e)
                (send texto-direcção set-value "Vai para Norte")
                (textodirecção-callback
                  texto-direcção
                  (new control-event% (event-type 'text-field-enter)))))

              (botão-Sul-callback
                (lambda (w e)
                  (send texto-direcção set-value "Vai para Sul")
                  (textodirecção-callback
                    texto-direcção
                    (new control-event% (event-type 'text-field-enter)))))

              (botão-Este-callback
                (lambda (w e)
                  (send texto-direcção set-value "Vai para Este")
                  (textodirecção-callback
                    texto-direcção
                    (new control-event% (event-type 'text-field-enter)))))

              (botão-Oeste-callback
                (lambda (w e)
                  (send texto-direcção set-value "Vai para Oeste")
                  (textodirecção-callback
                    texto-direcção
                    (new control-event% (event-type 'text-field-enter)))))

              ; ----- linha colocada manualmente -----
              (frame-6652
                (new
                  frame%
                  (parent #f)
                  (alignment '(center top)))))))
```

# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
(stretchable-height #t)
(label "comando")
(height 183)
(x 421)
(stretchable-width #t)
(min-height 0)
(style '())
(border 0)
(width 261)
(spacing 0)
(y 151)
(min-width 0)
(enabled #t)))
(botão-Norte
(new
button%
(parent frame-6652)
(min-width 0)
(stretchable-height #f)
(min-height 0)
(style '())
(label "Norte")
(vert-margin 2)
(horiz-margin 2)
(stretchable-width #f)
(enabled #t)
(callback botão-Norte-callback)))
(botão-Sul
(new
button%
(parent frame-6652)
(min-width 0)
(stretchable-height #f)
(min-height 0)
(style '())
(label "Sul")
(vert-margin 2)
(horiz-margin 2)
(stretchable-width #f)
(enabled #t)
(callback botão-Sul-callback)))
(botão-Este
(new
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
button%
  (parent frame-6652)
  (min-width 0)
  (stretchable-height #f)
  (min-height 0)
  (style '())
  (label "Este")
  (vert-margin 2)
  (horiz-margin 2)
  (stretchable-width #f)
  (enabled #t)
  (callback botão-Este-callback)))
(botão-Oeste
  ...
  (texto-direcção
    (new
      text-field%
        (parent frame-6652)
        (min-width 0)
        (stretchable-height #f)
        (label "Direcção")
        (min-height 0)
        (horiz-margin 2)
        (stretchable-width #t)
        (style '(single))
        (init-value "")
        (enabled #t)
        (vert-margin 2)
        (callback texto-direcção-callback)))
    (send frame-6652 show #t)))
  (main))
```



Teste a interface gráfica que permite a simulação das direcções geográficas.  
Experimente pôr como comentário as linhas de código que começam assim:  
(texto-direcção-callback...  
e verifique se a interface continua a funcionar correctamente.



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

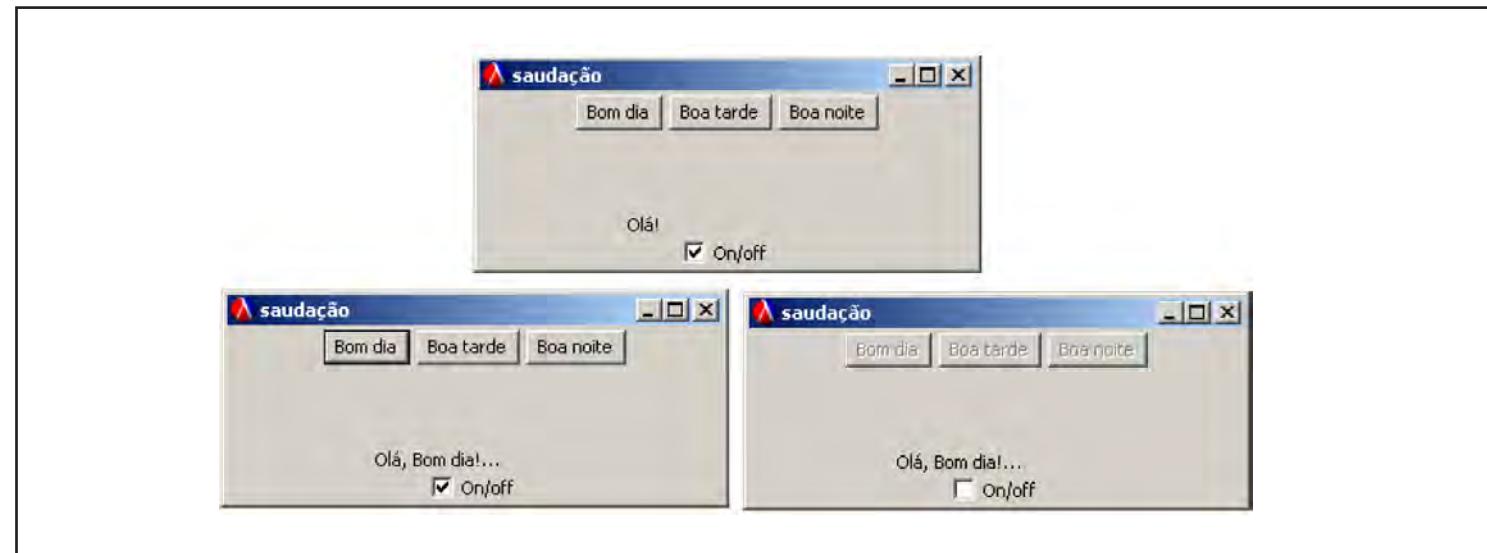
ANEXOS



### Exercício 3

Desenvolva uma interface gráfica, com a ajuda do MrEd Designer, que permita simular uma saudação de Bom dia, Boa tarde ou Boa noite. Estas saudações devem poder ser inibidas.

Observar nas figuras uma sugestão para a interface pedida.



Desenvolva e teste a interface gráfica que simula uma saudação.

# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

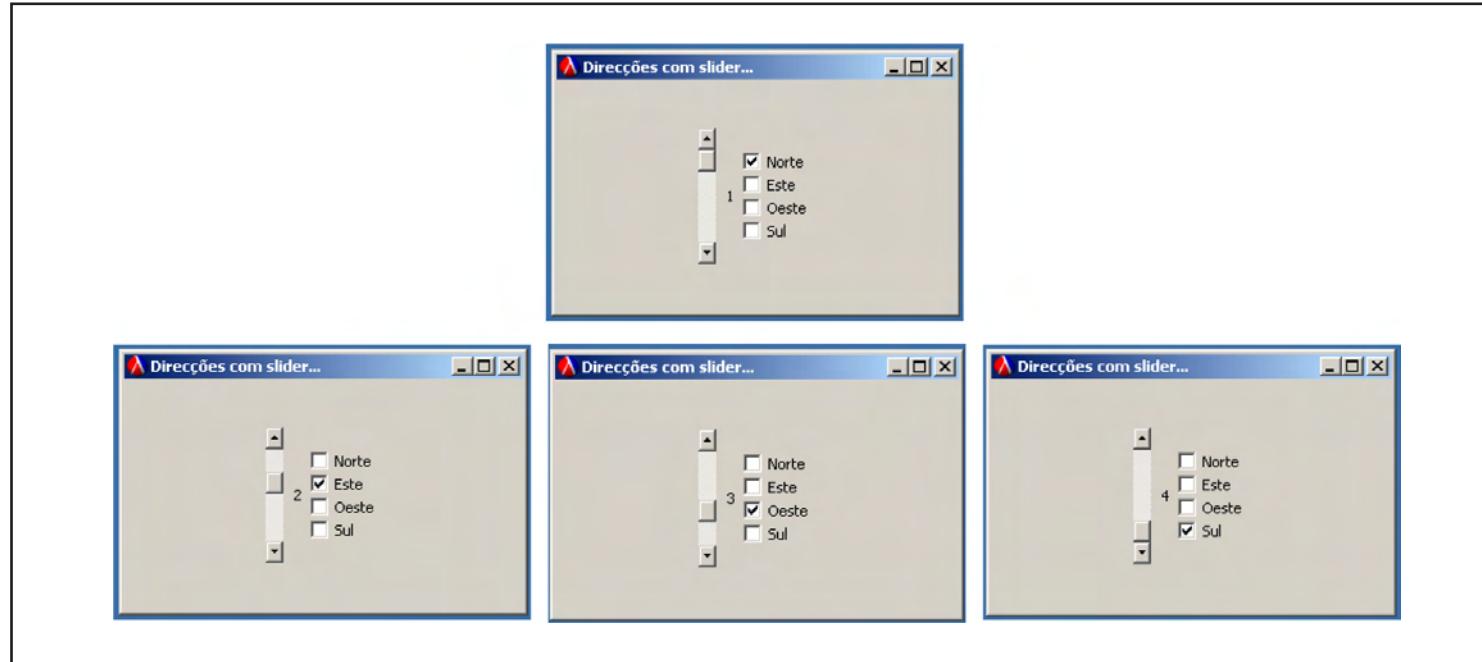
ANEXOS



### Exercício 4

Pretende-se construir com a ajuda do MrEd Designer uma interface gráfica que inclua um *slider* de 4 posições (de 1 a 4) e 4 caixas de teste (*check box*) com as etiquetas Norte, Sul, Este e Oeste. A posição do *slider* seleccionada indicará, automaticamente, a caixa de teste que deverá ser actuada.

A figura superior indica a situação inicial da interface.



Desenvolva e teste a interface gráfica que permite a simulação das direcções geográficas.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

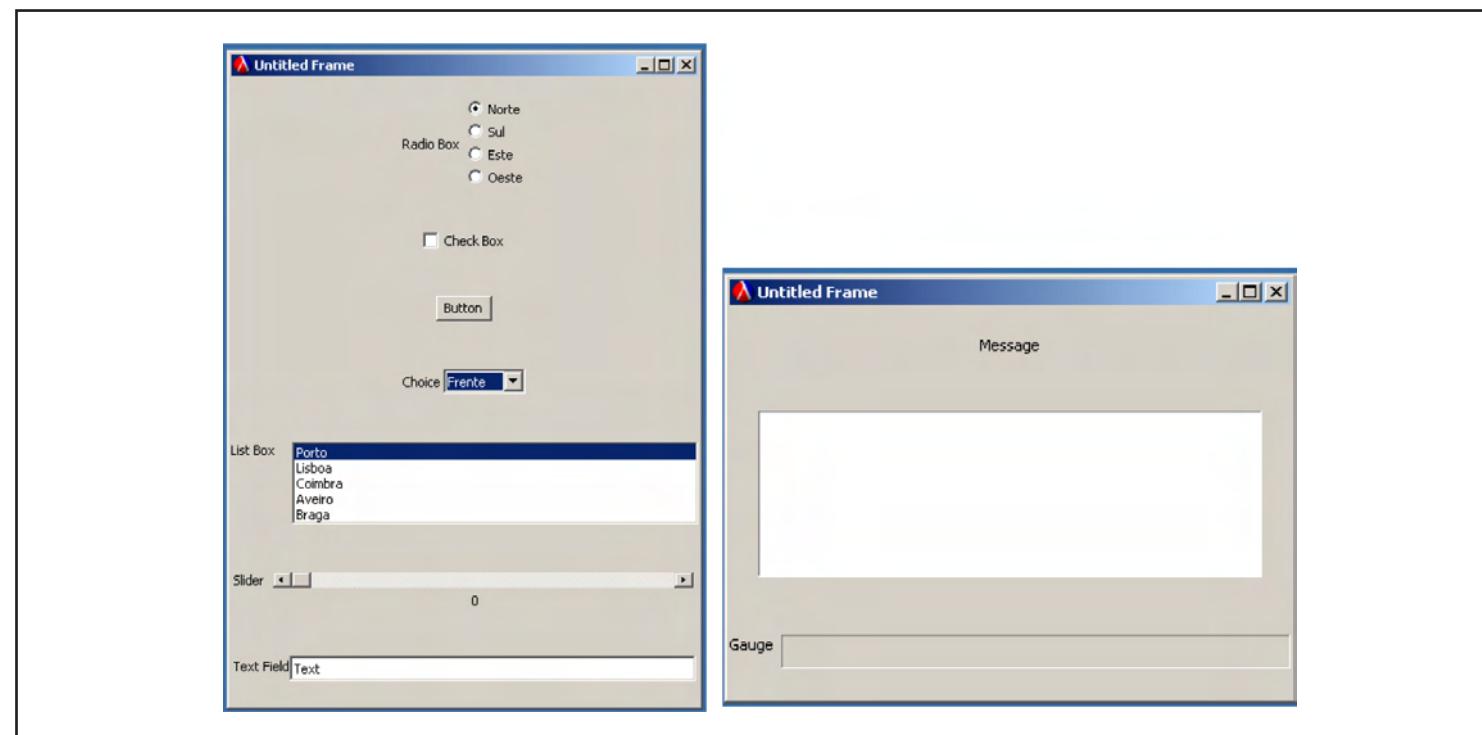


## O MrEd Designer pode exigir alguma intervenção manual!

Os exercícios e exemplos apresentados colocaram-no em contacto com alguns dos elementos de interface do MrEd Designer. Na figura são mostrados, para além dos elementos que já conhece, os outros que, provavelmente, ainda não terá encontrado. Na parte esquerda da figura surgem os elementos que podem funcionar quer como entrada quer como saída. Na parte direita, pode ver os que apenas funcionam como saída. Destes últimos, o que mais utilizará será, certamente, o campo de mensagem (*Message*).

O MrEd Designer facilita, de uma forma amigável, durante a sua utilização, todos os dados de que necessita sobre os elementos que terá escolhido para um projecto. No essencial, as acções que pode programar sobre cada elemento são:

- *enable, disable*
- *show, hide*
- *check, uncheck*
- *toggle*
- *set-label, set-text, set-value, set-selection, set-list*



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



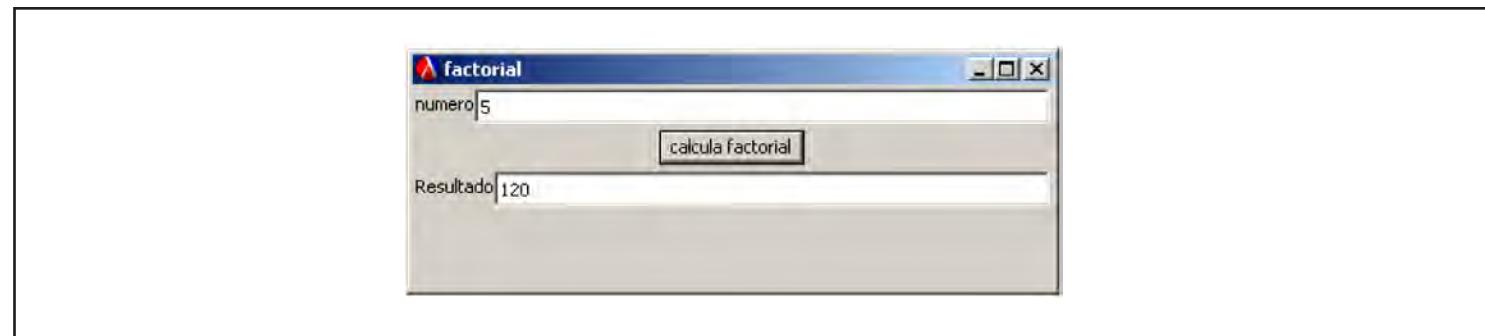
De uma forma geral, os elementos de interface disponibilizados na janela *Toolbox* do MrEd Designer não aceitam todas as acções apresentadas.

Como sugestão, pode abrir o MrEd Designer e construir uma tabela que indique, para cada elemento, as acções que ele aceita quando funciona como saída.

Pode enriquecer essa tabela juntando-lhe também o que pode obter desses elementos quando funcionam como entrada.

### Exemplo 4 - Como ultrapassar, manualmente, algumas limitações do MrEd Designer

Imagine que pretendia criar uma interface, como já fez com a biblioteca `gui.ss`, para determinar o factorial de um número, como mostra a figura.



Certamente não terá encontrado, até ao momento, uma forma de convencer o MrEd Designer a colocar no campo de texto o valor obtido pela chamada de um procedimento, neste caso, um procedimento que calculasse o factorial.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

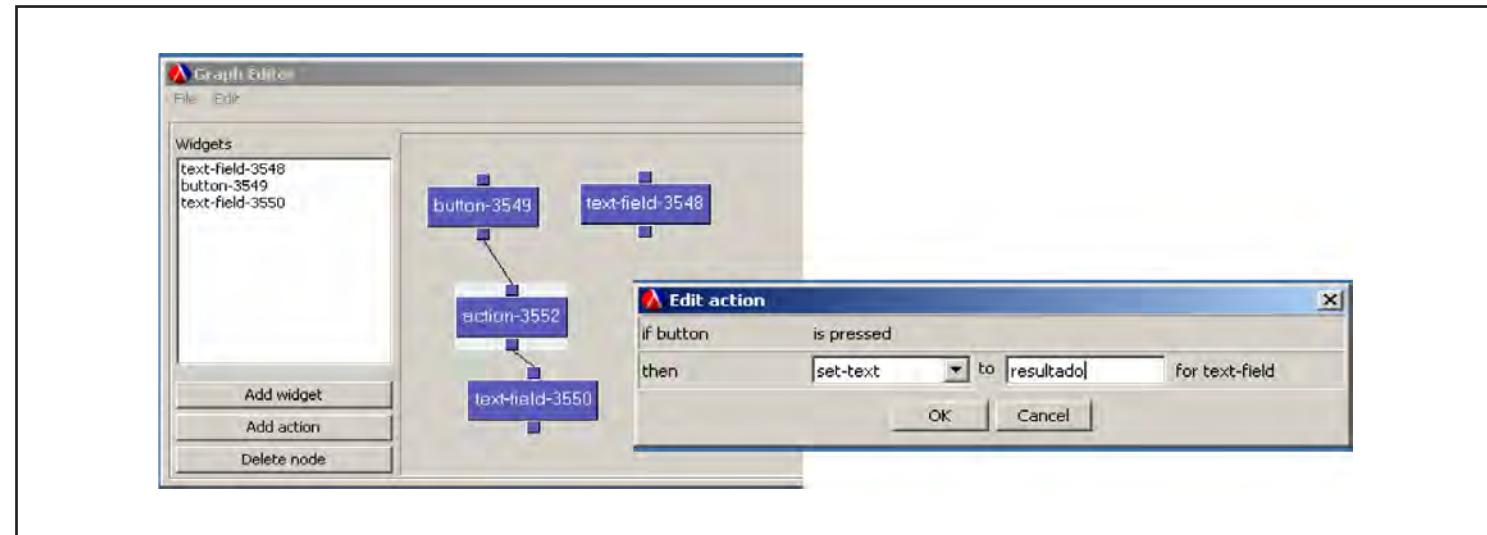
## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

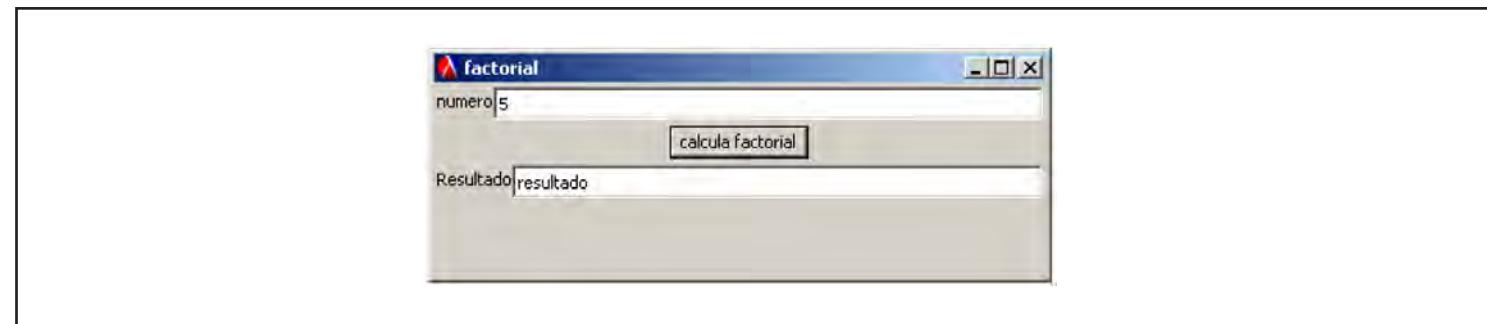
## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Nas acções disponibilizadas pelo MrEd Designer, o mais próximo a que poderá ficar do desejado será colocar no campo de texto uma cadeia de caracteres, como se pode ver na figura.



Com o código gerado automaticamente pelo MrEd Designer, obteria um resultado diferente do pretendido.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

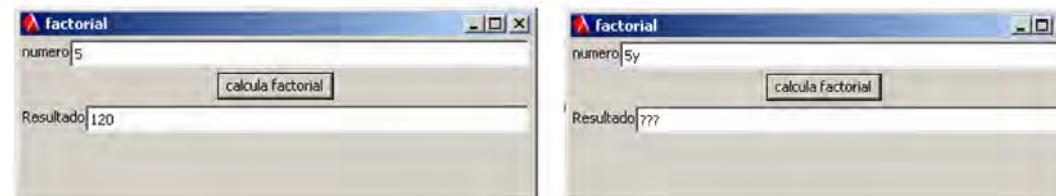


```
; =====
;== Code generated with MrEd Designer 2.0 ==
;== http://www.hexahedron.hu/personal/peteri/mreddesigner/index.html ==
;=====
; File: ...\\gui\\factorial_inter.scm
(module factorial_inter mzscheme
  (require (lib "class.ss") (lib "mred.ss" "mred"))
  (define
    (main)
      (letrec ((text-field-3548-callback (lambda (w e) (void)))
              (button-3549-callback
                (lambda (w e)
                  (send text-field-3550 set-value "resultado")
                  (text-field-3550-callback
                    text-field-3550
                    (new control-event% (event-type 'text-field-enter)))))))
        (text-field-3550-callback (lambda (w e) (void)))
        (frame-3547
          (new
            frame%
            (parent #f)
            (height 150)
            (alignment '(center top))
            (style '())
            (stretchable-width #t)
            ...
            
```



Teste a interface gráfica, ainda numa versão intermédia, que permite simular o cálculo do factorial de um número.

Para contornar dificuldades desta natureza, como se mostra na figura, bastará alguma intervenção manual sobre o código gerado automaticamente.



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Concretizando, para este caso particular:

- 1- Colocar no directório de trabalho (onde vai salvar o código da interface) um módulo com o procedimento factorial

```
; módulo a colocar num directório de trabalho
; com a designação proc-factorial.scm
;
(module proc-factorial
  mzscheme
  (define factorial
    (lambda (n)
      (if (zero? n)
          1
          (* n (factorial (sub1 n))))))

  (provide factorial))
```

- 2- Alterar o código gerado automaticamente, com duas intervenções manuais:

- 1<sup>a</sup> intervenção manual

Acrecentar

```
(require (file "proc-factorial.scm"))
```

- 2<sup>a</sup> intervenção manual

Alterar a função de resposta (*call-back*) do botão calcula factorial

No essencial, será substituir a cadeia "resultado" da função button-3549-callback por um código Scheme que:

- leia a cadeia de caracteres do campo de texto numero
- converta a cadeia lida em valor numérico
- verifique se, de facto, é um valor numérico
  - se for valor numérico, chama factorial com este valor como argumento, converte o resultado para uma cadeia de caracteres e envia a cadeia para o campo de texto Resultado.
  - se não for valor numérico, envia "???" para o campo de texto Resultado.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Provavelmente, torna-se tudo mais fácil, analisando o extracto de código que se segue.

```
;=====
;=====          Code generated with MrEd Designer 2.0      ====
;=====  http://www.hexahedron.hu/personal/peteri/mreddesigner/index.html ===
;=====

;; -----
;; código com duas pequenas intervenções manuais -
;; -----


;; File: ...gui\factorial.scm
(module factorial mzscheme
  (require (lib "class.ss") (lib "mred.ss" "mred"))
  ; ----- 1- início de intervenção manual -----
  (require (file "proc-factorial.scm"))
  ; ----- 1- fim de intervenção manual -----
  (define
    (main)
    (letrec ((text-field-3548-callback (lambda (w e) (void)))
             ; ----- 2- inicio de intervenção manual -----
             (button-3549-callback
              (lambda (w e)
                (let ((valor (string->number
                               (send text-field-3548 get-value))))
                  (send text-field-3550 set-value "resultado")
                  (send text-field-3550
                        set-value
                        (if (number? valor)
                            (number->string (factorial valor))
                            ; ----- 2- fim de intervenção manual -----
                            (text-field-3550-callback
                             text-field-3550
                             (new control-event% (event-type 'text-field-enter)))))))
                ; igual ao código da listagem anterior
                ; ...
              )
            )
          )
        )
      )
    )
  )
)
```



Teste a interface gráfica que permite calcular o factorial de um número.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Exercício 5

Comece por analisar a interface que se apresenta.



Agora, tente identificar os elementos de interface utilizados.

De seguida, no espaço de trabalho que poderá abrir, construa com o MrEd Designer uma interface semelhante a esta, numa versão intermédia, ou seja, numa versão produzida automaticamente e que necessitará de alguma intervenção manual para funcionar correctamente.



Desenvolva e teste a interface gráfica apresentada, ainda numa versão intermédia, que permite simular o cálculo da área e perímetro de rectângulos.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7 EXERCÍCIOS E PROJECTOS

### ANEXOS



De seguida, é apresentada a implementação da interface, já na sua versão final.

```
;=====
;=====          Code generated with MrEd Designer 2.0      ====
;=====  http://www.hexahedron.hu/personal/peteri/mreddesigner/index.html  ====
;=====

;; código adaptado manualmente
;; para responder ao enunciado do problema
;;
(module rectangulo-b mzscheme
  (require (lib "class.ss") (lib "mred.ss" "mred"))
  (define
    (main)
    (letrec ((text-field-27842-callback (lambda (w e) (void)))
            (text-field-27843-callback (lambda (w e) (void)))
            (choice-31317-callback (lambda (w e) (void)))
            (button-27847-callback
              (lambda (w e)
                ; ----- início da área de intervenção manual -----
                (let ((comp (string->number (send text-field-27842 get-value)))
                      (larg (string->number (send text-field-27843 get-value))))
                  (let ((str (send choice-31317 get-string-selection)))
                    (cond ((and (string? str)
                                (string-ci=? str "área"))
                            (send message-27844
                                  set-label ; <--set-label
                                  (string-append "tem como área "
                                                (number->string (* comp larg))
                                                " cm2")))
                           ((and (string? str)
                                 (string-ci=? str "perímetro"))
                            (send message-27844
                                  set-label ; <--set-label
                                  (string-append "tem como perímetro "
                                                (number->string (+ comp comp larg larg))
                                                " cm"))))
                           (else
                             (send frame-27841 show #f))))))
                ; ----- final da área de intervenção -----
                (frame-27841
                  ...
                  ))))))
```

INTRODUÇÃO
1 - O ESSENCIAL DO SCHEME
2 - RECURSIVIDADE
3 - ABSTRACÇÃO DE DADOS
4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE
5 - ABSTRACÇÕES COM DADOS MUTÁVEIS
6 - SCHEME E OUTRAS TECNOLOGIAS
R 1 2 3 4 5
7 EXERCÍCIOS E PROJECTOS
ANEXOS



Teste a interface gráfica, na versão final, que permite calcular a área e o perímetro de rectângulos.

## Exercício 6

Construa com a ajuda do MrEd Designer uma interface semelhante à do exercício anterior, mas que agora, para além de rectângulos, considere também triângulos e círculos.



Desenvolva e teste a interface gráfica que permite o cálculo de áreas e de perímetros de figuras geométricas.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

## Módulo 6.2 - Criar jogos... em Scheme

Os jogos de computador são um dos produtos informáticos com maior sucesso.

Ao longo deste módulo será feita uma introdução ao desenvolvimento de jogos de computador, utilizando recursos disponibilizados pela linguagem Scheme. Com base num dos jogos dos primórdios da indústria informática - o Pong - será desenvolvido um caso de estudo que permitirá a descoberta das diversas fases da concepção de um jogo de computador. Tome pois atenção, porque no final deste módulo, como desafio, ser-lhe-á pedido que desenvolva um jogo de computador...

Quase todos os jogos de computador são aplicações multimédia. O Scheme permite o desenvolvimento de aplicações multimédia através do módulo Allegro.plt, parte integrante da distribuição PLT Scheme. Paralelamente à abordagem ao desenvolvimento dos jogos electrónicos, será feita uma descrição das principais funcionalidades disponibilizadas por este módulo, que se encontra documentado em [\*\*Allegro: Multimedia library and game utilities\*\*](#).

### Palavras-Chave

Jogos de computador, jogo Pong, multimédia.

# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

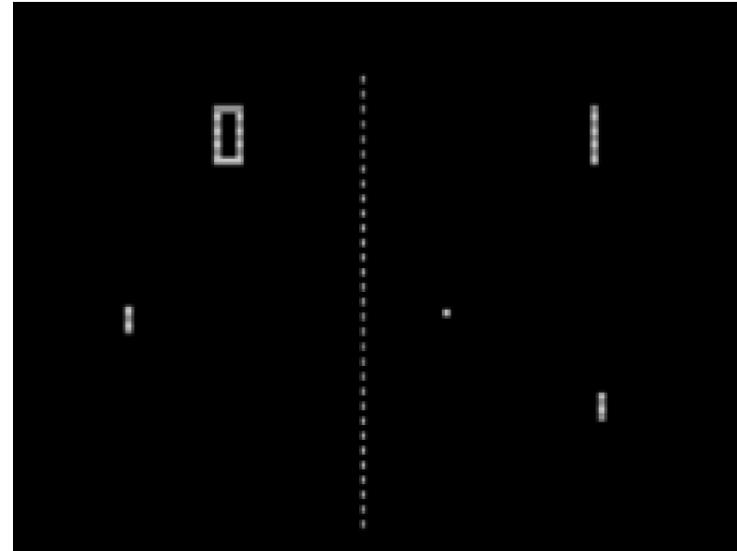


## Pong, o meu primeiro jogo... em Scheme

A indústria dos jogos de computador é uma das mais lucrativas e com maior crescimento da actualidade. Tendo sido considerados como parentes pobres das outras aplicações informáticas "mais sérias", os jogos de computador possuem, na actualidade, orçamentos equiparados a filmes de Hollywood.



**Em Abril de 2008, o jogo Grand Theft Auto tornou-se recordista de receitas na indústria do entretenimento, ao obter no primeiro dia de vendas 197 milhões de euros, suplantando os anteriores recordes de filmes (Homem-Aranha 3) e livros (Harry Potter e os Talismãs da Morte).**



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Um dos jogos mais marcantes da história dos jogos de computador foi o Pong, criado pela Atari Inc. em 1972. Não porque tivesse sido o primeiro jogo de computador, mas porque foi o primeiro a ter êxito comercial, quer em máquinas de café, quer em consolas domésticas, lançando as bases para a actual indústria de jogos de computador. Os jogos de computador são, na sua generalidade, aplicações multimédia desenvolvidas para o entretenimento. No entanto, esta tecnologia ganha cada vez mais terreno em outras áreas como a educação ou a formação profissional (*edutainment*) e as aplicações empresariais (*serious gaming*).

O DrScheme possibilita a criação de aplicações multimédia através do módulo **Allegro.plt**, parte integrante da distribuição PLT Scheme. Este módulo teve origem na adaptação para Scheme, por Jon Rafkind, da biblioteca **Allegro** que foi desenvolvida originalmente em C.

Para poder utilizar estas funcionalidades, deve incluir a biblioteca *util.ss* do módulo *Allegro.plt* através de:

```
(require (planet "util.ss" ("kazzmir" "allegro.plt" 1 0)))
; 1 0 significa importação de qualquer versão back-compatible com a versão 1.
```



Este módulo encontra-se documentado na página Web [Allegro: Multimedia library and game utilities](#).

Poderá também encontrar uma versão deste sítio Web neste CD.

Dos 9 capítulos deste documento, será capaz de identificar quais serão os mais relevantes para o desenvolvimento de jogos?

### O motor do jogo

O primeiro passo na criação de um jogo é o desenvolvimento do motor de jogo. Este é, muito simplesmente, o ciclo que actualiza todos os objectos e, sendo a maior parte dos jogos aplicações gráficas interactivas, lê os dispositivos de entrada e visualiza gráficos nos dispositivos de saída.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Assim, o motor do jogo realiza as seguintes tarefas de forma cíclica:



1. Entrada
2. Processamento
3. Saída

A frequência com que este ciclo é executado condiciona a velocidade do jogo podendo, em alguns casos, afectar também o seu nível de interactividade. Em Allegro, o procedimento que define e implementa este ciclo é:

```
(game-loop logic-proc draw-proc game-delay)
```

em que:

- logic-proc é o procedimento que irá tratar da entrada e do processamento do jogo (tarefas 1 e 2, anteriormente indicadas)
- draw-proc é o procedimento que irá desenhar na janela gráfica, que tem como parâmetro a imagem onde será feita a visualização - normalmente associa-se ao ecrã - (corresponde à tarefa 3)
- game-delay é o intervalo de tempo entre dois ciclos consecutivos do motor de jogo.

Mas antes de se definir o motor de jogo, é necessário inicializar o ambiente Allegro, bem como a sua janela gráfica, através do procedimento:

```
(easy-init width height depth mode)
```

onde

- width e height são as dimensões da janela (em *pixels*)
- depth o número de bits por *pixel* (define o número máximo de cores)
- mode define se a área gráfica será apresentada como uma janela ou *full-screen*.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

No final do programa deve-se chamar o procedimento easy-exit para fechar a janela gráfica e libertar os recursos associados ao ambiente Allegro.



Com o que sabe do Allegro, tente esboçar a estrutura de um jogo.

A estrutura de um jogo baseado no Allegro andará à volta de:

```
easy-init
game-loop
    logic-proc      tarefa 1. Entrada
                    tarefa 2. Processamento
    draw-proc       tarefa 3. Saída
    game-delay
easy-exit
```

Pormenorizando um pouco mais: o procedimento principal, jogo-pong, tem como parâmetros as dimensões da janela e o nível de dificuldade do jogo. O programa termina premindo a tecla Esc.

```
(require (planet "util.ss" ("kazzmir" "allegro.plt" 1 0)))
; na linha seguinte: 1 0 significa importação de qualquer versão back-compatible com a versão 1.0
(require (prefix image- (planet "image.ss" ("kazzmir" "allegro.plt" 1 0))))
(require (planet "keyboard.ss" ("kazzmir" "allegro.plt" 1 0)))
;;;;;;;;;;;;;;;;;;
; função principal do jogo Pong ;
;;;;;;;;;;;;;;;;;;
(define jogo-pong
  (lambda (x-janela y-janela nivel)
    ;;;;;;;;;;;;;;;;;;
    ; 1. Definições ;
    ;;;;;;;;;;;;;;;;;;
    (letrec (
      ;;;;;;;;;;;;;;;;;;
      ; 1.1. Variáveis do jogo ;
      ;;;;;;;;;;;;;;;;;;
      (pi (acos -1)) ; valor de Pi
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
;:::::::::::::::::::  
; 1.2. Procedimentos auxiliares ;  
;:::::::::::::::::::  
; retorna inteiro exacto  
(inteiro  
  (lambda (x)  
    (inexact->exact (round x))))
```

```
;inicializacao do Allegro  
; (easy-init width height depth mode)  
(easy-init x-janela y-janela 16 'WINDOWED) ; ou 'FULLSCREEN
```

```
; O motor de jogo  
; (game-loop logic-proc draw-proc game-delay)
```

```
(game-loop  
;::::::::::::::::::;  
; 2. Logica do jogo ;  
;::::::::::::::::::;  
(lambda ()  
  (if (keypressed? 'ESC)  
    #t ; devolvendo #t, termina o ciclo...  
    (begin  
      ;::::::::::::::::::;  
      ; 2.1. Entrada ;  
      ;::::::::::::::::::;  
      ; 2.2. Processamento do jogo ;  
      ;::::::::::::::::::;  
      ; devolvendo #f, repete novamente o ciclo  
      #f)))  
;::::::::::::::::::;
```

```
; 3. Saida grafica ;  
;::::::::::::::::::;  
(lambda (imagem)  
  (image-print-center imagem  
    (inteiro (/ x-janela 2)) (inteiro (* y-janela 0.4))  
    (image-color 255 255 255) -1 "PONG"))
```

```
; determina temporizacao do motor de jogo com base na velocidade desejada  
(frames-per-second 30)
```

```
; fecha janela Allegro e termina programa  
(easy-exit))  
(display "Obrigado por ter jogado o Pong."))
```

# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



Experimente o procedimento `jogo-pong`. O ecrã abre com

`> (jogo-pong 800 600 1)`

e o programa termina premindo a tecla `Esc`.

Com este esqueleto básico de um jogo em Scheme pouco poderá fazer, a não ser, em `image-print-center`, alterar a cadeia de caracteres "PONG", as coordenadas do centro desta cadeia ou a sua cor...

Aproveite para consultar, em [\*\*Allegro: Multimedia library and game utilities\*\*](#), a informação relativa a `image-print-center` e `image-print`.



Observe que se utilizou o procedimento `prefix` para associar um prefixo a todos os procedimentos do módulo `image.ss`. Assim todos os procedimentos deste módulo passam a incluir no seu nome `image-`.

Exemplo: O procedimento `color` passa a ser identificado por `image-color`.

Qual será a vantagem de utilizar este prefixo?



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

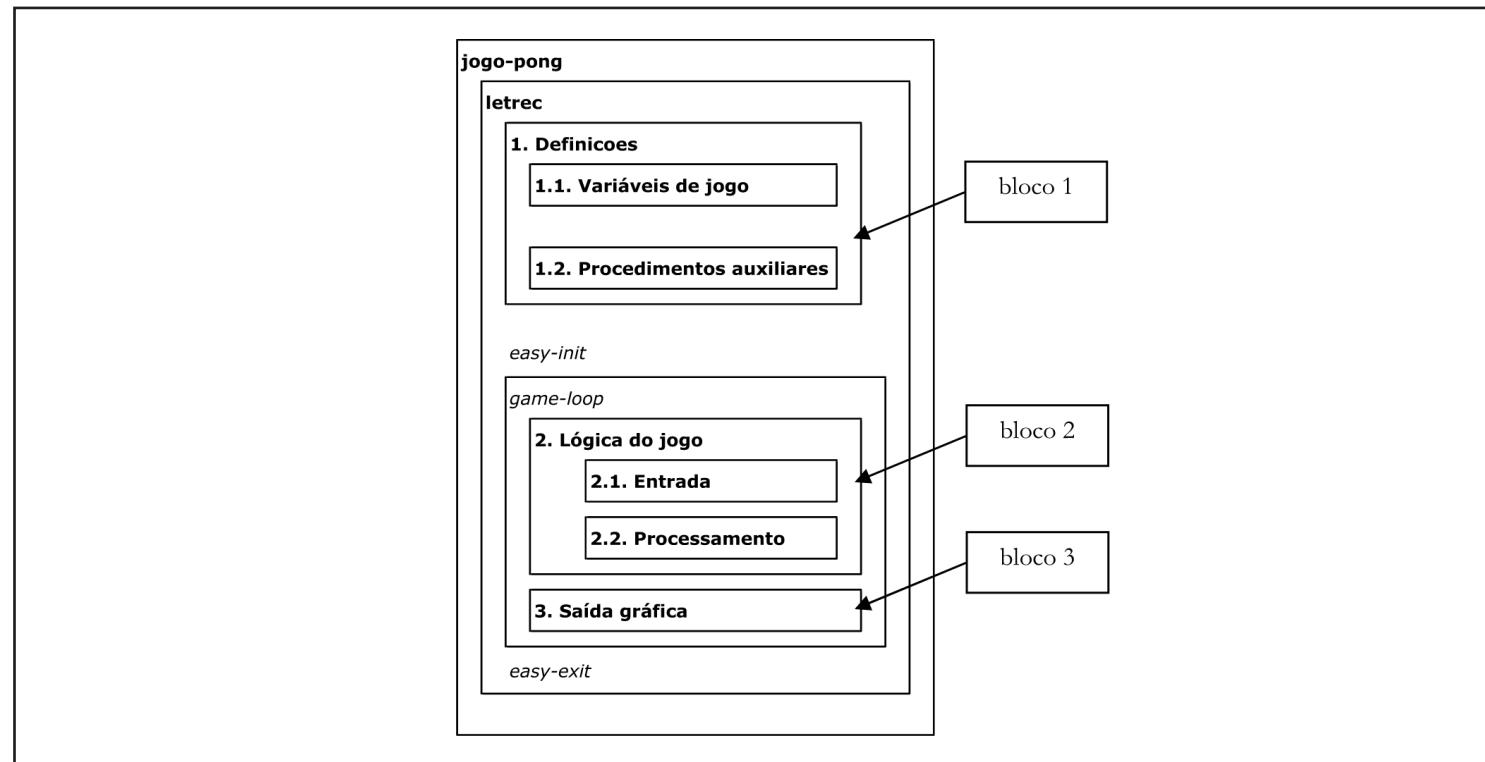
### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

O diagrama de blocos do procedimento `jogo-pong` é apresentado na figura.



O procedimento `jogo-pong` inicia o Allegro com a chamada a `easy-init` e, no final, termina-o com a chamada a `easy-exit`.

O bloco 1, encabeçado com um `letrec`, possibilita a definição das variáveis de jogo (posições dos objectos, pontuação, etc.) e a definição de procedimentos auxiliares. Este bloco deverá ser definido antes da inicialização do Allegro (`easy-init`) para permitir, caso seja necessário, a leitura de dados através do teclado, na janela de diálogo do DrScheme (com `read`).

O bloco 2 define a lógica de jogo, correspondendo ao primeiro procedimento do procedimento `game-loop`, e subdivide-se nos blocos de entrada e processamento.

Finalmente, o bloco 3 implementa as funcionalidades de saída, apresentando os elementos do jogo no ecrã.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Observe que para este primeiro programa, para além da biblioteca `util.ss`, foram utilizadas 2 bibliotecas adicionais para o desenho gráfico (`image.ss`) e para a leitura do teclado (`keyboard.ss`).

Talvez seja agora uma boa ocasião para dar uma rápida leitura ao pequeno capítulo *Keyboard*, do documento já referido sobre o Allegro: [Allegro: Multimedia library and game utilities](#)

No bloco 1.2, surge a definição de um procedimento inteiro, que será utilizado ao longo desta Parte para determinar as coordenadas de ecrã, e que transforma qualquer valor no seu valor inteiro arredondado e exacto (sem ser número racional, situação que causaria algumas perturbações ao Allegro).

```
(define inteiro
  (lambda (x)
    (inexact->exact (round x))))
```

## As peças do jogo

Qualquer jogo de computador precisa de um conjunto de variáveis que armazenem as propriedades dos diversos elementos ou peças do jogo. Neste caso particular, consideram-se os dois jogadores, as suas raquetes e a bola.

Em seguida apresentam-se os diversos componentes de jogo, as respectivas variáveis a considerar e o código a inserir no bloco 1.1.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

- O número de golos marcados pelo jogador 1.

(golos1 0) ; golos do jogador 1

- O número de golos marcados pelo jogador 2.

(golos2 0) ; golos do jogador 2

- A posição da raquete do jogador 1 no ecrã e a sua cor.

(x-raquetel 0)  
(y-raquetel 0)  
(cor-raquetel (image-color 0 0 255)) ; Azul

- A posição da raquete do jogador 2 no ecrã e a sua cor.

(x-raquete2 0)  
(y-raquete2 0)  
(cor-raquete2 (image-color 255 0 0)) ; Vermelho

- A bola possui uma determinada posição, uma determinada velocidade, uma determinada dimensão definida pelo seu raio e uma determinada cor.

(cor-bola)  
(x-bola 0.0)  
(y-bola 0.0)  
(x-vel-bola 0.0)  
(y-vel-bola 0.0)  
(raio-bola 25)

E cá temos a parte inicial do jogo Pong, com o código do bloco 1.1. Tudo o resto se mantém.

```
(require (planet "util.ss" ("kazzmir" "allegro.plt" 1 0)))  
(require (prefix image- (planet "image.ss" ("kazzmir" "allegro.plt" 1 0))))  
(require (planet "keyboard.ss" ("kazzmir" "allegro.plt" 1 0)))  
;;;;;;;  
; função principal do jogo Pong ;  
;;;;;;;  
(define jogo-pong  
  (lambda (x-janela y-janela nivel)
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
;::::::::::::::::::;
; 1. Definições ;
;::::::::::::::::::;
(letrec (
;::::::::::::::::::;
; 1.1. Variaveis do jogo ;
;::::::::::::::::::;
(pi (acos -1)) ; valor de Pi

; jogador 1
(golos1 0) ; golos do jogador 1

; jogador 2
(golos2 0) ; golos do jogador 2

; raquete do jogador 1
(x-raquetel 0)
(y-raquetel 0)
(corr-raquetel (image-color 0 0 255)) ; Azul

; raquete do jogador 2
(x-raquete2 0)
(y-raquete2 0)
(corr-raquete2 (image-color 255 0 0)) ; Vermelho

; bola
(x-bola 0.0)
(y-bola 0.0)
(x-vel-bola 0.0)
(y-vel-bola 0.0)
(corr-bola (image-color 255 255 0)) ; Amarelo

;::::::::::::::::::;
; 1.2. Procedimentos auxiliares ;
;::::::::::::::::::;
; retorna inteiro exacto
(inteiro
(lambda (x)
(inexact->exact (round x)))))

;inicializacao do Alegro
...
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Experimente o procedimento `jogo-pong`. O ecrã abre com

> `(jogo-pong 800 600 1)`

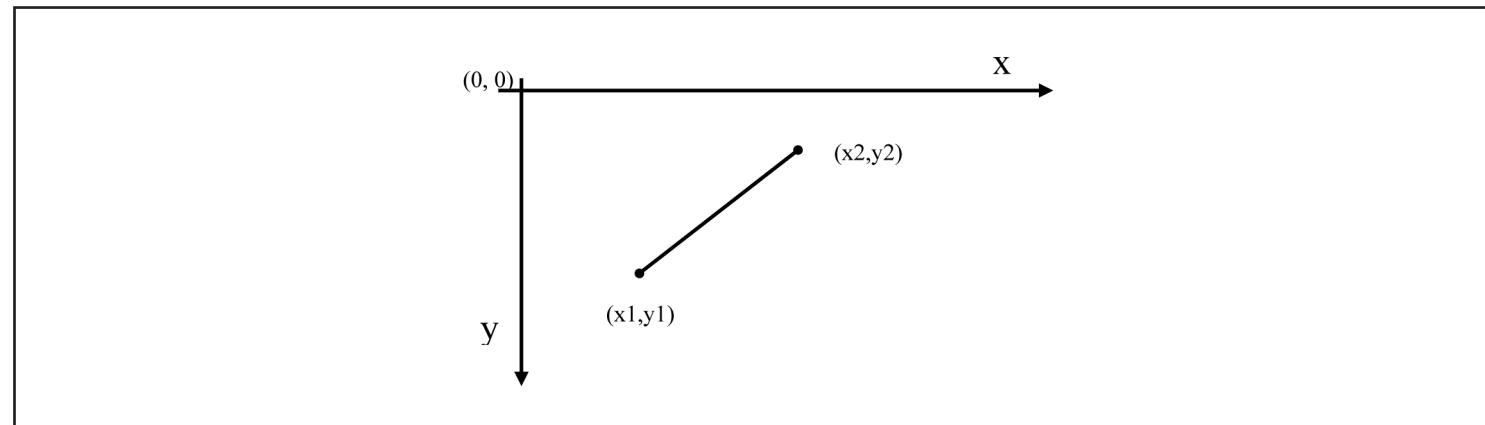
e o programa termina premindo a tecla `Esc`.

Acrescente mais duas variáveis para guardar o nome dos dois jogadores.

Adicione a funcionalidade que permite que o nome do jogador 1 seja lido do teclado através do procedimento `read` e o nome do jogador 2 seja "Computador". Mais tarde, estes nomes poderão ser utilizados nas mensagens do jogo e no marcador da pontuação.

### Desenhandoo no ecrã

Uma das saídas mais importantes para a maioria dos jogos de computador é a imagem (daí que sejam também conhecidos por videojogos). As imagens são apresentadas em ecrãs de computador segundo um sistema de referência cartesiano, onde cada objecto geométrico é representado pelos seus vértices, cada um dos quais se associa a um par de coordenadas  $(x, y)$ , tal como apresentado na figura.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

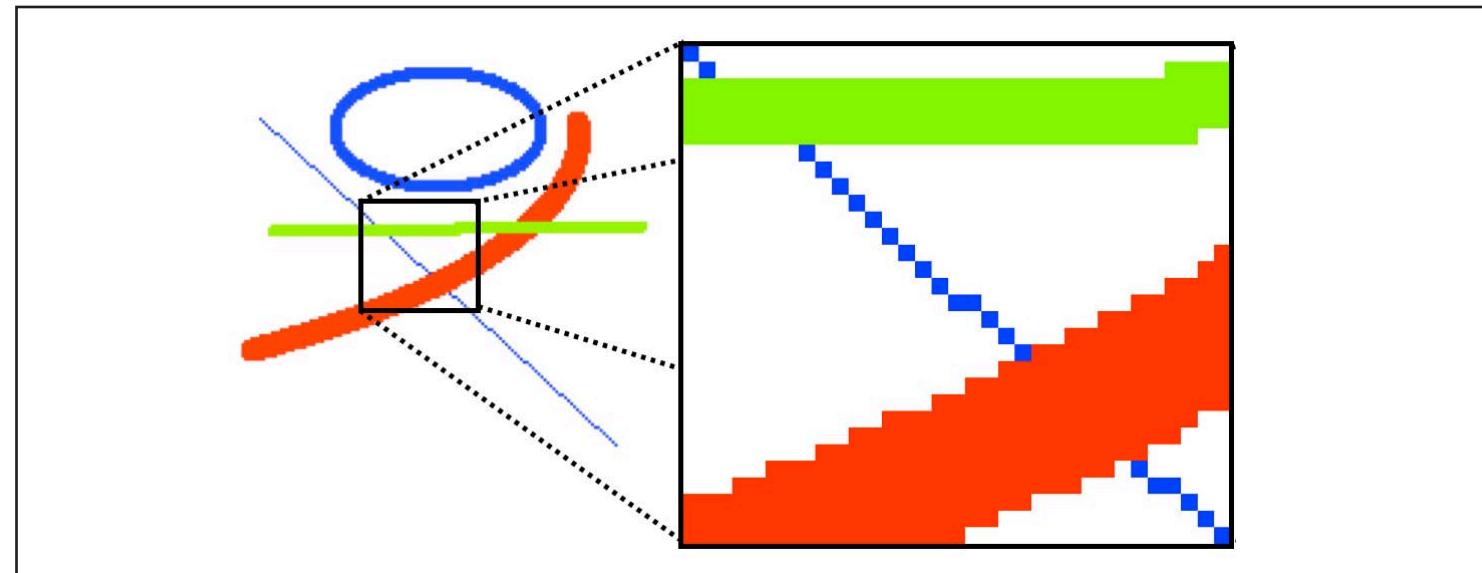
R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



No Allegro, os gráficos são apresentados em *bitmaps*, ou seja, matrizes de *pixels*. Um *pixel* (*PICTure ELelement*) é o menor elemento que compõe uma imagem e ao qual é possível atribuir uma cor. Desta forma, qualquer imagem em Allegro, inclusivamente o ecrã, é tratada como uma matriz, com uma determinada dimensão especificada em *pixel*, (largura x altura), na qual se pode atribuir um determinado valor de cor a cada *pixel* para formar qualquer gráfico, tal como é apresentado na figura.



O Allegro não opera directamente no ecrã, por razões que têm que ver com restrições impostas pelos sistemas operativos, mas sim em janelas gráficas. As coordenadas destas janelas, bem como dos restantes *bitmaps*, são valores inteiros entre 0 e as dimensões da imagem (subtraídos de 1). Ou seja, tomando como exemplo uma janela com dimensões 800 x 600, a coordenada x pode tomar valores entre 0 e 799 e a coordenada y pode tomar valores entre 0 e 599. A origem do referencial é o canto superior esquerdo do *bitmap*.



**Por momentos vai deixar o desenvolvimento do jogo Pong para fazer umas experiências sem utilizar o procedimento `game-loop`.**



Experimente o código apresentado.

O ecrã abre com

> (desenha)  
e o programa termina premindo uma tecla qualquer.

A definição da cor a atribuir a cada *pixel* é realizada através da especificação de 3 componentes que quantificam a contribuição de cada uma das três cores do modelo RGB. O modelo de cores RGB é um modelo aditivo no qual o vermelho, o verde e o azul são combinados entre si para reproduzir outras cores, tal como pode ser observado na figura.

O nome do modelo e a abreviação RGB vêm das três cores primárias: vermelho, verde e azul (*Red, Green e Blue*, em inglês).

No código apresentado anteriormente, a chamada `(image-color 0 0 255)` origina a cor de fundo a azul (componente *Blue* com valor máximo e as restantes a zero), enquanto as chamadas `(image-color 255 255 255)` e `(image-color 0 0 0)` definem as cores branco e preto, respectivamente, para o texto e para a caixa de texto.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Para criar uma determinada imagem bastará assim preencher todos os *pixels* com a cor desejada, através do procedimento `image-putpixel`. Apresenta-se, em seguida, um procedimento para desenhar, num determinado *bitmap* (utilizaremos a própria janela gráfica, obtida com a chamada (`image-screen`)), riscas verticais de uma determinada cor e determinada largura (esta definida em *pixels*). As dimensões do *bitmap* são determinadas com chamadas aos procedimentos `image-width` e `image-height` do Allegro.

```
; desenha riscas verticais
(define desenha-riscas
  (lambda (bitmap x y largura cor)
    (let ((maxx (image-width bitmap))
          (maxy (image-height bitmap)))
      (if (< (+ x largura) maxx)
          (begin
            (desenha-risca-vertical bitmap x y maxy largura cor)
            (desenha-riscas bitmap (+ x (* 2 largura)) y largura cor))))))
; desenha uma risca vertical
(define desenha-risca-vertical
  (lambda (bitmap x y altura largura cor)
    (if (> altura 0)
        (begin
          (desenha-risca-horizontal bitmap x y largura cor)
          (desenha-risca-vertical bitmap x (add1 y) (sub1 altura) largura cor)))))

; desenha uma linha horizontal
(define desenha-risca-horizontal
  (lambda (bitmap x y largura cor)
    (if (> largura 1)
        (begin
          (putpixel bitmap x y cor)
          (desenha-risca-horizontal bitmap (add1 x) y (sub1 largura) cor))))
```



Teste o procedimento `desenha-riscas` fazendo

`(desenha)`

O programa termina actuando uma tecla qualquer.

A partir deste procedimento pode desenvolver outros procedimentos para criar padrões diferentes como, por exemplo, um padrão axadrezado.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

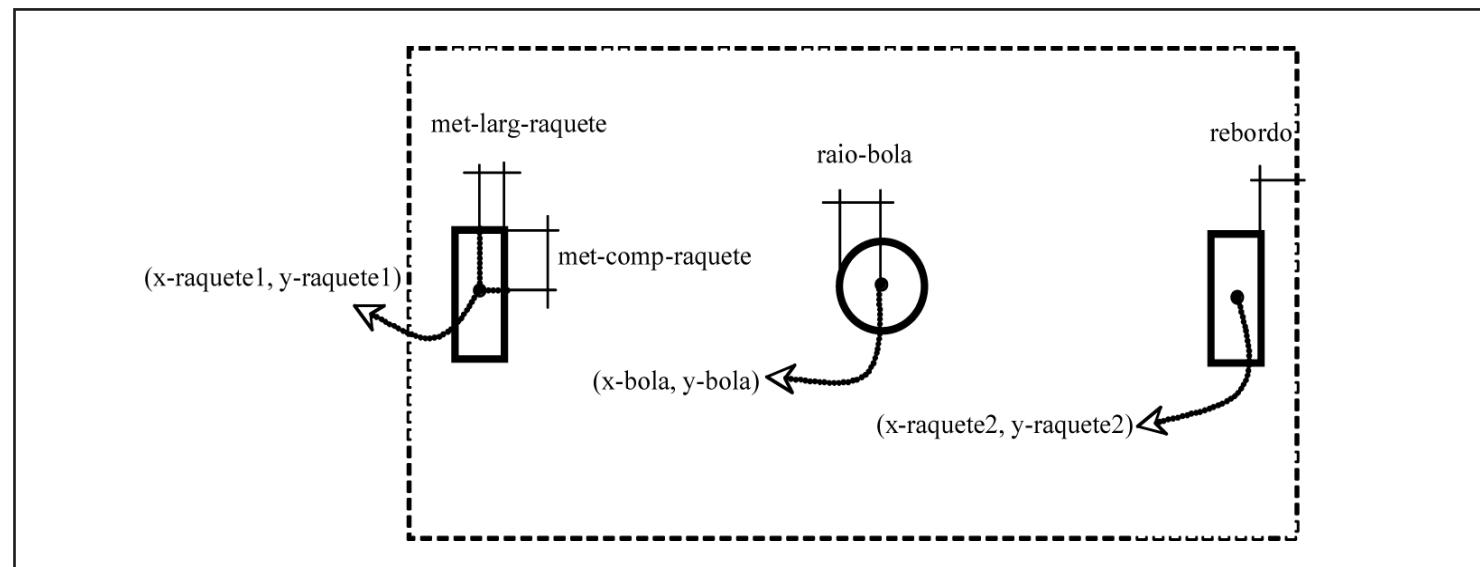


Trabalhar apenas com *pixels*, um a um (utilizando o procedimento `putpixel`), como aconteceu com os padrões, não é tarefa nada fácil.

Imagine que *pixels* teria que escolher para desenhar uma circunferência de raio 55.

Mas ainda bem que existem bibliotecas gráficas que, tal como o Allegro, disponibilizam várias funcionalidades para o desenho de figuras geométricas (linhas, triângulos, rectângulos, polígonos, etc).

Voltando ao nosso jogo do Pong, verifique a necessidade de desenhar a bola e ambas as raquetes. Para a primeira, utilizaremos um círculo, enquanto para as raquetes utilizaremos rectângulos.



Parece ser uma boa ocasião para fazer uma nova consulta a [Allegro: Multimedia library and game utilities](#) no que se refere a `image-circle-fill` e `image-rectangle-fill`.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



O código para o desenho da bola e das raquetes ficará definido pelas chamadas aos procedimentos `image-circle-fill` e `image-rectangle-fill`:

```
; desenha bola
(image-circle-fill imagem (inteiro x-bola) (inteiro y-bola) raio-bola cor-bola)
; raquete do jogador 1
(image-rectangle-fill imagem
(- x-raquete1 met-larg-raquete) (- y-raquete1 met-comp-raquete)
(+ x-raquete1 met-larg-raquete) (+ y-raquete1 met-comp-raquete)
cor-raquete1)
; raquete do jogador 2
(image-rectangle-fill imagem
(- x-raquete2 met-larg-raquete) (- y-raquete2 met-comp-raquete)
(+ x-raquete2 met-larg-raquete) (+ y-raquete2 met-comp-raquete)
cor-raquete2)
```



Certamente reparou que na chamada ao procedimento `image-circle-fill` foi necessário utilizar o procedimento `inteiro` para converter os parâmetros reais em números inteiros. Já na chamada ao procedimento `image-rectangle-fill` tal não foi necessário, pois todas as variáveis utilizadas nas expressões dos seus argumentos são inteiros.

Em conclusão, nas funções de desenho, sempre que as expressões dos argumentos contenham valores reais, deverá utilizar o procedimento `inteiro`.

No entanto, é necessário definir, no bloco 1.1, algumas variáveis auxiliares para controlar a dimensão das raquetes consoante o nível de dificuldade do jogo, bem como a distância entre estas raquetes e os limites da janela (rebordo):

```
; metade do comprimento da raquete: depende do nível
(met-comp-raquete (quotient (min x-janela y-janela) (* 4 (+ 1 nivel)))))

; metade da largura da raquete: 1/5 do seu comprimento
(met-larg-raquete (quotient met-comp-raquete 5))

; rebordo - distância entre a raquete e o limite da janela
(rebordo 10)
```

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



Como é que a dimensão das raquetes se relaciona com o nível de dificuldade do jogo, sabendo que quanto maior for o nível maior será a dificuldade?

Se tiver dificuldades em responder a esta pergunta, tente calcular manualmente

```
(met-comp-raquete (quotient (min x-janela y-janela)
                               (* 4 (+ 1 nivel))))
```

para os níveis 1, 2 e 3, numa janela de 800 x 600 e analise os resultados...

As dimensões das raquetes são definidas de acordo com as dimensões da janela e do nível de jogo, de forma a que, para níveis de dificuldade superiores, as suas dimensões sejam menores, exigindo assim uma maior perícia. Por razões de eficiência, foram definidas as metades dos valores da largura e do comprimento das raquetes para evitar as divisões por 2, uma vez que a origem das coordenadas das raquetes é o seu centro.

Para reiniciar o jogo foi ainda definido um novo procedimento auxiliar, a colocar no bloco 1.2, para a inicialização das variáveis de jogo. Este procedimento auxiliar deverá ser invocado logo no início (após o `easy-init`), ou sempre que o jogo seja reiniciado.

```
(require (planet "util.ss" ("kazzmir" "allegro.plt" 1 0)))
(require (prefix image- (planet "image.ss" ("kazzmir" "allegro.plt" 1 0))))
(require (planet "keyboard.ss" ("kazzmir" "allegro.plt" 1 0)))

;;;;;;;;;;;;;;;;;;
; função principal do jogo Pong ;
;;;;;;;;;;;;;;;;;;
(define jogo-pong
  (lambda (x-janela y-janela nivel)
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
;;;;;;;;;;;
; 1. Definições ;
;;;;;;;;;;
(letrec (
    ;;;;;;;;;;;;;;;
    ; 1.1. Variaveis do jogo ;
;;;;;;;;;;;;;;
(pi (acos -1)) ; valor de Pi
; jogador 1
(golos1 0) ; golos do jogador 1
(nome1 (symbol->string (begin (display "Nome do jogador 1: ") (read))))
; jogador 2
(golos2 0) ; golos do jogador 2
(nome2 "Computador") ; nome do jogador 2
; raquete do jogador 1
(x-raquete1 0)
(y-raquete1 0)
(cor-raquete1 (image-color 0 0 255)) ; Azul
; raquete do jogador 2
(x-raquete2 0)
(y-raquete2 0)
(cor-raquete2 (image-color 255 0 0)) ; Vermelho

; bola
(x-bola 0.0)
(y-bola 0.0)
(x-vel-bola 0.0)
(y-vel-bola 0.0)
(raio-bola 25)
(cor-bola (image-color 255 255 0)) ; Amarelo
; metade do comprimento da raquete: depende do nivel
(met-comp-raquete (quotient (min x-janela y-janela) (* 4 (+ 1 nivel))))
; metade da largura da raquete: 1/5 do seu comprimento
(met-larg-raquete (quotient met-comp-raquete 5))
; rebordo - distancia entre a raquete e o limite da janela
(rebordo 10)
;;;;;;;;;;
; 1.2. Procedimentos auxiliares ;
;;;;;;;;;;
; retorna inteiro exacto
(inteiro
  (lambda (x)
    (inexact->exact (round x))))
```



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
; inicializacao do jogo
(inicializacao
  (lambda ()
    ; golos
    (set! golos1 0)
    (set! golos2 0)
    ; bola
    (set! cor-bola (image-color 255 255 0)) ; amarelo
    (set! x-bola (/ x-janela 2))
    (set! y-bola (/ y-janela 2))
    ;raquete 1
    (set! cor-raquetel (image-color 0 0 255)) ; azul
    (set! x-raquetel (+ rebordo met-larg-raquete))
    (set! y-raquetel (inteiro (/ y-janela 2)))
    ;raquete 2
    (set! cor-raquete2 (image-color 255 0 0)) ; vermelho
    (set! x-raquete2 (- x-janela met-larg-raquete rebordo))
    (set! y-raquete2 y-raquetel)))

;inicializacao do Allegro
; (easy-init width height depth mode)
;
(easy-init x-janela y-janela 16 'WINDOWED) ; ou 'FULLSCREEN

; inicializacao do jogo
(inicializacao)

; O motor de jogo
; (game-loop logic-proc draw-proc game-delay)
;

(game-loop
;;;;;;
; 2. Logica do jogo ;
;;;;;;
(lambda ()
  (if (keypressed? 'ESC)
      #t ; devolvendo #t, termina o ciclo...
  (begin
```



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



```
;;;;;;;  
; 2.1. Entrada ;  
;;;;;;;  
  
;;;;;;;  
; 2.2. Processamento do jogo ;  
;;;;;;;  
; devolvendo #f, repete novamente o ciclo  
#f)))  
  
;;;;;;;  
; 3. Saída grafica ;  
;;;;;;;  
(lambda (imagem)  
  ; desenha bola  
  (image-circle-fill imagem (inteiro x-bola) (inteiro y-bola) raio-bola cor-bola)  
  ; raquete do jogador 1  
  (image-rectangle-fill imagem  
    (- x-raquete1 met-larg-raquete)  
    (- y-raquete1 met-comp-raquete)  
    (+ x-raquete1 met-larg-raquete)  
    (+ y-raquete1 met-comp-raquete)  
    cor-raquete1)  
  ; raquete do jogador 2  
  (image-rectangle-fill imagem  
    (- x-raquete2 met-larg-raquete)  
    (- y-raquete2 met-comp-raquete)  
    (+ x-raquete2 met-larg-raquete)  
    (+ y-raquete2 met-comp-raquete)  
    cor-raquete2))  
  
  ; determina temporizacao do motor de jogo com base na velocidade desejada  
  (frames-per-second 30)))  
  
  ; fecha janela Allegro e termina programa  
  (easy-exit)  
  (display "Obrigado por ter jogado o Pong.")))
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Experimente o código de visualização com diversas resoluções (dimensões) da janela gráfica e do nível. No início, o programa pede o nome do jogador que, nesta versão, não servirá para nada.

O ecrã, para uma janela 800 x 600 e um jogo de nível 1, abre com

> (jogo-pong 800 600 1)

e o programa termina premindo a tecla Esc.

No início, o programa pede o nome do jogador que, nesta versão, não servirá para nada.

Apenas para ganhar um pouco de experiência com o Allegro, tente desenhar as raquetes com elipses, utilizando o procedimento `image-ellipse-fill`.

Repare que a sintaxe deste procedimento é bastante diferente de `image-rectangle-fill`, assemelhando-se mais à de `image-circle-fill`.

(`image-ellipse-fill imagem x y raio-x raio-y cor`)

No entanto, o melhor seria consultar [Allegro: Multimedia library and game utilities](#).

Os actores do jogo aparecem finalmente no ecrã: as duas raquetes e a bola. Mas o ecrã continua estático, não havendo possibilidade de interacção com estes elementos...

O que faltará para o Pong se tornar um verdadeiro jogo?

### Há que fazer uma intervenção estética!

Na realidade, a maior parte dos jogos de computador usa gráficos mais elaborados. Uma das técnicas utilizadas para dar mais realismo aos elementos do jogo denomina-se *sprite*. Um *sprite* é uma imagem que representa um determinado objecto e que se misturará com o fundo ou com outras imagens, mantendo a ordenação aparente entre estas, bem como as transparências dos objectos. Há que ter em consideração que qualquer imagem é sempre um rectângulo e para definir personagens ou objectos com outra configuração é necessário impôr transparência na parte da imagem que não pertence à silhueta do personagem ou objecto representado. Imagine um *sprite* de um automóvel: este representa a forma do veículo, mas, através dos vidros, deve ser observável a paisagem do fundo.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Para definir um *sprite* é necessário associar uma máscara a uma imagem que indicará os *pixels* que não pertencem ao objecto.

O Allegro permite o desenho de *sprites* com o procedimento `image-draw`. É apenas necessário definir uma imagem que, para ter transparências, terá que definir esses *pixels* com a cor mangenta, (255 0 255), que é utilizada como máscara, tal como é representado na figura.

Para representar a bola do jogo por um *sprite* é necessário, em primeiro lugar, carregar a imagem correspondente à bola (já com a máscara definida).

O ficheiro com a imagem `bola.bmp` foi incorporado em `PLT\collects\user-feup`.

Assim, em primeiro lugar, iremos incluir uma biblioteca que nos disponibilize o procedimento `getPathCompleto`, que nos retorna o *path* do directório `user-feup`, e que será utilizado futuramente para outros ficheiros instalados no directório `user-feup` (como verá já a seguir). Para tal é necessário incluir no início do código o seguinte :

```
(require (lib "pathuserfeup.scm" "user-feup"))
```



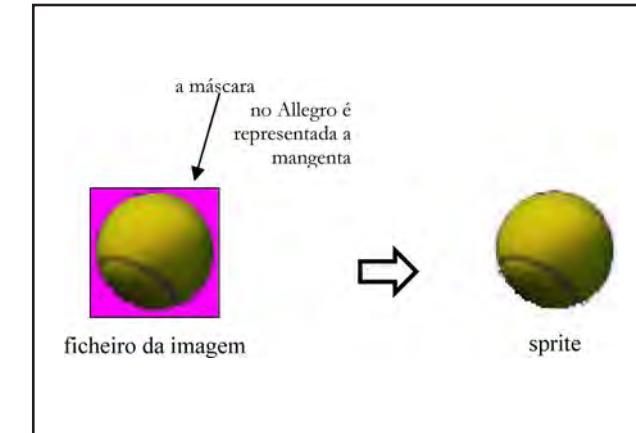
**Este procedimento apenas funciona em sistemas operativos Windows.  
Para UNIX ou Mac OS, deverá utilizar a biblioteca `xpathuserfeup.scm`.**

Bastará agora criar, no bloco 1.1, uma variável para guardar a imagem a ser utilizada como *sprite*.

```
(sprite-bola #f)
```

E carregar a imagem do *sprite*, a partir do ficheiro, chamando a seguinte instrução no procedimento `inicializacao`:

```
(set! sprite-bola (image-create-from-file (getPathCompleto "bola.bmp"))))
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Finalmente, para desenhar o *sprite* basta chamar o procedimento `image-draw`.

```
(image-draw imagem sprite x y)
```

Como este procedimento define as coordenadas do canto superior esquerdo da imagem do *sprite*, torna-se necessário subtrair o raio da bola às coordenadas do seu centro. Para concluir este processo, basta substituir a instrução de desenho da bola, no bloco 3, para:

```
(image-draw imagem sprite-bola (- (inteiro x-bola) raio-bola) (- (inteiro y-bola) raio-bola))
```



Experimente o novo jogo com *sprites*.

O ecrã abre com

```
> (jogo-pong 800 600 1)
```

e o programa termina premindo a tecla `Esc`.

No início, o programa pede o nome do jogador que, nesta versão, não servirá para nada.

Experimente criar também *sprites* para as raquetes. Para isso pode utilizar a imagem `raquete.bmp` em `user-feup`.



Já deve ter reparado em diversos jogos que alguns *sprites* são animados.

Este efeito é realizado pela definição de diversas imagens correspondentes a posições distintas desse elemento do jogo, e a imagem do *sprite* vai sendo trocada num determinado intervalo de tempo (definido normalmente por um número de ciclos).

Experimente colocar o *sprite* da bola a rodar. Pode rodar a imagem 8 vezes para obter 8 imagens correspondentes a 8 ângulos da bola, para a sequência de animação.

Em `user-feup` foram colocadas 8 imagens da bola (`bola1.bmp` a `bola8.bmp`).

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Agora o rato entra em jogo

Um jogo de computador é uma aplicação interactiva e como tal é essencial definir a forma como o jogador poderá interagir com ele. Para este jogo do Pong, optou-se por utilizar o rato para mover a raquete do jogador 1, sendo a raquete 2 movimentada pelo computador.

Para poder utilizar as funcionalidades associadas ao rato é necessário incluir o módulo `mouse.ss` do `allegro.plt`.

```
(require (prefix mouse- (planet "mouse.ss" ("kazzmir" "allegro.plt" 1 0))))
```



Também para este módulo se utilizou um prefixo, neste caso `mouse-`, para evitar colisões com os nomes dos procedimentos de outros módulos.

O conjunto de procedimentos é diminuto, mas permite obter toda a informação relevante associada ao rato: o seu posicionamento, através de `mouse-x` e `mouse-y`, o estado dos botões, com os procedimentos `mouse-left-click` e `mouse-right-click` e o seu deslocamento com `mouse-get-mickeys`.

A raquete do jogador 1 apenas se pode movimentar na vertical, pelo que utilizaremos o procedimento `mouse-y` para atribuir o novo valor da variável `yraquete1`, como poderá ver no extracto de código correspondente ao bloco 2.1 (Entrada):

```
...
(game-loop
  ; 2. Logica do jogo ;
  (lambda ()
    (if (keypressed? 'ESC)
        #t ; devolvendo #t, termina o ciclo...
        (begin
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
;;;;;;;;;;;  
; 2.1. Entrada ;  
;;;;;;;;;;;  
  
; move o jogador 1 - com o rato  
(set! y-raquetel (mouse-y))  
  
;;;;;;;;;;;  
; 2.2. Processamento do jogo ;  
;;;;;;;;;;;  
  
; devolvendo #f, repete novamente o ciclo  
#f)))  
  
;;;;;;;;;;;  
; 3. Saida grafica ;  
;;;;;;;;;;;  
...
```



Teste esta nova versão do jogo Pong e verifique que uma das raquetes é movimentada com o rato, quando o cursor se desloca dentro da janela do jogo.

O ecrã abre com

> (jogo-pong 800 600 1)

e o programa termina premindo a tecla Esc.

No início, o programa pede o nome do jogador que, nesta versão, ainda não será utilizado.

Como exercício, acrescente a funcionalidade de apresentar no fundo do ecrã (centrado) um texto com as coordenadas do rato, sempre que seja premido o seu botão esquerdo.

### Lendo do teclado

O rato é, tipicamente, um dispositivo apontador, o que significa que serve muito bem para apontar coordenadas no ecrã. Embora o rato possua botões, o dispositivo por excelência para indicar opções e comandos é o teclado. Na realidade, o teclado já está a ser utilizado para verificar quando a tecla Esc é premida, de forma a terminar o jogo. O procedimento que tem sido utilizado é o `keypressed?`, que devolve `#t` caso a tecla especificada seja premida.

```
(if (keypressed? 'ESC)  
    #t ; devolvendo #t, termina o ciclo...  
  (begin  
    ...
```

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Como alternativa à utilização do rato, no nosso jogo do Pong, iremos permitir a utilização do teclado. Para tal, a tecla T (ou t) inibirá a acção do rato, passando a ser utilizadas as setas do cursor (cima e baixo). Por outro lado, a tecla R (ou r) volta a atribuir o controlo da raquete ao rato.

Esta funcionalidade é disponibilizada introduzindo algumas alterações no bloco 2.1, como poderá verificar no extracto de código que se segue:

```
...
(game-loop

; 2. Logica do jogo ;
;;;;;;
(lambda ()
  (if (keypressed? 'ESC)
      #t ; devolvendo #t, termina o ciclo...
      (begin
        ;;;;;;;
        ; 2.1. Entrada ;
        ;;;;;;;

        (if (equal? controlo 'rato)
            ; rato
            (begin
              ; passa o controlo para o teclado
              (if (keypressed? 'T)
                  (set! controlo 'teclado))

              ; move o jogador 1 - com o rato
              (set! y-raquetel (mouse-y)))

            ;teclado

            (cond
              ; passa o controlo para o teclado
              ((keypressed? 'R)
               (set! controlo 'rato))
              ; cursor cima - sobe a raquete
              ((keypressed? 'UP)
               (if (> y-raquetel 0)
                   (set! y-raquetel (- y-raquetel 5)))
               (clear-keyboard))
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
; cursor baixo - baixa a raquete
((keypressed? 'DOWN)
 (if (< y-raquetel y-janela)
     (set! y-raquetel (+ y-raquetel 5)))
 (clear-keyboard))))
```

```
;;;;;;;;;;;
; 2.2. Processamento do jogo ;
;;;;;;;;;;;
```

```
; devolvendo #f, repete novamente o ciclo
#f)))
```

```
;;;;;;;;;;
; 3. Saida grafica ;
;;;;;;;;;;
...;
```

Para o programa funcionar correctamente, é necessário acrescentar uma nova variável, controlo, inicializada com o símbolo 'rato. O procedimento inicializacao deverá agora incluir esta variável.



Talvez seja uma boa ocasião para ler novamente o capítulo **Keyboard** do documento: [Allegro: Multimedia library and game utilities](#).

Aproveite para encontrar uma justificação para a utilização de `clear-keyboard` e também para ver como se identificam as setas para cima e para baixo.

Para poder utilizar as funcionalidades associadas ao teclado é necessário incluir o módulo `keyboard.ss` do `allegro.plt`.

```
(require (planet "keyboard.ss" ("kazzmir" "allegro.plt" 1 0)))
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Teste esta nova versão do jogo Pong e verifique que ao actuar a tecla T (ou t) o controlo da raquete passa para as setas do cursor (cima e baixo). Por outro lado, a actuação da tecla R (ou r) volta a atribuir o controlo da raquete ao rato.

O ecrã abre com

> (jogo-pong 800 600 1)

e o programa termina premindo a tecla Esc.

No início, o programa pede o nome do jogador que, nesta versão, continua a não ser utilizado.

Agora, como exercício, considere a seguinte situação: as versões mais antigas do jogo do Pong, para consola, eram a preto e branco. De forma a imitar essas versões mais antigas, utilize a tecla P (ou p) para alterar as cores das raquetas e da bola para preto e branco e a tecla C (ou c) para voltar a colocar o jogo a cores.

**Pista:** Não se esqueça de, na situação a preto e branco, mudar a forma de desenhar a bola, substituindo o *sprite* por um círculo, como inicialmente...

## Movimentando a bola

Entretanto, o nosso jogo do Pong já se tornou interativo, dado que é possível controlar a raquete do jogador 1, tanto através do rato como do teclado. No entanto, uma grande maioria de jogos de computador pressupõe a movimentação de diversos elementos do jogo. Neste caso particular, a acção do jogo baseia-se na movimentação da bola (deslocamento e batida nas paredes e nas raquetes). A simulação deste tipo de movimentos poderá ser efectuada desde uma forma muito simples até à utilização de uma biblioteca de funções para simulação dos processos físicos.

Para este jogo utilizaremos um mecanismo relativamente simples, baseado num vector velocidade, para actualizar a posição da bola a cada ciclo de jogo. Este vector é representado por duas variáveis, x-vel-bola e y-vel-bola, que representam as suas componentes x e y, e que já se encontram definidas no bloco 1.1. Quando a bola bate em algum obstáculo (parede ou raquete), uma das componentes (x ou y) do vector velocidade é tornada simétrica.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

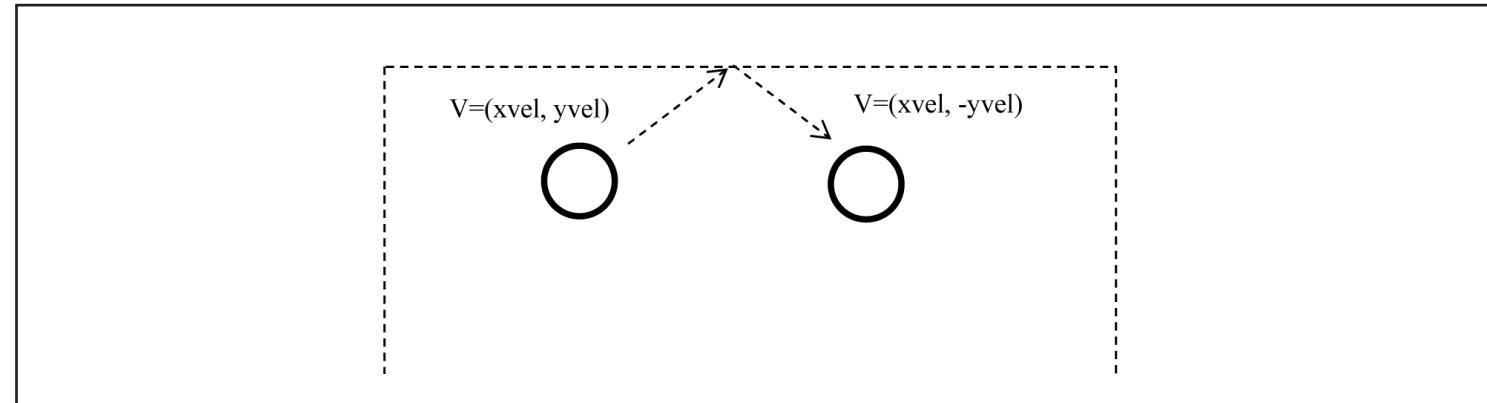
R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Exemplificando, quando a bola bate na parede de cima ou de baixo, a componente y da velocidade é tornada simétrica (por exemplo, se a bola subia, passa a descer) e, quando atinge qualquer uma das raquetes, é a componente x que é tornada simétrica, como pode ser observado na figura.



O módulo da velocidade será dependente do nível de dificuldade (maior dificuldade corresponderá a maior velocidade) e o ângulo inicial deverá ser obtido aleatoriamente. Este ângulo não deverá fazer mais de 45º com a horizontal de forma a não tornar o jogo demasiado lento (no limite, se a velocidade fosse vertical, a bola nunca atingiria as raquetes!). A posição inicial da bola será no centro do ecrã. Será assim necessário criar um procedimento `lanca-bola` no bloco 1.2, como pode ver no extracto de código que se segue. Este procedimento deverá ser utilizado no início do jogo, pelo que se deve inserir uma chamada a este no procedimento `inicializacao` ou sempre que haja um golo.



Procure identificar no extracto de código que se segue as situações em que `lanca-bola` é chamado.

# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
...
;;;;;;
; 1.2. Procedimentos auxiliares ;
;;;;;;
...

; lanca a bola
(lanca-bola
(lambda ()
  (let ((vel-bola (* nivel 4)) ; velocidade
        (ang-bola (* (random) pi))) ; angulo (cobre 180 graus)
    ; evita angulos superiores a 45° com a horizontal
    (if (>= ang-bola (/ pi 2))
        (set! ang-bola (+ ang-bola (/ pi 2))))
        (set! ang-bola (- ang-bola (/ pi 4))))
    ; componentes x e y do vector - conversao polar -> rectangular
    (set! x-vel-bola (* vel-bola (cos ang-bola)))
    (set! y-vel-bola (* vel-bola (sin ang-bola)))
    ; posicao da bola
    (set! x-bola (/ x-janela 2))
    (set! y-bola (/ y-janela 2)))))

; inicializacao do jogo
(inicializacao
(lambda ()
  ; golos
  (set! golos1 0)
  (set! golos2 0)
  ; bola
  (set! cor-bola (image-color 255 255 0)) ; amarelo
  (set! x-bola (/ x-janela 2))
  (set! y-bola (/ y-janela 2))
  ; raquete 1
  (set! cor-raquetel (image-color 0 0 255)) ; azul
  (set! x-raquetel (+ rebordo met-larg-raquete))
  (set! y-raquetel (inteiro (/ y-janela 2)))
  ; raquete 2
  (set! cor-raquete2 (image-color 255 0 0)) ; vermelho
  (set! x-raquete2 (- x-janela met-larg-raquete rebordo))
  (set! y-raquete2 y-raquetel)
  ; controlo do jogo
  (set! controlo 'rato)
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
; lanca a bola
(lanca-bola)
; sprite
(set! sprite-bola (image-create-from-file (getPathCompleto "bola.bmp"))))

; inicializacao do Allegro
; (easy-init width height depth mode)
;
(easy-init x-janela y-janela 16 'WINDOWED) ; ou 'FULLSCREEN
; inicializacao do jogo
(inicializacao)
; O motor de jogo
; (game-loop logic-proc draw-proc game-delay)
;
(game-loop

;;;;;;;;;;
; 2. Logica do jogo ;
;;;;;;;;;;
(lambda ()
  (if (keypressed? 'ESC)
      #t ; devolvendo #t, termina...
  (begin
    ;;;;;;;;;;;
    ; 2.1. Entrada ;
    ;;;;;;;;;;;
    (if (equal? controlo 'rato)
        ...
        ...))));
```



Procure neste extracto de código, no bloco 2.2, as seguintes situações:

- 1- animação da bola através da actualização da sua posição a cada ciclo de jogo.
- 2- confrontação da bola com os limites do jogo (as coordenadas da bola são relativas ao seu centro e a confrontação terá em conta o seu raio).
- 3- confrontação da bola com as raquetes (não esquecer que uma raquete é representada por um rectângulo e não por uma simples recta).

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
;;;;;;;;;;;  
; 2.2. Processamento do jogo ;  
;;;;;;;;;;;  
  
; movimentacao da bola - actualizacao  
(set! x-bola (+ x-bola x-vel-bola))  
(set! y-bola (+ y-bola y-vel-bola))  
  
; confrontacao com as paredes  
(cond  
    ; limite superior ou inferior  
    ((or (< y-bola raio-bola) (> y-bola (- y-janela raio-bola)))  
     (set! y-vel-bola (- y-vel-bola)))  
    ; limite esquerdo - golo do jogador 2  
    ((< x-bola raio-bola)  
     (set! x-vel-bola (- x-vel-bola))  
     (set! golos2 (add1 golos2))  
     (lanca-bola))  
    ; limite direito - golo do jogador 1  
    ((> x-bola (- x-janela raio-bola))  
     (set! x-vel-bola (- x-vel-bola))  
     (set! golos1 (add1 golos1))  
     (lanca-bola)))  
  
; confrontacao com as raquetes  
(cond  
    ; raquete do jogador 1  
    ((and (< y-bola (+ y-raquetel met-comp-raquete))  
          (> y-bola (- y-raquetel met-comp-raquete))  
          (< x-bola (+ x-raquetel met-larg-raquete raio-bola)))  
     (set! x-vel-bola (- x-vel-bola)))  
    ; raquete do jogador 2  
    ((and (< y-bola (+ y-raquete2 met-comp-raquete))  
          (> y-bola (- y-raquete2 met-comp-raquete))  
          (> x-bola (- x-raquete2 met-larg-raquete raio-bola)))  
     (set! x-vel-bola (- x-vel-bola))) )  
  
    ; devolvendo #f, repete novamente o ciclo  
    #f)))  
  
;;;;;;;;;;;  
; 3. Saida grafica ;  
;;;;;;;;;;;  
...  
;
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Experimente o jogo, agora que a bola já tem movimento.

O ecrã abre com

```
> (jogo-pong 800 600 2)
```

e o programa termina premindo a tecla Esc.

Não se esqueça que às teclas T (teclado) e R (rato) estão associados tipos diferentes de funcionamento do jogo.

Certamente reparou que o movimento da bola é muito previsível. Seria certamente mais aliciante se fosse possível alterar o ângulo de reflexão da bola com o movimento da raquete (uma espécie de "efeito" aplicado à bola).

Para determinar a deslocação do rato, pode utilizar o procedimento `mouse-get-mickeys` e, assim, calcular o deslocamento vertical (Y) da raquete quando a mesma é atingida pela bola. A cada chamada ao procedimento `mouse-get-mickeys` são retornadas as diferenças em X e em Y relativamente à posição do rato no instante da chamada anterior a este procedimento.

Será necessário alterar o código que determina se a raquete do jogador 1 foi atingida:

```
; raquete do jogador 1
((and (< y-bola (+ y-raquetel met-comp-raquete))
      (> y-bola (- y-raquetel met-comp-raquete)))
   (set! x-vel-bola (- x-vel-bola))
   ; efeito
   (let-values (((dx dy) (mouse-get-mickeys)))
     (efeito-bola dx dy)))
```





INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



O procedimento `get-mickeys` (ou `mouse-get-mickeys` se considerar o prefixo `mouse-`) devolve (`values dx dy`), ou seja, devolve os valores `dx` e `dy`.

Faça algumas experiências com o Scheme do seguinte tipo:

```
> (values 23 4 56)
23
4
56
>
```

Com `let-values` é possível atribuir os valores devolvidos a variáveis, como se mostra no exemplo.

```
> (let-values (((dx dy) (values 5 6)))
  (display "dx: ")
  (display dx)
  (newline)
  (display "dy: ")
  (display dy))
dx: 5
dy: 6
>
```

Veja que `let-values` tem, relativamente ao `let`, mais um par adicional de () a envolver as variáveis. De resto, é tudo a mesma coisa.

O código chama um procedimento, `efeito-bola`, que deverá ser definido de forma a alterar a orientação do vector da velocidade da bola. Este procedimento irá alterar esse valor consoante o sinal do deslocamento vertical do rato, `dy`, desde que este seja superior a um determinado valor limite:

```
; efeito na bola
(efeito-bola
  (lambda (dx dy)
    (let* ((vel-escalar (sqrt (+ (* x-vel-bola x-vel-bola) (* y-vel-bola y-vel-bola)))))
      (vel-ang (atan (/ y-vel-bola x-vel-bola))))
      (if (< x-vel-bola 0)
          (set! vel-ang (+ vel-ang pi)))
    ; calcula novo angulo
    (if (< (abs vel-ang) (/ pi 3))
        (cond
          ((> dy 0) (set! vel-ang (+ vel-ang (/ pi 10)))))
          ((< dy 0) (set! vel-ang (- vel-ang (/ pi 10))))))
      ))
```

### INTRODUÇÃO

### 1 - O ESSENIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
; componentes x e y do vector - conversao polar -> rectangular
(set! x-vel-bola (* vel-escalar (cos vel-ang)))
(set! y-vel-bola (* vel-escalar (sin vel-ang)))))
```



O efeito da bola é calculado da seguinte forma:

1. Determina-se o módulo e a direcção do vector velocidade através do valor da velocidade escalar (**vel-escalar**) e o ângulo com o eixo dos X (**vel-ang**);
2. O efeito é conseguido alterando a direcção do vector velocidade, mantendo a mesma velocidade escalar;
3. O novo vector de velocidade calculado é convertido para as variáveis que controlam a deslocação da bola (**x-vel-bola** e **y-vel-bola**).

Identifique estes passos no código anterior.

O deslocamento determinado com a chamada do procedimento `mouse-get-mickeys`, corresponde ao deslocamento que foi efectuado pelo rato desde a chamada anterior a esse procedimento. Assim, para que o deslocamento reflita, o melhor possível, a realidade actual no que se refere à movimentação do rato, é conveniente chamar aquele procedimento a cada ciclo de jogo, acrescentando ao código do final do bloco 2 a seguinte instrução:

```
; actualiza deslocamento do rato
(mouse-get-mickeys)

; devolvendo #f, repete novamente o ciclo
#f)))
```



Experimente a nova versão do jogo, com "efeito" sobre a bola.

O ecrã abre com

```
> (jogo-pong 800 600 2)
```

e o programa termina premindo a tecla Esc.

Não se esqueça que às teclas T (teclado) e R (rato) estão associados tipos diferentes de funcionamento do jogo.

Tente alterar esta versão do jogo de forma a aproveitar o movimento horizontal do rato (X) para acelerar ou abrandar o movimento da bola. Neste caso utilize ambos os valores retornados pelo procedimento `mouse-get-mickeys`, dx e dy, para alterar, respectivamente, a velocidade escalar e a direcção do vector velocidade.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Já se ouvem sons!

Um jogo deve proporcionar o máximo de imersão e, como tal, deverá despertar a maior parte dos nossos sentidos. O som é extremamente importante para a orientação do ser humano e deve ser parte integrante do processo de produção de um jogo. Geralmente existem duas componentes sonoras num jogo de computador, à semelhança de um filme: os efeitos sonoros e a banda sonora.

Os efeitos sonoros permitem ao jogador aperceber-se de determinados acontecimentos (colisões, explosões, accionamento de mecanismos, etc.) enquanto a banda sonora proporciona determinado tipo de emoções em distintas fases do jogo.

O Allegro possibilita a reprodução de sons de ficheiro WAV ou VOC, sendo necessário incluir a biblioteca sound.ss do módulo allegro.plt.

```
(require (planet "sound.ss" ("kazzmir" "allegro.plt" 1 0)))
```



**Tem agora uma boa ocasião para consultar o pequeno capítulo Sound do documento: [Allegro: Multimedia library and game utilities.](#)**

A primeira fase corresponde ao carregamento em memória dos sons a reproduzir ao longo do jogo. No caso do jogo do Pong, para a reprodução dos efeitos sonoros iremos utilizar dois sons: um para o batimento da bola na raquete e outro para o batimento da bola na parede (estes ficheiros já foram incorporado na instalação do pacote user-feup).

Em primeiro lugar é necessário definir mais duas variáveis para estes sons, no bloco de inicializações (bloco 1.1):

```
;sons
(som-raquete #f)
(som-parede #f)
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



E em seguida, carregar estes sons através do procedimento `load-sound` do Allegro, que deverá ser colocado no procedimento `inicializacao`.

```
; sons
(set! som-raquete (load-sound (getPathCompleto "raquete.wav")))
(set! som-parede (load-sound (getPathCompleto "parede.wav")))
```

Para a reprodução do som, basta utilizar o procedimento `play-sound`, também do Allegro, cujos parâmetros (opcionais) permitem indicar a intensidade de som (`volume`), o balanceamento lateral (`pan`) e a velocidade de reprodução do som (`frequency`):

```
(play-sound sound [volume] [pan] [frequency])
```

Assim, para a reprodução do som de batimento na parede, deverá acrescentar, no bloco 2.2., ao código associado à determinação do limite superior ou inferior, o código:

```
; limite superior ou inferior
((or (< y-bola raio-bola) (> y-bola (- y-janela raio-bola)))
  (set! y-vel-bola (- y-vel-bola))
  (play-sound som-parede))
```

No caso do batimento das raquetes, associaremos ao som a lateralização do mesmo, consoante se trate do jogador 1 ou do jogador 2. Esta funcionalidade é definida no parâmetro `pan`, com um valor entre 0 (esquerda) e 255 (direita), sendo 128 o valor por omissão (sem lateralização).

Assim, para o jogador 1, situado à esquerda, o valor de `pan` deverá ser 0:

```
; raquete do jogador 1
((and (< y-bola (+ y-raquetel met-comp-raquete))
      (> y-bola (- y-raquetel met-comp-raquete))
      (< x-bola (+ x-raquetel met-larg-raquete raio-bola)))
  (set! x-vel-bola (- x-vel-bola))
  ; efeito
  (let-values (((dx dy) (mouse-get-mickeys)))
    (efeito-bola dx dy))
  (play-sound som-raquete 255 0))
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



E para o jogador 2, situado à direita, o valor de pan deverá ser 255:

```
; raquete do jogador 2
((and (< y-bola (+ y-raquete2 met-comp-raquete))
      (> y-bola (- y-raquete2 met-comp-raquete))
      (> x-bola (- x-raquete2 met-larg-raquete raiobola)))
   (set! x-vel-bola (- x-vel-bola))
   (play-sound som-raquete 255 255)))
```



Experimente o código desenvolvido para ouvir o som produzido pela bola quando toca nas raquetes e nas paredes.

O ecrã abre com

> (jogo-pong 800 600 2)

e o programa termina premindo a tecla Esc.

Não se esqueça que às teclas T (teclado) e R (rato) estão associados tipos diferentes de funcionamento do jogo.

Depois das experiências com o jogo, procure acrescentar um som de aplauso sempre que for marcado um golo (pode utilizar o ficheiro aplausos.wav em user-feup).

Embora não seja tão relevante, é comum acrescentar-se aos jogos uma banda sonora que permita transmitir o ambiente emocional de cada fase do jogo. Uma das opções é reproduzir, de forma cíclica, um determinado ficheiro de som com a banda sonora. Este tipo de funcionalidade pode ser implementada através do procedimento play-sound-looped do Allegro. Para parar a reprodução de um determinado som pode-se utilizar o procedimento stop-sound, também do Allegro.

Nem sempre a audição da banda sonora é desejada, pelo que deverá ser possível optar por este efeito. Para este jogo vai ser utilizada a tecla M (ou m) para ligar a banda sonora e a tecla S (ou s) para a desligar, de acordo com o código, a colocar no bloco 2.1. – Input.

```
; banda sonora
(if (keypressed? 'S) ; silencio
  (if banda-on
    (begin
      (set! banda-on #f)
      (stop-sound som-bandasonora)))
  (if (keypressed? 'M) ; banda sonora
    (if (not banda-on)
      (begin
        (set! banda-on #t)
        (play-sound-looped som-bandasonora))))
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Claro que é necessário acrescentar as duas variáveis utilizadas, no bloco 1.1.

```
;sons
(som-raquete #f)
(som-parede #f)
(som-bandasonora #f)
(banda-on #f)
```

Em seguida, carregar este som através do procedimento `load-sound`, que deverá ser colocado no procedimento `inicializacao`.

```
;sons
(set! som-raquete (load-sound (getPathCompleto "raquete.wav")))
(set! som-parede (load-sound (getPathCompleto "parede.wav")))
(set! som-bandasonora (load-sound (getPathCompleto "bandasonora.wav")))
```



Experimente o código desenvolvido para ter o efeito de uma banda sonora.

O ecrã abre com

> (jogo-pong 800 600 2)

e o programa termina premindo a tecla `Esc`.

Não se esqueça que às teclas `T` (teclado) e `R` (rato) `M` (música) e `S` (silêncio) estão associados tipos diferentes de funcionamento do jogo.

Acrescente a possibilidade de a banda sonora ser distinta quando o jogador 1 está a ganhar (mais alegre) ou quando está a perder (mais triste). Considere que o utilizador está a ganhar quando a variável `golos1` tem valor superior à variável `golos2`.

Obs.: Terá que acrescentar outro ficheiro musical... Utilize o ficheiro `mar.wav`, colocado em `user-feup`, quando jogador 1 estiver a "meter água"...

### O computador também sabe jogar!

Até aqui, não deverá ter tido dificuldade em derrotar o jogador adversário... porque ele não existe!

Nos jogos de computador que requeiram mais do que um jogador, é necessário arranjar uma forma de controlar o segundo jogador (ou restantes). Uma possibilidade é permitir que mais do que um jogador possa utilizar o teclado ou o rato (ou outro periférico) no mesmo computador, ou em computadores distintos, partilhando uma ligação em rede.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Outra forma é "dar inteligência" ao computador para que possa tomar o lugar dos outros jogadores. Esta "inteligência" pode ir de algo extremamente simples, como será o caso deste jogo, até algoritmos extremamente complexos, como no caso de jogos de estratégia, que podem cair no âmbito de áreas científicas como a Inteligência Artificial.

Neste jogo do Pong, optaremos por dar alguma "inteligência" ao computador, que fará de jogador 2. A estratégia será simples: fazer deslocar a raquete 2 na vertical enquanto a coordenada y da bola for diferente da coordenada y da raquete. A raquete só se deslocará quando a bola for na sua direcção e, conforme a bola estiver acima ou abaixo da raquete, o sentido do deslocamento será distinto. O seguinte código, a colocar no bloco 2.2., executa esta funcionalidade:

```
; jogada do computador
(if (> x-vel-bola 0)      ; a bola desloca-se na direccao da raquete 2?
    (if (> y-bola y-raquete2)
        (set! y-raquete2 (inteiro (+ y-raquete2
                                         (min vel-raquete2 (- y-bola y-raquete2)))))

        (set! y-raquete2 (inteiro (- y-raquete2
                                         (min vel-raquete2 (- y-raquete2 y-bola)))))))
```

É também necessário definir, no bloco 1.1., a velocidade de deslocamento da raquete do jogador 2 (computador), que depende do nível de dificuldade do jogo.

```
; velocidade do jogador 2 (pixel/ciclo) - depende do nivel
(vel-raquete2 (* nivel 2))
```



**Antes de passar ao teste desta nova versão do jogo, analise o código associado à jogada do computador.**

**Tente depois explicar o que aconteceria se, em vez de,**

**(min vel-raquete2 (- y-bola y-raquete2))**

**fosse apenas**

**(- y-bola y-raquete2)**

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Experimente esta nova versão do jogo em que, finalmente, vai contar com a oposição do jogador 2, o computador.

O ecrã abre com

```
> (jogo-pong 800 600 2)
```

e o programa termina premindo a tecla Esc.

Não se esqueça que às teclas T (teclado) e R (rato) M (música) e S (silêncio) estão associados tipos diferentes de funcionamento do jogo.

Já deve ter notado que o computador é um adversário muito forte, bastante eficiente e preciso a jogar, pois tem acesso às variáveis do jogo. Isto nem sempre é uma vantagem do jogo, pois desmoraliza o jogador humano que, se vir que será quase impossível ganhar, desistirá na maioria das vezes. Assim, é comum diminuir-se a precisão da jogada do computador, através da limitação do acesso a estes dados ou da introdução de "ruído", incorporando erros aleatórios. No nosso caso, poderíamos tentar limitar a resposta do computador a um "campo de visão" mais pequeno. Ou seja, o computador só poderia mexer a sua raquete quanto a bola estivesse a uma distância inferior a um determinado valor (por exemplo, metade do campo de jogo).

Diminua a precisão do computador utilizando a estratégia de limitação do seu "campo de visão". Depois experimente o jogo e veja se consegue ganhar ao computador...

## Quem vence?

O objectivo final de um jogo de computador, tal como de outro jogo qualquer, é... vencê-lo!

É assim imprescindível mostrar ao jogador qual a sua pontuação ou a progressão no jogo, quando este é jogado por níveis ou etapas. No caso do jogo do Pong, a situação de jogo é retratada pelos golos de cada um dos jogadores. Quem marcou mais golos está a ganhar...

Para mostrar o resultado do jogo colocar-se-á um texto com o número de golos de cada um dos jogadores, especificados nas variáveis golos1 e golos2. Assim, no bloco 3 (saída gráfica) será necessário acrescentar o código:

```
; marcador do jogo
(image-print-center imagem (inteiro (/ x-janela 2)) (inteiro (/ y-janela 5)))
(image-color 255 255 255) -1
(string-append (number->string golos1) " - " (number->string golos2)))
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Experimente esta nova versão do jogo, agora já com a visualização dos golos marcados.

O ecrã abre com

```
> (jogo-pong 800 600 2)
```

e o programa termina premindo a tecla Esc.

Não se esqueça que às teclas T (teclado) e R (rato) M (música) e S (silêncio) estão associados tipos diferentes de funcionamento do jogo. Depois das experiências, introduza as alterações necessárias para que o marcador inclua também o nome dos jogadores.

## O enredo do Jogo

Um jogo é composto por diversas etapas, denominadas estados, traduzindo a evolução do seu enredo. É como o desenrolar de uma história, no qual o jogador, de alguma forma, também consegue interferir. Estas etapas tanto podem corresponder a diversos cenários do jogo, como a níveis de dificuldade distintos. Servem, basicamente, para manter o jogador imerso nesta actividade, renovando o efeito de surpresa, a cada nova etapa do jogo. Para implementar este tipo de estratégia é muito comum recorrer-se ao conceito de máquina de estados.

Uma máquina de estados permite saber, a cada momento, em que estado se desenrola o jogo, e quais as acções ou eventos que provocam a transição entre estados.

Neste jogo do Pong, são considerados apenas 3 estados: o menu inicial, o decorrer do jogo com bola e o menu final do jogo, que indica o vencedor.

Uma vez que o motor do jogo funciona de forma cíclica, é necessário operar esta característica através da implementação de uma máquina de estados. Para se saber o estado actual em cada ciclo do motor de jogo, este é armazenado numa variável, estado, que deverá ser adicionada ao bloco 1.1, juntamente com a definição do número de golos que faz terminar um jogo, a variável max-golos:

```
; maquina de estados
(estado 0)
; numero de golos que termina o jogo
(max-golos 5)
```



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

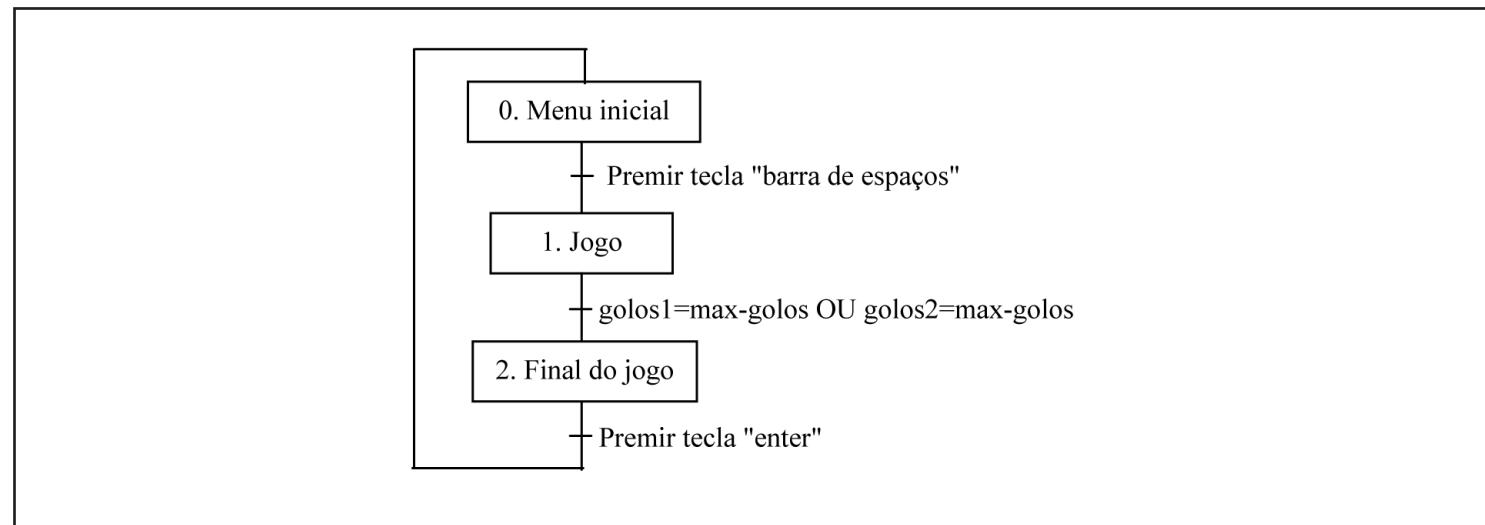
6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

O diagrama apresenta a máquina de estados que iremos implementar para este jogo.



Em cada uma das fases de um ciclo de jogo (Fases: Entrada, Processamento e Saída), uma estrutura de decisão com base na variável `estado` (por exemplo, uma estrutura do tipo `case`) selecciona o código a executar:

```
(case estado ; maquina de estados  
  
; menu inicial  
((0)  
; ...  
; jogo  
((1)  
; ...  
;final do jogo  
((2)  
; ...  
)
```

Assim, será necessário introduzir esta estrutura de controlo em todos os blocos do ciclo de jogo: 2.1. Entrada; 2.2. Processamento e 3. Saída gráfica.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



No essencial, o código da versão mais recente será incluído no estado 1-jogo, mas terá que ser definido o código para os restantes dois estados (estado 0 - menu inicial; estado 2 - final do jogo), tal como é indicado no extracto de código que se segue.

Também será necessário definir a mudança de estado, tal como indicado no diagrama anterior. Como a maior parte das mudanças de estado se processa por teclas, a alteração de estado será fundamentalmente realizada no bloco 2.1.



**Preste agora muita atenção ao extracto de código que se segue...**

```
(require (planet "util.ss" ("kazzmir" "allegro.plt" 1 0)))
(require (prefix image- (planet "image.ss" ("kazzmir" "allegro.plt" 1 0))))
(require (planet "keyboard.ss" ("kazzmir" "allegro.plt" 1 0)))
(require (prefix mouse- (planet "mouse.ss" ("kazzmir" "allegro.plt" 1 0))))
(require (planet "sound.ss" ("kazzmir" "allegro.plt" 1 0)))

;;;;;;;;;;
; função principal do jogo Pong ;
;;;;;;;;;
(define jogo-pong
  (lambda (x-janela y-janela nivel)

    ;;;;;;;;;;
    ; 1. Definições ;
    ;;;;;;;;;;
    (letrec (
      ;;;;;;;;;;
      ; 1.1. Variaveis do jogo ;
      ;;;;;;;;;;
      (pi (acos -1)) ; valor de Pi
      ...
      ...)))
```



Aqui, no bloco 1.1, foram acrescentadas 2 variáveis, estando o código indicado a negrito.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
; maquina de estados
(estado 0)
; numero de golos que termina o jogo
(max-golos 5)

;;;;;;;;;;
; 1.2. Procedimentos auxiliares ;
;;;;;;;;;;
; retorna inteiro exacto
...
...

;inicializacao do Allegro
; (easy-init width height depth mode)
;
(easy-init x-janela y-janela 16 'WINDOWED) ; ou 'FULLSCREEN

; inicializacao do jogo
(inicializacao)

; O motor de jogo
; (game-loop logic-proc draw-proc game-delay)
;
(game-loop
;;;;;;;;;;
; 2. Logica do jogo ;
;;;;;;;;;;
(lambda ()
  (if (keypressed? 'ESC)
      #t ; devolvendo #t, termina...
  (begin
   ;;;;;;;;;;
    ; 2.1. Entrada ;
   ;;;;;;;;;;;
```





Aqui, no bloco 2.1, na fase Entrada, foi acrescentado o código que contempla os 3 estados previstos, o código indicado a negrito.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 **2** 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(case estado ; maquina de estados
  ; menu inicial
  ((0)
    (if (keypressed? 'SPACE)
        (begin
          (inicializacao)
          (lanca-bola)
          (set! estado 1)))))

  ; jogo
  ((1)
    ; identifica o final do jogo
    (if (or (>= golos1 max-golos) (>= golos2 max-golos))
        (begin
          (set! estado 2)))

    (if (equal? controlo 'rato)
        ; rato
        ...

        ...
        ; banda sonora
        (if (keypressed? 'S) ; silencio
            (if banda-on
                (begin
                  (set! banda-on #f)
                  (stop-sound som-bandasonora))))
            (if (keypressed? 'M) ; banda sonora
                (if (not banda-on)
                    (begin
                      (set! banda-on #t)
                      (play-sound-looped som-bandasonora)))))))
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
;final do jogo
((2)
  (if (keypressed? 'ENTER)
    (begin
      (set! estado 0)))))

; 2.2. Processamento do jogo ;
;
```



Aqui, no bloco 2.2, na fase Processamento, apenas foi necessário restringir o código existente ao estado 1.  
Indique uma justificação para não ser necessário considerar os estados 0 e 2.

```
(if (= estado 1) ; apenas no estado 1 (jogo)
  (begin
    ; movimentacao da bola - actualizacao
    ...
    ; confrontacao com as paredes
    ...
    ; confrontacao com as raquetes
    ...
    ; jogada do computador
    ...
    ; actualiza deslocamento do rato
    (mouse-get-mickeys()))

    ; devolvendo #f, repete novamente o ciclo
    #f))

; 3. Saida grafica ;
;(lambda (imagem)
```



Finalmente, no bloco 3, na fase Saída, foi acrescentado o código que contempla os 3 estados previstos, o código indicado a negrito.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(case estado ; maquina de estados
  ((0) ; inicio
   (image-print-center imagem (inteiro (/ x-janela 2)) (inteiro (* y-janela 4/10))
   (image-color 255 255 255) -1 "PONG")
   (image-print-center imagem (inteiro (/ x-janela 2)) (inteiro (* y-janela 6/10))
   (image-color 255 0 0) -1
   "Prima <barra de espaços> para comeclar o jogo..."))

  ((1) ; jogo
   ; desenha bola
   (image-draw imagem sprite-bola
   (- (inteiro x-bola) raio-bola) (- (inteiro y-bola) raio-bola))

   ; marcador do jogo
   (image-print-center imagem (inteiro (/ x-janela 2)) (inteiro (/ y-janela 5)))
   (image-color 255 255 255) -1
   (string-append (number->string golos1)
   " - " (number->string golos2)))

   ; raquete do jogador 1
   (image-rectangle-fill imagem
   (- x-raquetel met-larg-raquete)
   (- y-raquetel met-comp-raquete)
   (+ x-raquetel met-larg-raquete)
   (+ y-raquetel met-comp-raquete)
   cor-raquetel)

   ; raquete do jogador 2
   (image-rectangle-fill imagem
   (- x-raquete2 met-larg-raquete)
   (- y-raquete2 met-comp-raquete)
   (+ x-raquete2 met-larg-raquete)
   (+ y-raquete2 met-comp-raquete)
   cor-raquete2)
  ) ; fecho do estado 1, também acrescentado...
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
((2) ; fim do jogo
  (if (> golos1 golos2)
    (image-print-center imagem (inteiro (/ x-janela 2)) (inteiro (/ y-janela 2))
    (image-color 255 0 0) -1
    "O jogador 1 é o vencedor. Parabéns! <prima enter>")
    (image-print-center imagem (inteiro (/ x-janela 2)) (inteiro (/ y-janela 2))
    (image-color 255 0 0) -1
    "O jogador 2 é o vencedor. Parabéns! <prima enter>"))))

; determina temporizacao do motor de jogo com base na velocidade desejada
(frames-per-second 30))

; fecha janela Allegro e termina programa
(easy-exit)
(display "Obrigado por ter jogado o Pong."))
```



Experimente a versão final do jogo.

O ecrã abre com

> (jogo-pong 800 600 2)

e o programa termina premindo a tecla Esc.

Não se esqueça que às teclas T (teclado) e R (rato) M (música) e S (silêncio) estão associados tipos diferentes de funcionamento do jogo.

Depois de experimentar esta versão do jogo, acrescente um novo estado "pausa" que interrompa o seguimento do jogo sempre que for premida a tecla H (ou h). O jogo deverá ser retomado com a tecla enter.

# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

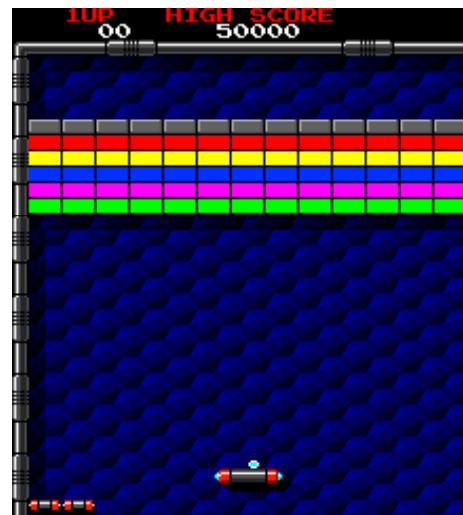


### Projecto final

Crie um novo jogo com base no jogo que agora mesmo concluiu, em que no lugar do jogador adversário existe uma parede de tijolos. O objectivo do jogo é destruir todos os tijolos que formam essa parede, destruindo-os com a bola. Cada vez que um tijolo é atingido pela bola, desaparece, rebatendo a bola de volta.

O jogo termina quando todos os tijolos forem destruídos ou quando a bola sair do tabuleiro (pelo fundo), um número de vezes predeterminado.

Este jogo baseia-se no clássico [Arkanoid](#).



Desenvolva o jogo proposto.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



## Módulo 6.3 - Testes unitários

Testar o código que se produz, de uma forma sistemática, é uma actividade que nem sempre tem merecido a atenção que lhe é devida. No decurso dos nossos trabalhos, temos recomendado que, ao analisar um problema, deve ser preparado, manualmente, um conjunto de entradas de teste que ajudarão a melhor entender esse problema, servindo também, mais tarde, para testar o código desenvolvido. Muitas vezes, por comodismo, nem se dispensa um pouco de tempo para experimentar todas as entradas de teste anteriormente preparadas. Um dia, se esse código sofre alguma alteração, aquele conjunto de entradas de teste ou já foi esquecido ou a paciência não é suficiente para o aproveitar.

Os testes unitários são programas, normalmente preparados antes de ser desenvolvido o código que irão testar, que deverão ser guardados num ficheiro cujo nome mostre bem que é o teste unitário desse código. E então, quando o código acaba de ser desenvolvido pela primeira vez, ou após qualquer actualização, grande ou pequena, bastará executar o teste unitário que lhe está associado. Da execução do teste unitário resulta informação que permite localizar rapidamente os pontos do código em teste que exijam intervenção...

O Scheme permite o desenvolvimento de testes unitários através da biblioteca SchemeUnit.plt, parte integrante da distribuição PLT Scheme, documentada em [\*\*SchemeUnit: Unit Testing in Scheme\*\*](#).

Neste módulo é feita uma abordagem à utilização da biblioteca SchemeUnit.plt, ilustrada com vários exemplos. Haverá também espaço quer para uma breve incursão ao código que prevê o lançamento de excepções quer para os testes unitários respectivos.

### Palavras-Chave

Testes unitários, excepções, processador (*handler*) de excepções, *thunk* (procedimento sem parâmetros).

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Preparar um teste unitário

Como já deve ter entendido, um teste unitário é um programa que reúne um conjunto de casos de teste de um dado código, preparado para ser usado de uma forma muito simples. Se os casos de teste forem bem seleccionados, certamente que da execução do teste unitário resultará informação suficiente para localizar rapidamente os pontos do código que exijam intervenção.

O exemplo que nos vai servir para definir e desenvolver o nosso primeiro teste unitário é o cálculo do factorial de um inteiro não negativo. A resolução deste problema deve passar por um procedimento com o nome `factorial`. Como entradas de teste, poderá considerar as seguintes:

```
> (factorial 0)  
1  
> (factorial 1)  
1  
> (factorial 2)  
2  
> (factorial 7)  
5040
```

Imagine que o procedimento `factorial`, depois de desenvolvido, será guardado no ficheiro `factorial.scm`.

Com esta especificação pode já preparar um teste unitário, mesmo sem ter ainda desenvolvido o procedimento `factorial`. Esse teste unitário, para ficar claramente associado a `factorial.scm`, vai ser armazenado no ficheiro `factorial-test-1.scm`.



**Aponte alguma razão para o cuidado tido na escolha do nome do ficheiro onde é colocado o teste unitário.**

# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Surge agora o primeiro teste unitário deste Módulo, que deve analisar cuidadosamente, sem passar por cima dos comentários incluídos.

```
; importação da biblioteca SchemeUnit
; na linha seguinte: ... 2 (= 9))) significa importação da versão 2.9
(require (planet "test.ss" ("schematics" "schemeunit.plt" 2(= 9)))

; importação da interface textual da biblioteca SchemeUnit
(require (planet "text-ui.ss" ("schematics" "schemeunit.plt")))

; carregar o código a testar, neste caso, colocado no ficheiro factorial.scm
(load "factorial.scm")

; definição dos testes a realizar
(define testes-a-realizar
  (test-suite
    "Testar factorial.scm"
    (test-case
      "Testar procedimento factorial"
      (check-equal? (factorial 0) 1 "factorial 0")
      (check-equal? (factorial 1) 1 "factorial 1")
      (check-equal? (factorial 2) 2 "factorial 2")
      (check-equal? (factorial 7) 5040 "factorial 7"))))

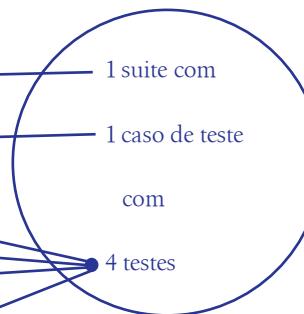
; execução dos testes definidos
(test/text-ui testes-a-realizar)
```



Pela análise do teste unitário, procure identificar:

- 1- quais são os parâmetros de check-equal?.
- 2- e também os parâmetros de test/text-ui.

Em caso de dificuldades, consulte o documento [SchemeUnit: Unit Testing in Scheme](#)



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



O desenvolvimento do procedimento pretendido é muito simples e vai ser guardado, como havia sido combinado, em factorial.scm.

```
; Procedimento recursivo com um parâmetro, num.  
; Devolve o factorial de num.  
;  
(define factorial  
  (lambda (num)  
    (if (zero? num)  
        1 ; o caso base  
        (* num (factorial (sub1 num)))))) ; o caso geral
```

Agora, em vez de correr os 4 testes, manualmente, um de cada vez, bastará fazer executar o teste unitário armazenado em factorial-test-1.scm, como se se tratasse de um programa Scheme. Desta operação resultará:

```
1 success(es) 0 failure(s) 0 error(s) 1 test(s) run  
0
```



**Esperaria provavelmentever uma mensagem indicando 4 sucessos!  
Tente uma justificação para o facto de ter surgido apenas 1.**



Experimente o teste unitário factorial-test-1.scm.  
Aproveite para juntar ou alterar alguns testes.



**Os 4 testes foram agrupados num único caso de teste.  
O que aconteceria se um dos 4 testes detectasse uma falha?**

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

A ideia agora é introduzir dois testes incorrectos no mesmo caso de teste, como se pode verificar no extracto do teste unitário armazenado em factorial-test-2.scm.

```
...
(define testes-a-realizar
  (test-suite
    "Testar factorial.scm"
    (test-case
      "Testar procedimento factorial"
      (check-equal? (factorial 0) 1 "factorial 0")
      (check-equal? (factorial 1) 1 "factorial 1")
      ; o teste que se segue é intencionalmente incorrecto...
      (check-equal? (factorial 5) 100 "factorial 5")
      ; o teste que se segue é intencionalmente incorrecto...
      (check-equal? (factorial 6) 110 "factorial 6")
      (check-equal? (factorial 2) 2 "factorial 2")
      (check-equal? (factorial 7) 5040 "factorial 7"))))
```

Apesar dos 6 testes realizados, 4 correctos e 2 intencionalmente com falha, da execução deste novo teste unitário vão resultar 0 casos de sucesso e 1 falha.

```
Testar factorial.scm > Testar procedimento factorial
Testar procedimento factorial has a FAILURE
name: check-equal?
location: #<struct:object:...pper/stepper-tool.ss:618:8>:20:4
message: "factorial 5"
actual: 120
expected: 100
0 success(es) 1 failure(s) 0 error(s) 1 test(s) run
1
>
```



**Analise a informação fornecida.**

**Fica a saber que houve, pelo menos, uma falha no teste "factorial 5", que o valor actual devolvido por factorial de 5 foi 120 e que o valor que o teste unitário esperava era 100!**

**O que terá acontecido aos 4 testes correctos e também ao teste incorrecto, este relativo ao factorial de 6?**



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Experimente o teste unitário factorial-test-2.scm.  
Aproveite para juntar ou alterar alguns testes.



**Se, em factorial.scm, fizer o caso base a devolver 2, em vez de 1, e, seguidamente, executar o teste unitário factorial-test-2.scm, que mensagem pensa receber?**

### Testes unitários quando o código envolve vários procedimentos

O problema que agora lhe colocam continua a envolver o cálculo do factorial, mas é-lhe exigida uma solução que gere processos iterativos e não processos recursivos, como acontecia com factorial.scm.

A solução desenvolvida baseia-se em dois procedimentos, como vai poder ver, e será armazenado no ficheiro factorial-iter.scm.

O procedimento factorial-novo tem dois parâmetros e gera processos iterativos.

```
; cálculo do factorial - processo iterativo
(define factorial-novo
  (lambda (n acumulador)
    (if (zero? n)
        acumulador ; caso base, em que n=0...
        ;
        (factorial-novo (sub1 n) (* n acumulador)))))) ; caso geral

> (factorial-novo 0 1)
1
> (factorial-novo 4 1)
24
```





O segundo argumento em **factorial-novo** é sempre 1.  
O que é que se pretenderá com o procedimento **factorial-iter** que vem a seguir?

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



```
(define factorial-iter
  (lambda (num)
    (factorial-novo num 1)))

> (factorial-iter 0)
1
> (factorial-iter 4)
24
```

O segundo procedimento, **factorial-iter**, serve apenas de interface ao procedimento **factorial-novo**, escondendo o segundo parâmetro.

Para esta situação, a unidade de teste terá dois casos de teste, um para cada procedimento, e vai ser armazenado no ficheiro **factorial-iter-test-1.scm**.

```
; importação da biblioteca SchemeUnit
; na linha seguinte: ... 2 (= 9))) significa importação da versão 2.9
(require (planet "test.ss" ("schematics" "schemeunit.plt" 2 (= 9))))
; importação da interface textual
(require (planet "text-ui.ss" ("schematics" "schemeunit.plt" 2 (= 9)))

; código a testar
(load "factorial-iter.scm")

; definição dos testes a realizar
(define testes-a-realizar
  (test-suite
    "Testar factorial-iter.scm"

    (test-case
      "Testar procedimento factorial-novo"
      (check-equal? (factorial-novo 0 1) 1 "factorial-novo 0 1")
      (check-equal? (factorial-novo 4 1) 24 "factorial-novo 4 1"))))
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
(test-case
  "Testar procedimento factorial-iter"
  (check-equal? (factorial-iter 0) 1 "factorial-iter 0")
  (check-equal? (factorial-iter 4) 24 "factorial-iter 4")))

; execução dos testes definidos
(test/text-ui testes-a-realizar)
```

Da execução deste teste unitário, resultou:

```
2 success(es) 0 failure(s) 0 error(s) 2 test(s) run
0
```



Experimente o teste unitário factorial-iter-test-1.scm.  
Aproveite para juntar ou alterar alguns testes.

### Como testar com valores numéricos aproximados?

O problema que vai servir de exemplo para esta nova situação é mesmo muito simples. Trata-se do cálculo da área de um círculo, dado o respectivo raio, para o qual a solução é quase imediata.

```
; definição de pi, pois algumas implementações do Scheme não prevêem esta constante
(define pi (acos -1))

; recebe o raio de um circulo
; devolve a respectiva area
;
(define area-circ
  (lambda (r)           ; area do círculo = pi * r * r
    (* pi r r)))
```

Este código vai ser armazenado no ficheiro area-circulo.scm e da sua execução resulta, por exemplo, o seguinte:

```
> (area-circ 1)
3.141592653589793
> (area-circ 0.0)
0.0
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (area-cir 0.1)
0.031415926535897934
>
```

Na unidade de teste que se segue, armazenada em `area-circulo-test-1.scm`, são usados dois casos de teste: um, erradamente, usa `check-equal?`, que não é adequado para trabalhar com valores aproximados; o outro caso de teste usa `check-=`, que admite mais um parâmetro que funciona como tolerância nas comparações.

```
; importação da biblioteca SchemeUnit
; na linha seguinte: ... 2 (= 9))) significa importação da versão 2.9
(require (planet "test.ss" ("schematics" "schemeunit.plt" 2 (= 9))))
; importação da interface textual
(require (planet "text-ui.ss" ("schematics" "schemeunit.plt" 2 (= 9)))

; código a testar
(load "area-circulo.scm")

; definição dos testes a realizar
(define testes-a-realizar
  (test-suite
    "Testar Area-cir.scm"
    (test-case
      "com tolerância 0.001"
      (check-= (area-cir 0.0) 0.0 0.001 "area-cir 0.0")
      (check-= (area-cir 0.1) 0.03141592 0.001 "area-cir 0.1")
      (check-= (area-cir 1.0) 3.14159265 0.001 "area-cir 1.0"))
    ;
    (test-case
      "sem tolerância"
      (check-equal? (area-cir 0.0) 0.0 "area-cir 0.0")
      (check-equal? (area-cir 0.1) 0.03141592 "area-cir 0.1")
      (check-equal? (area-cir 1.0) 3.14159265 "area-cir 1.0")))
  )
  ; execução dos testes definidos
  (test/text-ui testes-a-realizar)
```



Analise a unidade de teste e tente perceber a sintaxe de `check-=`.  
Consegue prever o que resultará da execução desta unidade de teste?



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Da execução deste teste unitário resultará:

```
Testar Area-cir.scm > sem tolerância
sem tolerância has a FAILURE
name: check-equal?
location: #<struct:object:...pper/stepper-tool.ss:618:8>:23:6
message: "area-cir 0.1"
actual: 0.031415926535897934
expected: 0.03141592
1 success(es) 1 failure(s) 0 error(s) 2 test(s) run
1
>
```



Entre `check-equal?` ou `check-=` qual escolherá para testes com valores numéricos aproximados? Justifique.



Experimente o teste unitário `area-circulo-test-1.scm`. Aproveite para corrigir o teste unitário, eliminando os testes inadequados.

### Testes que envolvem dados compostos

O exemplo que nos vai servir para ilustrar esta situação relaciona-se com o desenvolvimento em Scheme do procedimento `remove-primeiros-n`, com um primeiro parâmetro que é uma lista e um segundo que é um inteiro positivo,  $n$ , e que devolve uma sublist da lista dada, saltando os seus primeiros  $n$  elementos.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

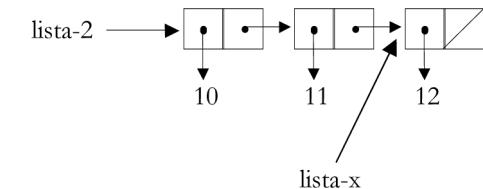
R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (define lista-2 (list 10 11 12))
> lista-2
(10 11 12)
> (define lista-x (remove-primeiros-n lista-2 2))
> lista-x
(12)
> lista-2
(10 11 12)

; remove os primeiros n elementos de uma lista
;
(define remove-primeiros-n
  (lambda(lis n)
    (if (zero? n)
        lis
        (remove-primeiros-n (cdr lis)
                             (sub1 n))))))
```



Com o desenvolvimento deste procedimento, até parece que esquecemos que o Scheme disponibiliza `list-tail`, um procedimento com uma funcionalidade semelhante a `remove-primeiros-n`.

```
> (list-tail lista-2 2)
(12)
```

Um ponto importante é o facto de `remove-primeiros-n` não funcionar como construtor, ou seja, não devolve uma lista nova, mas sim uma parte, uma sublista da lista original, conforme foi pedido.

Tome atenção à unidade de teste e, em particular, aos comentários associados aos casos de teste.

```
; importação da biblioteca SchemeUnit
; na linha seguinte: ... 2 (= 9))) significa importação da versão 2.9
(require (planet "test.ss" ("schematics" "schemeunit.plt" 2 (= 9))))
; importação da interface textual
(require (planet "text-ui.ss" ("schematics" "schemeunit.plt" 2 (= 9))))
; código a testar
(load "remove-primeiros.scm")
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
; definição dos testes a realizar
(define testes-a-realizar
  (test-suite
    "Testar remove-primeiros-n"

    (test-case
      "1- não interessa"
      (let ((lis '(1 2 3 4 5)))
        (check-equal? (remove-primeiros-n lis 2) '(3 4 5) "1- com equal?"))
      ;
      ; inadequado!
      ; pois 2 listas distintas podem ter o mesmo conteúdo!

    (test-case
      "2- não interessa"
      (let ((lis '(1 2 3 4 5)))
        (check-equal? (remove-primeiros-n lis 2) (cddr lis) "2- com equal?"))
      ;
      ; inadequado!
      ; pois 2 listas distintas podem ter o mesmo conteúdo!

    (test-case
      "3- não interessa"
      (let ((lis '(1 2 3 4 5)))
        (check-eqv? (remove-primeiros-n lis 2) '(3 4 5) "3- com eqv?"))
      ;
      ; inadequado!
      ; daqui resultará sempre #f

    (test-case
      "4- interessa"
      (let ((lis '(1 2 3 4 5)))
        (check-eqv? (remove-primeiros-n lis 2) (cddr lis) "4- com eqv?")))
      ;
      ; adequado...
      ; se #t, trata-se da mesma lista
      ; execução dos testes definidos

    (test/text-ui testes-a-realizar)
```



Tendo analisado os comentários indicados neste teste unitário, não avance sem ter a certeza que entendeu bem o respectivo significado.  
Provavelmente terá que relembrar a diferença entre os predicados `equal?` e `eqv?`...  
Uma ajuda: `equal?` compara conteúdos e `eqv?` compara apontadores.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Deste teste unitário resultará:

```
Testar remove-primeiros-n > 3- não interessa
3- não interessa has a FAILURE
name: check-eqv?
location: #<struct:object:...pper/stepper-tool.ss:618:8>:28:6
message: "3- com eqv?"
actual: (3 4 5)
expected: (3 4 5)
3 success(es) 1 failure(s) 0 error(s) 4 test(s) run
1
>
```



Como explica a falha detectada, sendo iguais os valores de **actual** e de **expected!**?



Experimente o teste unitário `remove-primeiros-test.scm`.  
Aproveite para corrigir o teste unitário, eliminando os testes inadequados.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Que casos seleccionar para o teste unitário?

Não se falou, até aqui, nos casos a seleccionar para garantir que o teste unitário não deixa escapar nenhuma eventual falha. Normalmente não se seleccionam muitos casos, mas apenas aqueles que são verdadeiramente necessários.

De uma forma geral, o teste unitário deve incluir:

- os casos de base ou de terminação, se o procedimento a testar for recursivo
- as condições fronteira ou limite
- as entradas não usadas, se o procedimento estiver preparado para tratar esse tipo de situação



Um exemplo é o procedimento **factorial** que, na versão apresentada, não está preparado para receber entradas negativas ou entradas não inteiras.

Não tente, por isso, chamar o procedimento **factorial** com entradas desta natureza, pois arrisca-se a bloquear o DrScheme...

Vamos deixar este assunto para o final deste módulo, pois isto leva-nos até ao processamento de excepções que, na nossa abordagem ao Scheme, nunca chegou a ser tratado.

- um caso geral, fora das situações anteriores.

O exemplo que vamos aproveitar para ilustrar este tema é o procedimento **junta-duas-listas** que aceita duas listas como argumentos e devolve uma lista com uma cópia de todos os elementos daquelas listas. Trata-se de um caso particular do procedimento **append**, já que este procedimento primitivo do Scheme aceita um número não fixo de listas.

# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
> (define lista-1 (list 1 2 3 4 5 6))
> lista-1
(1 2 3 4 5 6)
> (define lista-2 (list 10 11 12))
> lista-2
(10 11 12)
> (junta-duas-listas lista-1 lista-2)
(1 2 3 4 5 6 10 11 12)

> (append lista-1 lista-2)
(1 2 3 4 5 6 10 11 12)
> (append lista-1 lista-2 lista-1)
(1 2 3 4 5 6 10 11 12 1 2 3 4 5 6)

> (junta-duas-listas lista-1 (junta-duas-listas lista-2 lista-1))
(1 2 3 4 5 6 10 11 12 1 2 3 4 5 6)
> lista-2
(10 11 12)
> lista-1
(1 2 3 4 5 6)

(define junta-duas-listas
  (lambda (lis-1 lis-2)
    (if (null? lis-1)      ; caso base, se lis-1 é lista vazia,
        lis-2              ; então é devolvida lis-2
        (cons (car lis-1)    ; caso geral: juntar o primeiro elemento de
              (junta-duas-listas (cdr lis-1) ; lis-1 ao resultado de juntar
                                (cdr lis-2)))))) ; (cdr lis-1) a lis-2
```

Este código vai ser armazenado em junta-2-listas.scm.



Para cobrir adequadamente este procedimento, que casos seleccionaria?  
Justifique face ao que aqui já foi exposto sobre este assunto.

O teste unitário que se segue vai ser armazenado em junta-2-listas-test.scm. Analise cuidadosamente os vários casos de teste.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



```
; importação da biblioteca SchemeUnit
; na linha seguinte: ... 2 (= 9))) significa importação da versão 2.9
(require (planet "test.ss" ("schematics" "schemeunit.plt" 2 (= 9))))
; importação da interface textual da biblioteca SchemeUnit
(require (planet "text-ui.ss" ("schematics" "schemeunit.plt" 2 (= 9)))

; carregar o código a testar
(load "junta-2-listas.scm")
; listas a utilizar nos testes
(define l-1 '(1 2 3 4))
(define l-2 '(10 11 12))
(define l-v '())

; definição dos testes a realizar
(define testes-a-realizar
  (test-suite
    "Testar junta-2-listas.scm"

    (test-case
      "caso: l-vazia l-vazia"
      (check-equal? (junta-duas-listas l-v l-v) l-v "l-v l-v" ))

    (test-case
      "caso: l-vazia l-nao-vazia"
      (check-equal? (junta-duas-listas l-v l-1) l-1 "l-v l-1" ))

    (test-case
      "caso: l-nao-vazia l-vazia"
      (check-equal? (junta-duas-listas l-1 l-v) '(1 2 3 4) "l-1 l-v"))

    (test-case
      "caso: l-nao-vazia l-nao-vazia"
      (check-equal? (junta-duas-listas l-1 l-2) '(1 2 3 4 10 11 12) "l-1 l-2")))

; execução dos testes definidos
(test/text-ui testes-a-realizar)
```



Neste teste unitário, é utilizado `check-equal?` e não `check-eqv?`.  
Avance com uma justificação para este facto.

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Para o procedimento `junta-duas-listas`, nada foi imposto sobre a lista resultante, contrariamente ao que aconteceu no exemplo anterior, em que o resultado deveria ser uma sublist... Optou-se, portanto, por um teste menos restritivo: `check-equal?`.

Da execução deste teste unitário resulta:

```
4 success(es) 0 failure(s) 0 error(s) 4 test(s) run
0
```



Experimente o teste unitário `junta-2-listas-test.scm`.

Aproveite para alterar ou juntar mais alguns testes, incluindo também casos que provoquem falhas, para experimentar listas diferentes das já utilizadas e verifique o que resulta da execução do teste unitário assim modificado.

### Exercício 1

Antes de passar ao enunciado deste exercício, convém relembrar a noção de árvore de pesquisa binária.

Uma árvore binária A é de pesquisa binária se for uma árvore vazia ou então se

- todos os elementos da sub-árvore à esquerda de A forem menores que a raiz de A;
- todos os elementos da sub-árvore à direita de A forem maiores que a raiz de A;
- e as sub-árvores à esquerda e à direita forem árvores de pesquisa binária.



Face à definição apresentada, verifique se a árvore representada na figura é uma árvore de pesquisa binária.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

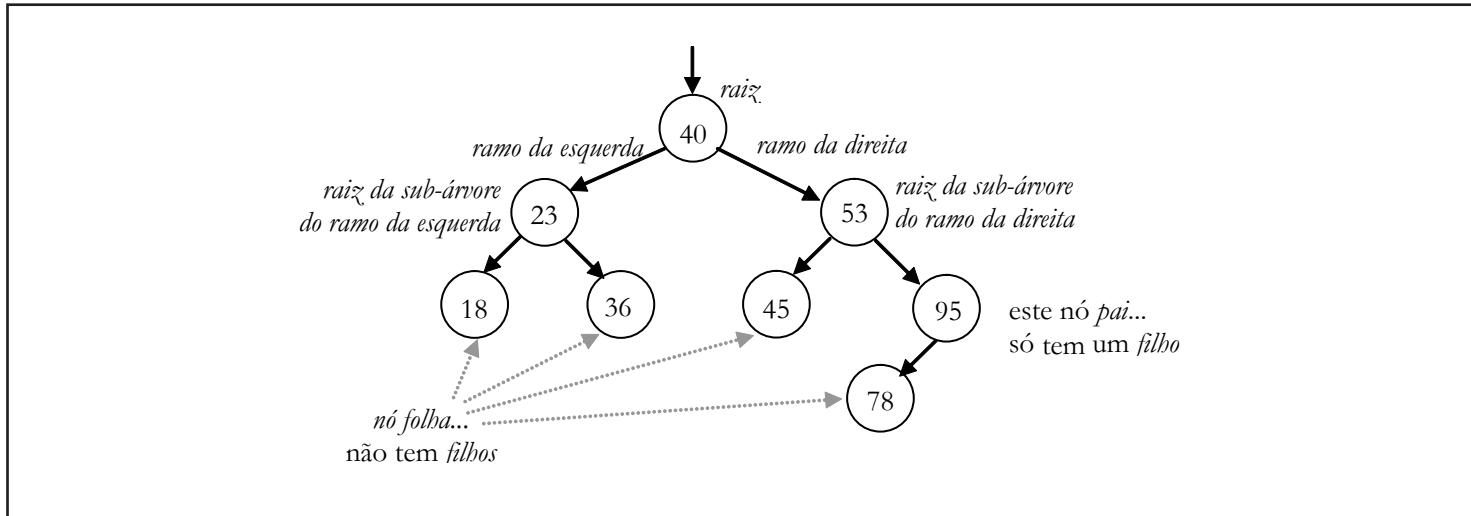
### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

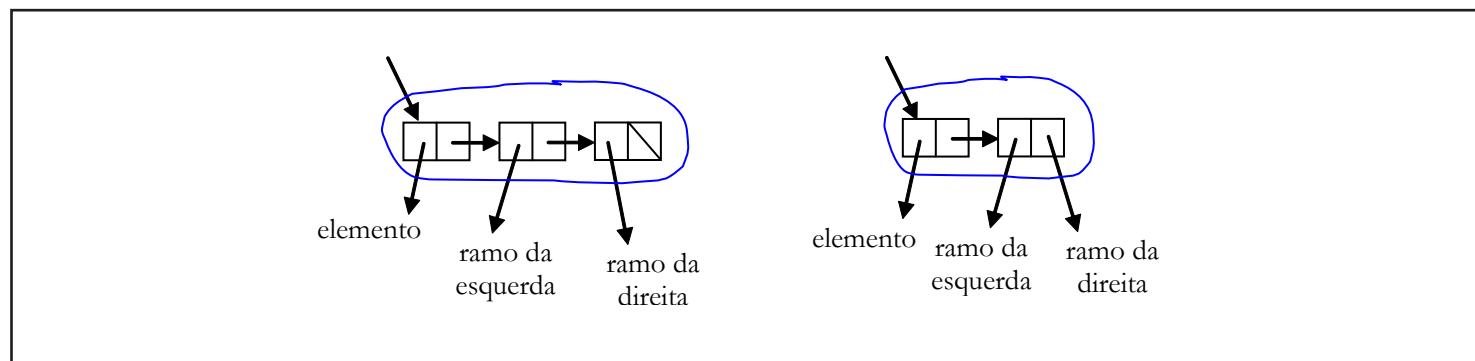
R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Cada um dos nós desta árvore tem 3 elementos: o elemento do nó, o ramo da esquerda e o ramo da direita. A figura mostra duas formas semelhantes para implementar um nó da árvore. Nós optámos pela forma da esquerda.





Tendo em conta a decisão tomada sobre a implementação de um nó, verifique se está correcta a definição de `arv-0`, que pretende ser a representação codificada em Scheme da árvore apresentada na figura.

```
(define arv-0 '(40 (23 (18 () ())  
                     (36 () ()))  
                  (53 (45 () ()))  
                  (95 (78 () ()))  
                  ())))))
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



O código que se segue é uma implementação incompleta de uma abstracção árvore de pesquisa binária.

```
; abstracção árvore de pesquisa binária  
; implementação INCOMPLETA...  
;  
(define faz-arvore  
  (lambda (elem r-esq r-dir)  
    (list elem (cons r-esq r-dir))))  
;  
(define raiz  
  (lambda (arv)  
    (car arv)))  
;  
(define ramo-esq  
  (lambda (arv)  
    (car (cdr arv))))  
;  
(define ramo-dir  
  (lambda (arv)  
    (car (cdr (cdr arv)))))  
;  
(define arv-vazia?  
  (lambda (arv)  
    (null? arv)))  
;  
; se elem for um elemento de arv, devolve o nó onde se encontra elem,  
; caso contrário devolve #f. Trata-se de um selector.  
;(define na-arv?  
;  (lambda (elem arv)  
;    .... .... .... para completar (mais tarde)
```

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



Um algoritmo para o procedimento na-arv?

```
se a árvore for vazia, a chave não está na árvore e devolve #f.  
se não,  
    se a chave for a raiz, encontrou a chave e devolve #t.  
    se não,  
        se a chave for menor que a raiz,  
            procura na sub-árvore do ramo da esquerda.  
        se não,  
            procura na sub-árvore do ramo da direita.
```

```
; se elem não for elemento de arv,  
; põe elem em arv garantido que esta continuará a ser uma árvore de pesquisa binária.  
; Se elem já for elemento de arv, então nada modifica. Em ambos os casos devolve o símbolo ok.  
; Trata-se de um modificador.  
(define poe-na-arv!  
;   (lambda (elem arv)  
;       .... .... para completar (mais tarde)
```



Um algoritmo para o procedimento poe-na-arv!

```
Se a árvore for vazia, põe o elemento na raiz da árvore.  
se não,  
    se a chave for a raiz, não põe o elemento na árvore  
        (para evitar duplicações).  
    se não,  
        se a chave for menor que a raiz,  
            põe o elemento na sub-árvore do ramo da esquerda.  
        se não,  
            põe o elemento na sub-árvore do ramo da direita.  
;  
devolve 'ok.
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

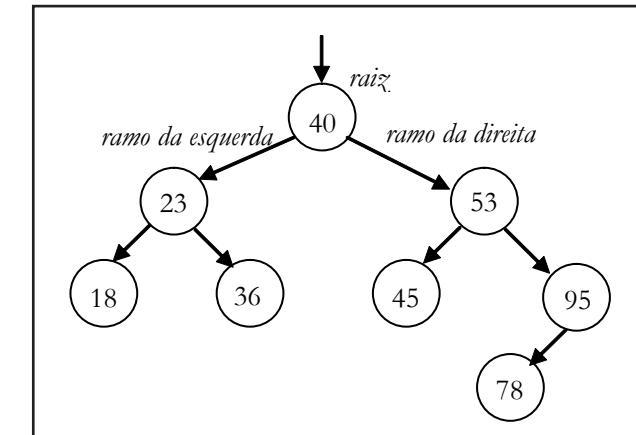
Imagine que lhe é dada a árvore representada na figura e lhe pedem o caminho do elemento 45, tendo como partida o nó raiz.

Podia responder assim:      ramo da direita (pois  $45 > 40$ )  
                                ramo da esquerda (pois  $45 < 53$ ).

Resumindo, o caminho podia ser indicado da seguinte maneira: Dir Esq

Para o 78, seria: Dir Dir Esq

Para o 37, poderia ser: Esq Dir Fim - não encontrado.



```
(define caminho
  (lambda (elem arv)
    (cond ((arv-vazia? arv)
           '(Fim - nao encontrado))
          ((< elem (raiz arv)) ; vai para ramo Esq
           (cons 'Esq
                 (caminho elem (ramo-esq arv))))
          ((> elem (raiz arv)) ; vai para ramo Dir
           (cons 'Dir
                 (caminho elem (ramo-dir arv))))
          (else '()))) ; foi encontrado!
```



Experimente a abstracção árvore de pesquisa binária.

Para o que se segue, não é fundamental ter a implementação completa desta abstracção, mas se tiver alguma disponibilidade, seria interessante desenvolver `na-arv?` e `poe-na-arv!`. Com esse esforço, certamente consolidará muitos dos conhecimentos de programação adquiridos até aqui...



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Finalmente, é-lhe apresentado o enunciado do exercício: defina e desenvolva um teste unitário para o procedimento caminho. Inclua também casos de teste para todos os procedimentos utilizados por aquele procedimento.



Desenvolva e experimente o teste unitário a colocar em arv-bin-test.scm.

### Testes unitários envolvendo a geração de números aleatórios

O problema que ilustra este tipo de testes é um procedimento, designado por faz-roleta que, por sua vez, devolve um outro procedimento. O procedimento devolvido funciona como uma espécie de roleta, pois, quando é chamado, devolve um número inteiro, aleatório, situado numa gama definida pelos limites inferior e superior. Mas o melhor será analisar o código que se segue.

```
; recebe dois inteiros que definem uma gama de inteiros
; devolve um procedimento que gera, por cada chamada,
; um inteiro aleatório, na gama acima indicada.
(define faz-roleta
  (lambda (inf sup)
    ;
    (lambda ()
      (+ (if (< inf sup)
              inf
              sup)
         (random (add1 (abs (- inf sup)))))))))

> (define dado (faz-roleta 1 6)) ; ao procedimento devolvido por faz-roleta
; foi associado ao identificador dado
```



INTRODUÇÃO
1 - O ESSENCIAL DO SCHEME
2 - RECURSIVIDADE
3 - ABSTRACÇÃO DE DADOS
4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE
5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS
6 - SCHEME E OUTRAS TECNOLOGIAS
R 1 2 3 4 5
7-EXERCÍCIOS E PROJECTOS
ANEXOS



O procedimento devolvido por (`faz-roleta 1 6`) foi associado ao identificador `dado`.  
Avance com uma justificação para a escolha deste identificador.

```
> (dado)  
3  
> (dado)  
3  
> (dado)  
5  
> (dado)  
1  
> (dado)  
4  
>
```



A situação nova que agora encontrámos é a necessidade de testar procedimentos que devolvem números de uma forma aleatória.  
No caso do procedimento `dado`, devolverá umas vezes 5, outras 3, outras ainda qualquer inteiro entre 1 e 6.  
A que testes deverão ser sujeitos estes procedimentos para se verificar que estão a funcionar correctamente?  
Tente responder antes de avançar para o próximo teste unitário.



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Tome atenção a este novo teste unitário, pois ele aparece com um check-true que ainda não tinha sido utilizado. A propósito, também existe o check-false.

```
(require (planet "test.ss" ("schematics" "schemeunit.plt" 2 (= 9))))
(require (planet "text-ui.ss" ("schematics" "schemeunit.plt" 2 (= 9))))
(load "faz-roleta.scm")

(define rol-4-6 (faz-roleta 4 6)) ; cria roleta na gama 4 a 6

(define roleta-tests
  (test-suite
    "Testar faz-roleta.scm"

    (test-case
      "cria e testa um caso"
      (let ((lancamento (rol-4-6))) ; faz um lançamento
        ; com os dois testes determina-se
        ; se lançamento é maior ou igual que 4
        (check-true (>= lancamento 4) "caso >= 4")

        ; se lançamento é menor ou igual que 6
        (check-true (<= lancamento 6) "caso <= 6"))))

    ;
    (test/text-ui roleta-tests)
  )
)
```

Em vários ensaios realizados com este teste unitário, sempre resultou:

```
1 success(es) 0 failure(s) 0 error(s) 1 test(s) run
0
```



Experimente o teste unitário `faz-roleta-test-1.scm`.

Como foi referido, após vários ensaios com este teste unitário, nunca foi detectada qualquer falha.

Mas pode garantir que `faz-roleta` funciona mesmo bem?

Também funcionará bem se, por exemplo, o primeiro argumento for maior que o segundo?

E se os argumentos forem iguais?

Escreva um teste unitário com os casos de teste necessários para cobrir todas as situações identificadas e proceda a vários ensaios.

# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Um outro teste unitário desenvolvido em torno de faz-roleta é agora apresentado. Este tem dois casos de teste, num dos quais se introduz um teste com um limite errado.

```
(require (planet "test.ss" ("schematics" "schemeunit.plt" 2 (= 9))))
(require (planet "text-ui.ss" ("schematics" "schemeunit.plt" 2 (= 9)))))

(load "faz-roleta.scm")

(define rol-4-6 (faz-roleta 4 6)) ; gama 4 a 6

(define roleta-tests
  (test-suite
    "Testar faz-roleta.scm"

    (test-case
      "1- teste adequado"
      (let ((lancamento (rol-4-6))) ; faz um lançamento
        (check-true (>= lancamento 4) "caso 1, >= 4")
        (check-true (<= lancamento 6) "caso 1, <= 6")))
    ;
    (test-case
      "2- teste inadequado"
      (let ((lancamento (rol-4-6))) ; faz outro lançamento
        (check-true (>= lancamento 4) "caso 2, >= 4") ; teste correcto
        ;
        (check-true (<= lancamento 5) "caso 2, <= 5"))))) ; teste incorrecto
    ;
    (test/text-ui roleta-tests)
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Num primeiro ensaio deste teste unitário resultou:

```
2 success(es) 0 failure(s) 0 error(s) 2 test(s) run
0
```

Num dos ensaios seguintes resultou:

```
Testar faz-roleta.scm > 2- teste inadequado
2- teste inadequado has a FAILURE
name: check-true
location: #<struct:object:...pper/stepper-tool.ss:618:8>:27:6
params:
#f
message: "caso 2, <= 5"
1 success(es) 1 failure(s) 0 error(s) 2 test(s) run
1
```



**Como justifica que, estando um dos limites errados, por vezes o teste unitário não detecte qualquer falha?.**



Experimente o teste unitário `faz-roleta-test-2.scm`.  
Corrija o teste unitário e, de seguida, faça vários ensaios.  
Que resultados obteve? Justifique.

# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Continuando com o procedimento `faz-roleta`, pretende-se agora desenvolver um teste unitário que permita realizar ensaios automáticos, em grande número. Por exemplo, testes que façam 1000 ensaios a uma função roleta gerada por `faz-roleta`.

Este teste unitário merece uma atenção especial, pois apresenta algumas novidades:

- usa um procedimento para executar ciclos que vai permitir a realização, neste caso, de 1000 ensaios
- usa `with-check-info` para indicar em que ensaio do ciclo de 1000 ocorreu a primeira falha.

```
(require (planet "test.ss" ("schematics" "schemeunit.plt" 2 (= 9))))
(require (planet "text-ui.ss" ("schematics" "schemeunit.plt" 2 (= 9)))))

(load "faz-roleta.scm")

(define rol-20-30 (faz-roleta 20 30)) ; gama 20 a 30
(define beg 1) ; de 1
(define end 1001) ; a 1000
(define step 1)

(define roleta-tests
  (test-suite
    "Testar faz-roleta.scm"

    (test-case
      "testa 1000 casos"
      ;
      (letrec ((ciclo
                (lambda (beg)
                  (if (< beg end)
                      (let ((saiu (rol-20-30)))
                        (with-check-info
                          ('ensaio-corrente beg) ('lancamento saiu)
                          (check-true (>= saiu 20) ">= 20")
                          (check-true (<= saiu 29) "<= 30")) ; erro provocado...
                      ;
                      (ciclo (+ beg step)))))))
      ;
      (ciclo 1))))
;
(test/text-ui roleta-tests)
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Num primeiro ensaio deste teste unitário resultou:

```
Testar faz-roleta.scm > testa 1000 casos
testa 1000 casos has a FAILURE
ensaio-corrente: 6
lancamento: 30
name: check-true
location: #<struct:object:...pper/stepper-tool.ss:618:8>:26:23
params:
#f
message: "<= 30"
0 success(es) 1 failure(s) 0 error(s) 1 test(s) run
1
```



**Analise cuidadosamente o resultado fornecido pelo teste unitário e, com base nisso, procure identificar a sintaxe de `with-check-info`.**



**A probabilidade de executar este teste unitário sem detectar uma falha é praticamente zero...  
Explique por que razão, nas mensagens obtidas, no campo `ensaio-corrente` o valor é normalmente diferente de execução em execução, mas o valor no campo `lancamento` é sempre 30.**



Experimente o teste unitário `faz-roleta-test-3.scm`.

Altere o teste unitário para que possa ocorrer em `lancamento`, para além do valor 30, também o valor 29. Faça alguns ensaios e justifique os resultados que obteve.

Finalmente, corrija o teste unitário para que funcione correctamente. Faça vários ensaios e justifique os resultados que obteve.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 2

Se pretender não utilizar o procedimento auxiliar `ciclo`, o Scheme disponibiliza uma alternativa, o procedimento `do`. Crie um novo teste unitário a partir do teste armazenado em `faz-roleta-test-3.scm`, substituindo o procedimento auxiliar `ciclo` pelo procedimento `do`.



Analise o código que se segue e o respectivo resultado e, a partir daí, deduza a sintaxe do procedimento `do`.

```
> (do ((control 5 (+ control 1)))
      ((= control 10))
      (display control)
      (newline))
5
6
7
8
9
>
```



Desenvolva e teste o teste unitário `faz-roleta-test-3.scm`.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Testes Unitários com interface gráfica

Até aqui, os resultados fornecidos pelos testes unitários surgiam sobre a forma textual. O exemplo que se segue refere-se aos 1000 ensaios de um procedimento gerado por faz-roleta.

O teste unitário com a designação faz-roleta-test-graph.scm, mostra, a negrito, o código introduzido para substituir o código anterior.

```
(require (planet "test.ss" ("schematics" "schemeunit.plt" 2 (= 9))))
; importação da biblioteca SchemeUnit

;(require (planet "text-ui.ss" ("schematics" "schemeunit.plt" 2 (= 9))) ; código anterior
(require (planet "graphical-ui.ss" ("schematics" "schemeunit.plt" 2 (= 9)))

; importação da interface gráfica da biblioteca SchemeUnit

(load "faz-roleta.scm")
(define rol-20-30 (faz-roleta 20 30)) ; faz roleta na gama 20 a 30
; para realizar 1000 testes de 1 a 1000
(define beg 1) ; de 1
(define end 1001) ; a 1000
(define step 1)

(define roleta-tests
  (test-suite
    "Testar faz-roleta.scm"
    (test-case
      "testa 1000 casos"
      ;
      (do ((control beg (+ control step))) ; usando do
          (= control end)) ; termina a iteração
      ;
      (let ((saiu (rol-20-30)))
        (with-check-info
          ('ensaios-corrente control) ('lancamento saiu))
        (check-true (>= saiu 20) ">= 20")
        (check-true (<= saiu 30) "<= 30")))))
;

;(test/text-ui roleta-tests) ; código anterior
(test/graphical-ui roleta-tests)
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

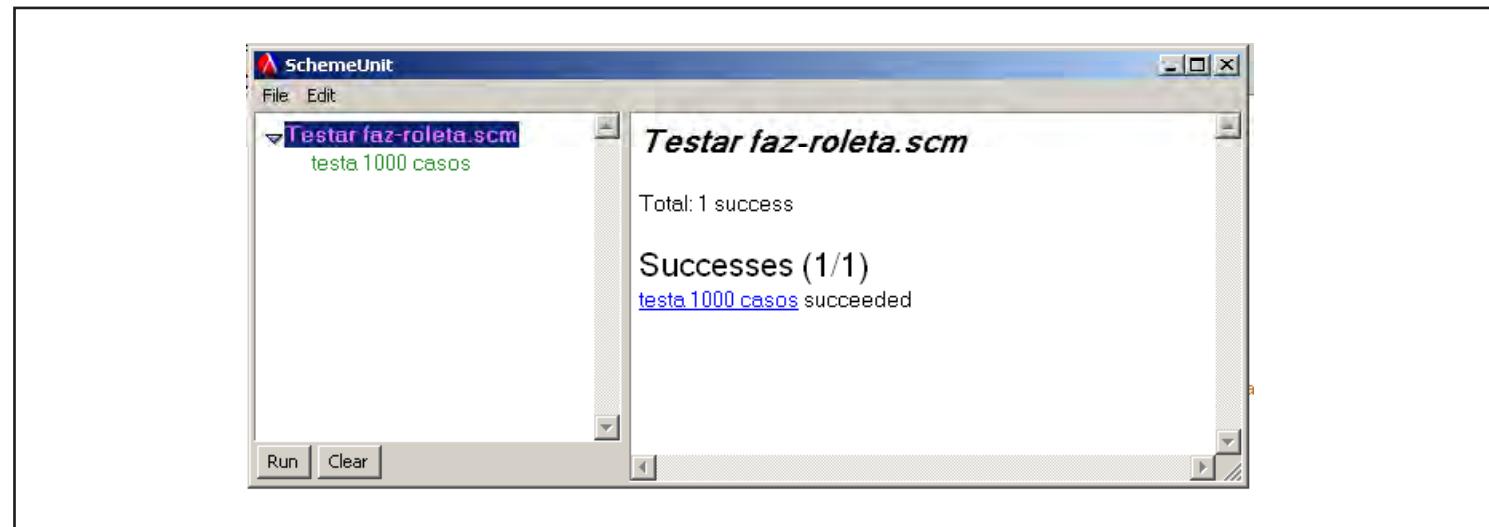
R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

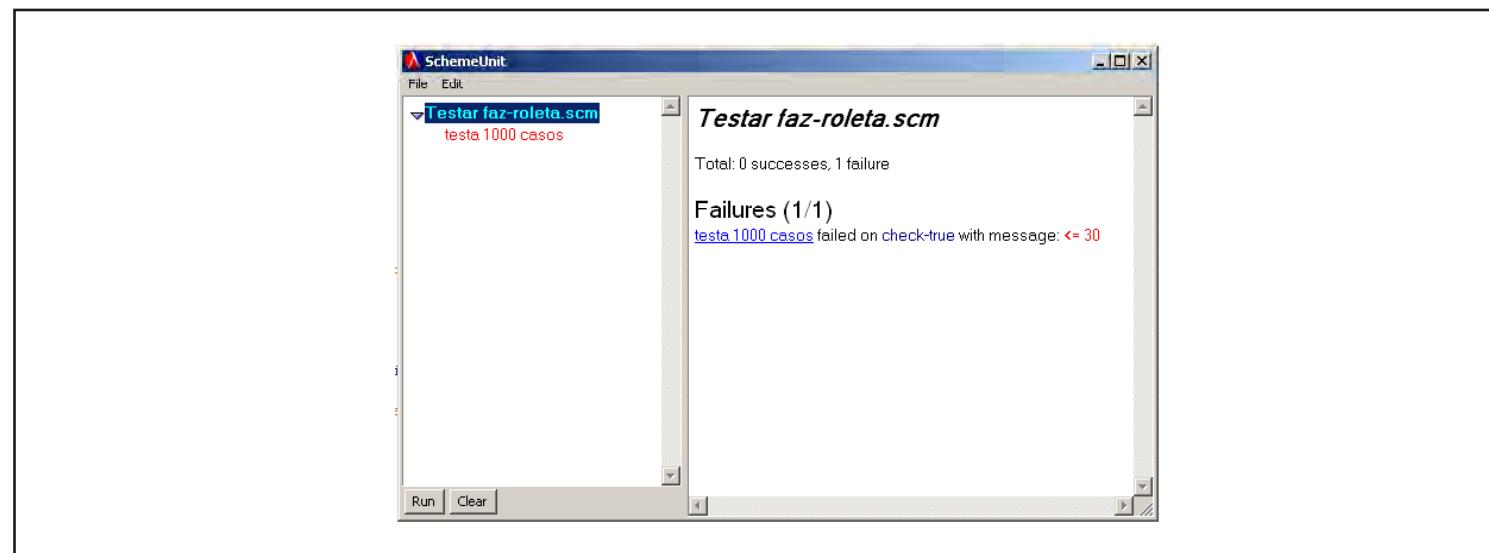


A figura mostra a janela produzida quando é executado este novo teste unitário:



As duas figuras que se seguem mostram duas vistas da janela produzida quando ocorre erro, pois agora foi introduzido um teste que está, intencionalmente, incorrecto:

```
(check-true (<= dice 29) "<= 30") ; erro provocado
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

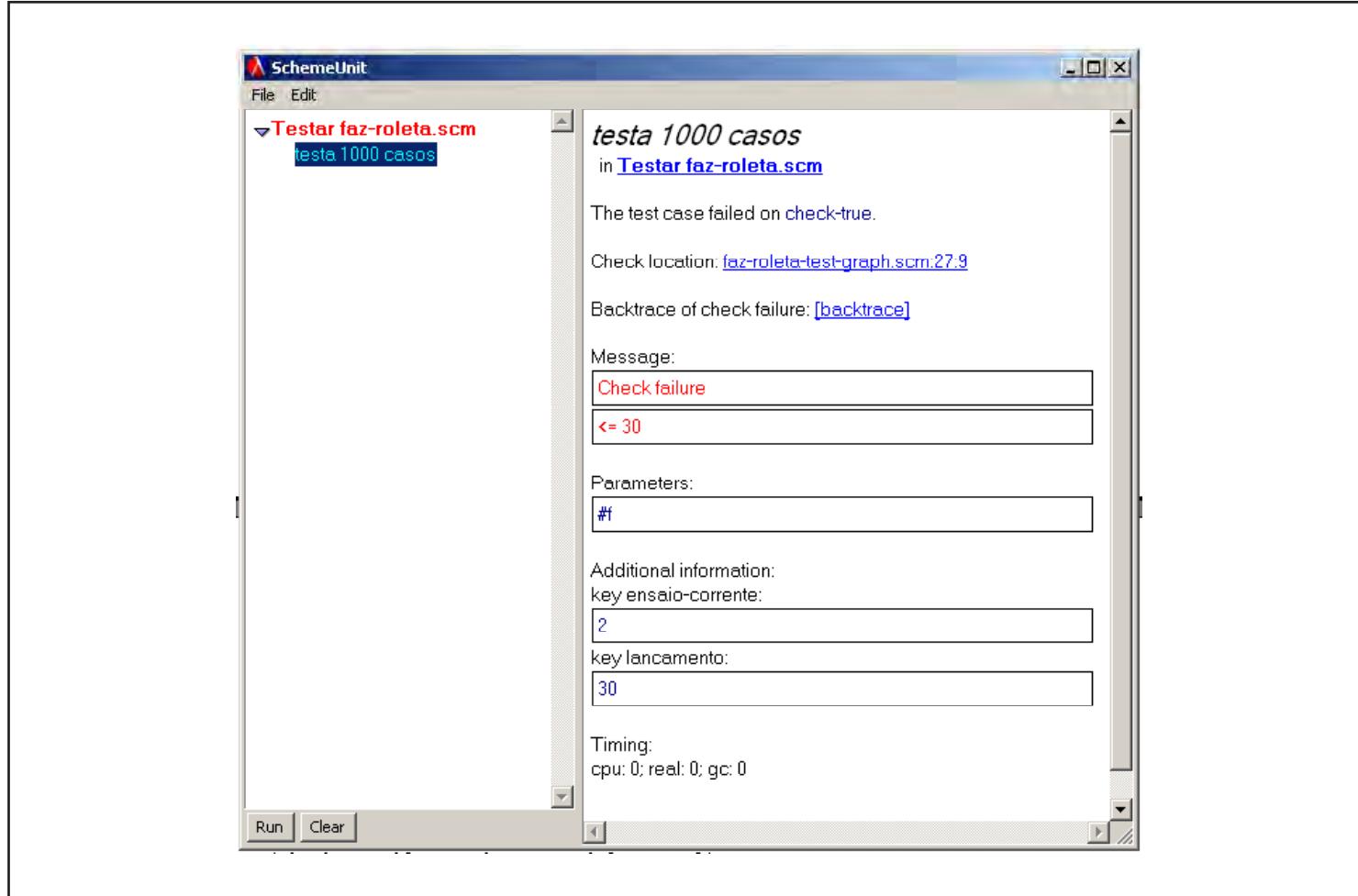
### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Teste o teste unitário `faz-roleta-test-graph.scm`.

Aproveite para alterar ou juntar mais alguns testes, incluindo também casos que provoquem falhas, e verifique o que resulta da execução do teste unitário.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## Código com exceções

O tratamento de exceções tem fugido à abordagem ao Scheme que temos vindo a fazer. No entanto, parece que surgiu o momento para uma breve visita ao tema, o que permitirá desenvolver testes unitários para código que preveja situações de exceção.

Considere o primeiro exemplo apresentado neste módulo, o procedimento factorial tal e qual como ele nos tem aparecido.

```
; Procedimento recursivo com um parâmetro, num.  
; Devolve o factorial de num.  
;  
(define factorial  
  (lambda (num)  
    (if (zero? num)  
        1 ; o caso base  
        (* num (factorial (sub1 num)))))) ; o caso geral
```

Uma chamada a factorial com um valor negativo, por exemplo (factorial -2), para além de não fazer grande sentido, provavelmente bloquearia o DrScheme, pois a condição de terminação dificilmente seria alcançada. E algo de semelhante ocorreria se o argumento fosse um não inteiro, por exemplo (factorial 2.3).



**Explique o que aconteceria com (factorial 2.3)  
e  
também com a chamada (factorial 'abc).**

A nossa intenção não é alargar muito o tema das exceções, mas apenas tratar aquilo que é essencial e que vai começar a ser ilustrado com a nova versão do cálculo do factorial, agora através do procedimento fac.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
(define fac
  (lambda (n)
    (cond ((not (integer? n))
            (error 'error-in-user-proc "~s; ~s: not positive integer." 'fac n))
           ((negative? n)
            (error 'error-in-user-proc "~s; ~s: not positive integer." 'fac n))
           (else
            (fac-aux n)))))

(define fac-aux
  (lambda (n)
    (if (zero? n)
        1
        (* n (fac-aux (sub1 n))))))
```



Que novidade apresenta o procedimento **fac**?

Pista: No diálogo que se segue, procure identificar como funciona o procedimento **error**.

```
> (error 'simbolo "abc def ~s ghil" -7)
simbolo: abc def -7 ghil
>
```

Neste pequeno diálogo, pode verificar que a chamada de **error** aparece com 3 argumentos e que da execução desta chamada resulta a visualização no ecrã de: o primeiro argumento de **error** seguido de ":" e depois uma cadeia de caracteres, que é o segundo argumento de **error**, em que "**~s**" é substituído pela cadeia correspondente ao terceiro argumento.

O diálogo que se segue foi extraído do procedimento **fac**. Neste caso, a cadeia de caracteres, que é o segundo argumento de **error**, tem dois "**~s**" e, assim, **error** aparece com 4 argumentos...

```
> (error 'error-in-user-proc "~s; ~s: not positive integer." 'fac -5)
error-in-user-proc: fac; -5: not positive integer.
>
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



**error** até parece confundir-se com **format**,

```
> (format "Soma ~s e ~s vale ~s." 5 13 (+ 5 13))  
"Soma 5 e 13 vale 18."  
> (display (format "Soma ~s e ~s vale ~s." 5 13 (+ 5 13)))  
Soma 5 e 13 vale 18.  
>
```

**mas error** é muito mais do que isto...

Vamos aprofundar um pouco mais o que realmente acontece quando **error** é chamado, pois não é tão simples como nos poderão fazer crer os pequenos diálogos indicados.

Os argumentos de **error**, se forem bem escolhidos, identificam o erro ocorrido e o local onde ocorreu.



**Se recuar um pouco, até fac, pode analisar com atenção os argumentos das duas chamadas do procedimento error.**

O procedimento **error** não visualiza no ecrã, como poderá neste momento pensar! Este procedimento constrói uma estrutura com os seus argumentos, que vamos designar por **exn**, e, seguidamente, lança uma excepção.

Uma excepção provoca uma rotura na sequência normal do programa, chamando um procedimento especial, o processador de excepções (*exception handler*) corrente. Se nada fizer em contrário, o processador de excepções será o que o Scheme disponibiliza por omissão.



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



O *exception handler* corrente disponibilizado pelo Scheme limitou-se a enviar para o ecrã parte da estrutura construída por `error`, estrutura que designámos por `exn`, e, de seguida, interrompeu o programa. É isto que acontece, de uma forma mais evidente, quando se tenta executar o procedimento `ciclo` e, entre os dados fornecidos, vai algum que não é adequado...

```
(define ciclo
  (lambda (i)
    (if (not (zero? i))
        (begin
          (display "numero: ")
          (let ((numero (read)))
            (display "factorial: ")
            (display (fac numero))
            (newline)
            (ciclo (sub1 i))))))
    > (ciclo 5)
  numero: 3
  factorial: 6
  numero: 5
  factorial: 120
  numero: -3
  factorial: error-in-user-proc: fac; -3: not positive integer.
  >
```



Observe que, com o *exception handler* disponibilizado pelo Scheme, a ocorrência de uma excepção, para além de uma mensagem onde poderá ver informação associada à estrutura `exn`, faz interromper o percurso normal do procedimento `ciclo`.



Teste os procedimentos `fac` e `ciclo`.

Aproveite para alterar, por exemplo, os argumentos de `error` no procedimento `fac`.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Em vez do *exception handler* corrente disponibilizado pelo Scheme, poderá preparar um outro *handler* corrente que considere mais apropriado. Antes, porém, observe com atenção os dois procedimentos que se seguem.

```
(define always-true
  (lambda (exn)
    #t))
```



Qual é a particularidade deste predicado, que até se deduz pelo próprio nome?

```
(define my-handler
  (lambda (exn)
    (display (exn-message exn))
    (newline)))
```



Que novidade apresenta o procedimento **my-handler**?

Pista: **exn-message** toma a estrutura **exn** construída pelo procedimento **error** e, desta estrutura, retira uma cópia da respectiva mensagem... como verá mais à frente.

Agora está quase em condições de chamar o procedimento **fac**, associando à chamada o *handler* que pretender. Para isso terá que conhecer, antes de o poder usar, o procedimento **with-handlers** do Scheme.

```
(with-handlers ((exn-predicado exn-handler))
  (expressão-com chamadas-ao-procedimento-fac))
```

O predicado **exn-predicado**, quando **#t**, fará com que **exn-handler**, um *handler* da sua autoria, passe a ser o *handler* corrente, o qual será chamado em caso de exceção.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Sendo always-true sempre #t, isto significa que não escapará nenhuma das excepções ocorridas com fac...

```
> (with-handlers ((always-true
                     my-handler))
  (* 10 (fac 3)))

60
> (with-handlers ((always-true
                     my-handler))
  (* 10 (fac -3)))
error-in-user-proc: fac; -3: not positive integer.
>
```



Teste o procedimento fac com chamadas directas e com chamadas através de *with-handlers*.

Analise a nova versão do procedimento ciclo, agora com chamadas de fac através de with-handlers. Em negrito é realçado o código alterado

```
(define ciclo
  (lambda (i)
    (if (not (zero? i))
        (begin
          (display "numero: ")
          (let ((numero (read)))
            (display "factorial: ")

            ;(display (fac numero)) ; versão anterior...
            ((with-handlers ((always-true my-handler))
              (display (fac numero)))
            (newline)
            (ciclo (sub1 i)))))))
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
> (ciclo 5)
numero: 3
factorial: 6
numero: -3
factorial: error-in-user-proc: fac; -3: not positive integer.
```

```
numero: 5
factorial: 120
numero: 'abc
factorial: error-in-user-proc: fac; (quote abc): not positive integer.
```

```
numero: 7
factorial: 5040
>
```



Que conclusão pode retirar desta experiência, com esta nova versão do procedimento **ciclo**?



Teste os procedimentos **fac** e a nova versão de **ciclo**.



Em que tipo de situações utilizará directamente chamadas ao procedimento **fac**?  
E em que tipo de situações preferirá chamá-lo através de **with-handlers**?



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



## Teste unitário para código com excepções

Para que o teste unitário possa também tratar situações de excepção, vai ter que conhecer mais um `check`, para além dos que já conhece.

```
(check-exn exn-predicate thunk [message])
```

O primeiro parâmetro é um predicado que define que tipo de excepções devem ser consideradas no teste.



Como verá, vamos usar um predicado que devolve sempre `#t`, o que significa que aceitaremos qualquer excepção sem distinção.

Ou também pode querer dizer que, neste momento, não estamos interessados em aprofundar muito mais este assunto...

O segundo parâmetro é um *thunk*, que é um procedimento sem parâmetros. Ou seja, `check-exn`, contrariamente ao que acontecia com os outros `check` que já conhece, não quer uma chamada de `fac`, mas uma chamada atrasada!



Se estiver com dúvidas sobre o que é um *thunk*, veja com atenção o diálogo que se segue.

```
> (fac -4)                                ; chamada normal
   error-in-user-proc: fac; -4: not positive integer.

> (lambda () (fac -4))                  ; thunk, chamada atrasada
#<procedure>                            ; thunk é um procedimento
> (
  (lambda () (fac -4))                  ; agora, execução do thunk
  )
  error-in-user-proc: fac; -4: not positive integer.
> (define p (lambda () (fac -4)))      ; ligação do thunk a p
> p
#<procedure:p>
> (p)                                    ; execução do thunk
  error-in-user-proc: fac; -4: not positive integer.
>
```

Finalmente, o terceiro parâmetro é uma mensagem opcional.

A unidade de teste que agora se apresenta tem como novidade, indicada a negrito, o caso de teste com as situações de excepção.

```
; importação da biblioteca SchemeUnit
; na linha seguinte: ... 2 (= 9))) significa versão 2.9
(require (planet "test.ss" ("schematics" "schemeunit.plt" 2 (= 9)))

; importação da interface textual da biblioteca SchemeUnit
(require (planet "text-ui.ss" ("schematics" "schemeunit.plt" 2 (= 9))))
; carregar o código a testar, neste caso, colocado no ficheiro factorial.scm
(load "excep-error-fac.scm")

; definição dos testes a realizar
(define testes-a-realizar
  (test-suite
    "Testar excep-error-fac.scm"
```

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1ª CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
(test-case
  "Testar procedimento fac"
  (check-equal? (fac 0) 1 "fac 0")
  (check-equal? (fac 1) 1 "fac 1")
  (check-equal? (fac 2) 2 "fac 2")
  (check-equal? (fac 7) 5040 "fac 7"))

(test-case
  "entradas incorrectas, situações de excepção"
  (check-exn always-true
    (lambda () (fac -5)) "entrada negativa")
  (check-exn always-true
    (lambda () (fac 5.3)) "entrada não inteira")
  (check-exn always-true
    (lambda () (fac 'abc)) "entrada não inteira")))

; execução dos testes definidos
(test/text-ui testes-a-realizar)
```



O código que se segue não faz parte do teste unitário. Serve só para mostrar que este teste não se atrapalha com situações que possam provocar excepções, pois o procedimento `fac` está preparado para isso.

```
; pós teste unitário
(newline)
(display "pós teste unitário!")
(newline)
(display "observe o que aconteceria em chamadas com handlers")
(newline)
(display "nas duas situações de excepção...")
(newline)
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
; "entrada não inteira, with-handlers"
(with-handlers ((always-true
                  my-handler))
               (fac 5.3))
;
;"entrada negativa, with-handlers"
(with-handlers ((always-true
                  my-handler))
               (fac -5))
;"entrada negativa, with-handlers"
(with-handlers ((always-true
                  my-handler))
               (fac 'abc))

2 success(es) 0 failure(s) 0 error(s) 2 test(s) run
0

pós teste unitário!
observe o que aconteceria em chamadas com handlers
nas situações de excepção...
error-in-user-proc: fac; 5.3: not positive integer.
error-in-user-proc: fac; -5: not positive integer.
error-in-user-proc: fac; abc: not positive integer.
>
```



Teste o teste unitário fac-excepcoes-test.scm.





Na lista que se segue, encontram outros `check(s)` que não foram utilizados nos testes unitários já desenvolvidos.

- `(check binary-predicate actual expected [message])`
- `(check-equal? actual expected [message])`
- `(check-eqv? actual expected [message])`
- `(check-eq? actual expected [message])`
- `(check-= actual expected epsilon [message])`
- `(check-true actual [message])`
- `(check-false actual [message])`
- `(check-not-false actual [message])`
- `(check-pred unary-predicate actual [message])`
- `(check-exn exn-predicate thunk [message])`
- `(check-not-exn thunk [message])`
- `(fail [message])`

Poderá sempre procurar aprofundar um pouco mais o seu conhecimento sobre este tema, consultando o documento relativo à biblioteca `SchemeUnit.plt`, colocado em: [SchemeUnit: Unit Testing in Scheme](#)

INTRODUÇÃO
1 - O ESSENCIAL DO SCHEME
2 - RECURSIVIDADE
3 - ABSTRACÇÃO DE DADOS
4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE
5 - ABSTRACÇÕES COM DADOS MUTÁVEIS
6 - SCHEME E OUTRAS TECNOLOGIAS
R 1 2 3 4 5
7-EXERCÍCIOS E PROJECTOS
ANEXOS



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



## Módulo 6.4 - Código executável criado a partir de programas Scheme

Os programas que tem encontrado, sejam eles muito simples ou mais complexos, com interface textual ou gráfica, com ou sem interacção, têm sido sempre experimentados no ambiente do Scheme. Tudo se tem passado, necessariamente, num computador onde o Scheme esteja instalado. Certamente que já passou por alguma situação em que pretendeu enviar um dos seus programas mais interessante a um amigo ou a um familiar mas, apesar do acesso ao Scheme ser livre, o seu amigo ou familiar não tinha instalado, nem queria instalar, o Scheme no computador.

A ideia deste pequeno módulo é apresentar a forma de ultrapassar este tipo de situação e mostrar como o DrScheme permite produzir código executável que correrá em computadores, mesmo que estes não tenham o Scheme instalado. Esta possibilidade, como verá, aplica-se não só a pequenos programas, mas também a programas mais complexos, com interface gráfica, como a navegação em labirintos e até jogos...

Neste pequeno módulo é indicada a forma de criar código executável, a partir do DrScheme, sempre ilustrada com vários exemplos de complexidade muito variada.

### Palavras-Chave

DrScheme em *Language module*, código executável.

# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



## É fácil criar código executável a partir de programas Scheme

Em várias situações, já terá pretendido produzir um código executável a partir de programas que tenha desenvolvido em Scheme, para mostrar a algum amigo ou familiar as suas habilidades em programação. Com esse código executável, o feliz destinatário do seu código podê-lo-á experimentar sem precisar de instalar o Scheme.

Vamos começar com um programa de teste muito simples.

The screenshot shows the DrScheme interface. The menu bar includes File, Edit, View, Language, Scheme, Special, Help, and a dropdown showing 'ola.scm'. Below the menu is a '(define ...)' dropdown. The code area contains the following Scheme code:

```
(define teste
  (lambda (x)
    (display "O teu nome: ")
    (let ((nome (read)))
      (newline)
      (display "ola ")
      (display nome)
      (display "!")
      (newline)
      (display "O argumento vale: ")
      (display x)))
```

The interaction window below shows the program's output:

```
Welcome to DrScheme, version 372 [3m].
Language: Graphical (MrEd, includes MzScheme).
> (teste 25)
O teu nome: Maria

ola Maria!
O argumento vale: 25
>
```



Tenho quase a certeza que este programa não lhe trará qualquer novidade.



Teste, se pretender, este pequeno programa.

# S C H E M E

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

Até aqui, no DrScheme, temos utilizado Language: Graphical (MrEd, includes MzScheme). Pois agora, no contexto deste Módulo, vamos escolher Language: (module ...) e introduzir as alterações ao código, para o transformar num módulo.



Um pormenor importante é o nome do módulo, neste caso, **ola** que DEVE SER igual ao nome do ficheiro respetivo (sem a extensão).

```
(module ola
  mzscheme
;-----;

(define teste
  (lambda (x)
    (display "O teu nome: ")
    (let ((nome (read)))
      (newline)
      (display "ola ")
      (display nome)
      (display "!"))
    (newline)
    (display "O argumento vale: ")
    (display x)))

; --- código acrescentado:
(teste 25))
```



Compare as duas versões do código que está a experimentar.  
Qual o inconveniente da segunda versão em relação à primeira?



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

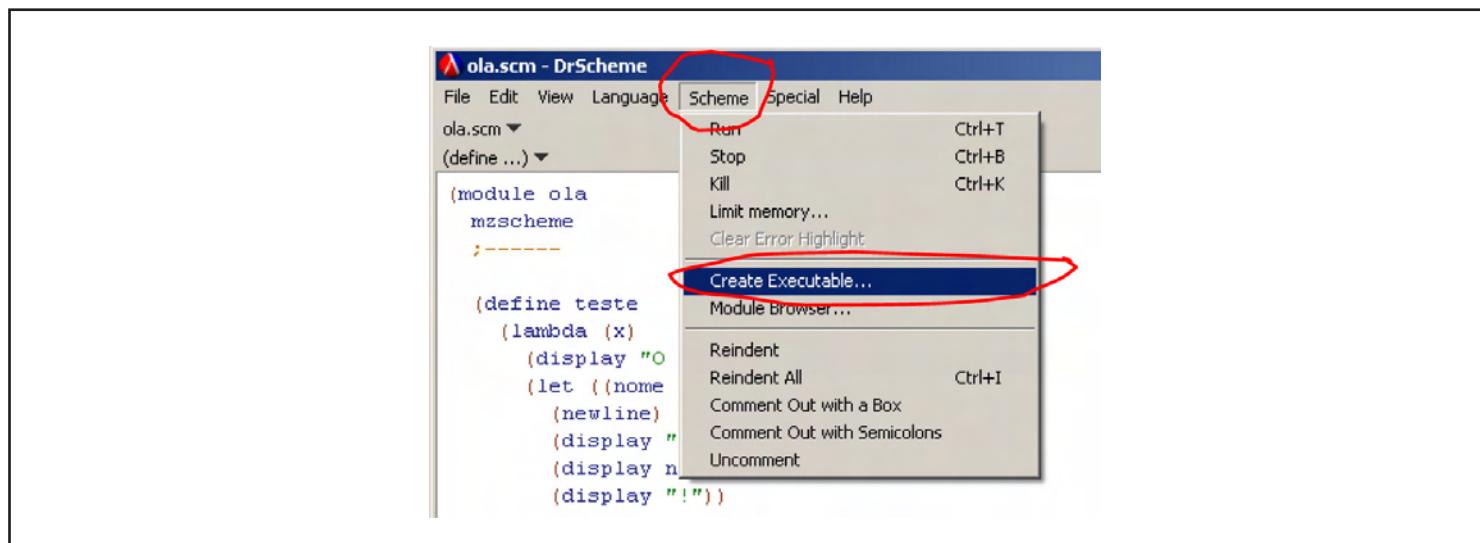
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

Welcome to DrScheme, version 372 [3m].  
Language: (module ...).  
O teu nome: Manuel  
  
ola Manuel!  
O argumento vale: 25  
>

Esta versão, em que o código é transformado num módulo para ter uma interactividade semelhante à primeira, deveria ser alterada para pedir o argumento do procedimento teste. Como está, esse argumento será sempre 25...

A criação do código executável, no DrScheme, começa no menu Scheme, seleccionando a opção Create Executable.



Na janela de diálogo aberta, deve escolher Distribution e MrED, fazer Create e



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

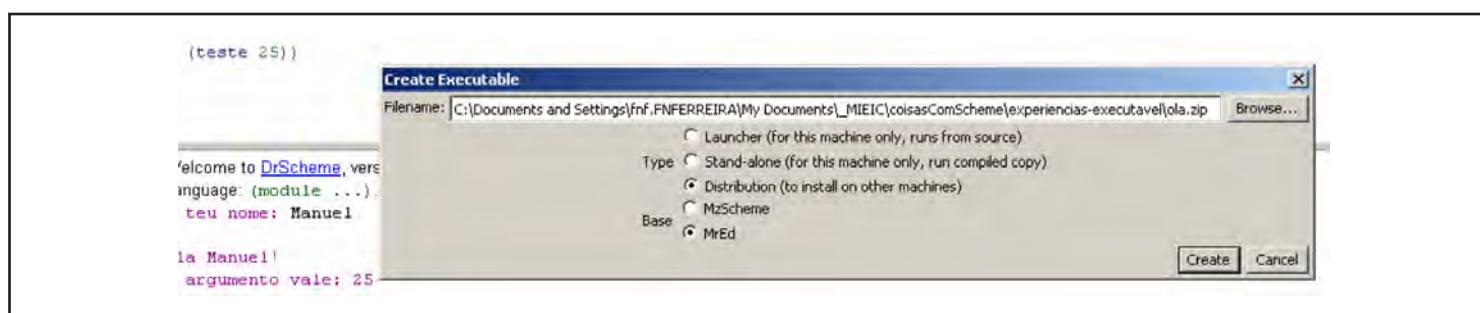
## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



esperar que surja, no directório onde se encontra ola.scm, o ficheiro ola.zip.



As demonstrações apresentadas neste módulo foram desenvolvidas em Windows. Para outros sistemas operativos, o processo é equivalente, embora as versões executáveis sejam produzidas noutro formato, de acordo com as especificidades desses sistemas.



Tem alguma explicação para o enorme tamanho do ficheiro ola.zip?

Este ficheiro ola.zip contém todos os recursos para executar o programa respectivo, sem qualquer ajuda do DrScheme, e está pronto para ser enviado a um amigo ou familiar que tenha um computador com o mesmo sistema operativo que o seu. Aquele amigo ou familiar começará por descomprimir o ficheiro recebido, ola.zip. Mas vamos ver o que está no directório ola, agora já descomprimido.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

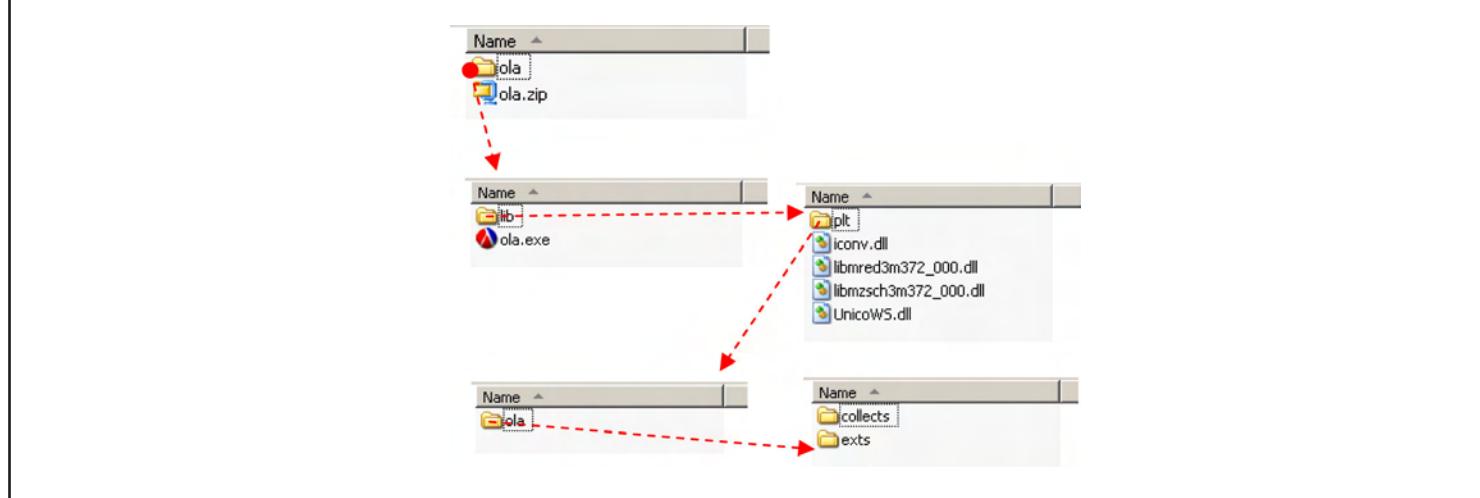
### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

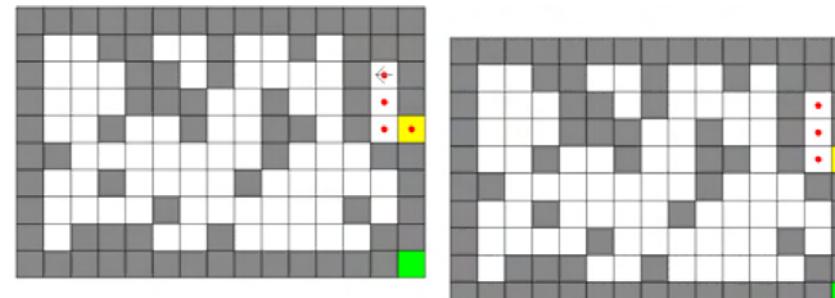


Se continuasse, verificaria que o directório **collects** está vazio.

Bastará actuar sobre **ola.exe** para executar o programa, independentemente de ter ou não instalado o Scheme no seu computador.

**Um exemplo com saída gráfica e sons...**

Este novo exemplo refere-se à travessia de um labirinto, onde uma pequena nave, não muito inteligente, procura uma saída. A nave parte da célula amarela e procura um caminho para chegar à célula verde.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

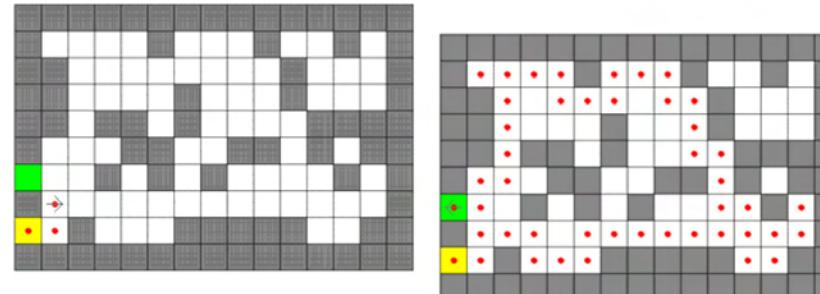
### ANEXOS



Mas nem sempre consegue, pois, por vezes, a saída encontra-se completamente bloqueada. Não há um caminho possível! Depois de algum esforço, a nave acaba por chegar ao ponto de partida e, nesta situação, emite um sinal de frustração e tristeza, quer de uma forma textual quer de uma forma sonora.

```
Voltei ao inicio!!!...
Não consegui sair!!!
Bem me esforcei (só às vezes), mas sem êxito!!!
E não vale rir! Será que conseguiram melhor que eu?
```

Quando a saída está aberta, com voltas e mais voltas (confirma-se que a nave não é muito dotada de inteligência, pois a célula verde estava muito perto...), a nave chega ao fim, aí com uma forte expressão de grande orgulho e alegria.



```
Desta vez consegui atingir o objectivo
Sou o Maior... ou quase...
>
```

Apesar da complexidade deste programa que, nesse aspecto, em nada se compara com o pequeno programa inicial, as alterações necessárias para o transformar numa versão executável são muito semelhantes.

A obtenção do ficheiro labirinto.zip segue o caminho já indicado para a obtenção de ola.zip.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
(module labirinto
  mzscheme
  ; labirinto usado no Módulo da Introdução à programação
  ; através da abstracções
  ;
  (require (lib "audio.scm" "user-feup"))
  (require (lib "tabuleiro.scm" "user-feup"))
  (require (lib "naves.scm" "user-feup"))

  ; -- prepara uma janela gráfica onde será visualizado um tabuleiro -----
  (define largura-jan 680)
  (define altura-jan 480)
  (define titulo-jan "experiencia com labirintos simples")
  ;
  (define jan-tab (janela largura-jan altura-jan titulo-jan))
  ...
  ...
  (nave-para-fim nave)))))

;----- (nave-no-labirinto n))
```

No entanto, surge agora um problema adicional e este ficheiro, contrariamente ao que acontecia no exemplo anterior, não está preparado para ser enviado para outra pessoa!

E porquê?

Simplesmente porque o "compilador" não faz ideia dos ficheiros de som, com extensão wav, situados em PLT\collects\user-feup, que são utilizados pelo nosso programa. Assim, vai ter que dar uma ajudinha ao compilador, com um pouco de esforço.

- Aqui tem um directório local, user-feup-sons-imagens, onde encontrará um outro directório com o nome user-feup, este apenas com uma cópia dos ficheiros dos sons (wav) e dos ficheiros de imagens (bmp) colocados no user feup do Dr. Scheme.
- Passe agora à descompressão do ficheiro labirinto.zip e, a partir do directório labirinto obtido, siga o caminho PLT\labirinto\collects. E, neste collects, coloque uma cópia do user-feup dos sons e das imagens.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

- Regresse ao directório labirinto, descomprimido e, por compressão, obterá um novo labirinto.zip, mas este agora já com os ficheiros de sons e imagens.

Depois destas operações, tem o ficheiro labirinto.zip pronto para enviar ao seu amigo, como aconteceu, no primeiro exemplo, com ola.zip.

O seu amigo começará por descomprimir o ficheiro recebido, labirinto.zip, e actuará em labirinto.exe, para ver uma nave atravessar um labirinto.

### Um jogo, um exemplo com sons e interacção gráfica

Este exemplo é o jogo Pong e, dentro de momentos, vai poder jogar ténis com o computador.

Terá o rato para controlar a sua raquete, mas se preferir também pode usar o teclado. Para além de ouvir a bola, quando esta bate nas paredes ou nas raquetes, também poderá usufruir de uma música de fundo, que poderá desligar em qualquer altura. Este jogo, desenvolvido em Scheme, sobre uma biblioteca multimédia designada por Allegro, foi o exemplo que ilustrou um dos Módulos desta Parte que está agora a descobrir.

O jogo Pong, para além do Allegro, utiliza ainda recursos de som e imagem do collects\user-feup e, por este facto, requer os mesmos cuidados que o exemplo anterior, ou seja, depois da obtenção do pong.zip, deve descomprimi-lo, seguir o directório pong e colocar, em collects, uma cópiado directório user-feup, com os ficheiros dos sons e das imagens. Depois deve voltar novamente ao directório pong e comprimí-lo, para obter o novo pong.zip, este já com os ficheiros de som e imagem e preparado para ser distribuído. Estas operações são exactamente as mesmas que as apresentadas no exemplo do labirinto.

Antes de lhe dar acesso ao jogo, através de pong.exe, pode antever, nas páginas que se seguem, uma possível sessão com este jogo...

Eis o diálogo inicial, a partir do qual o computador quer conhecer o nome do jogador adversário.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

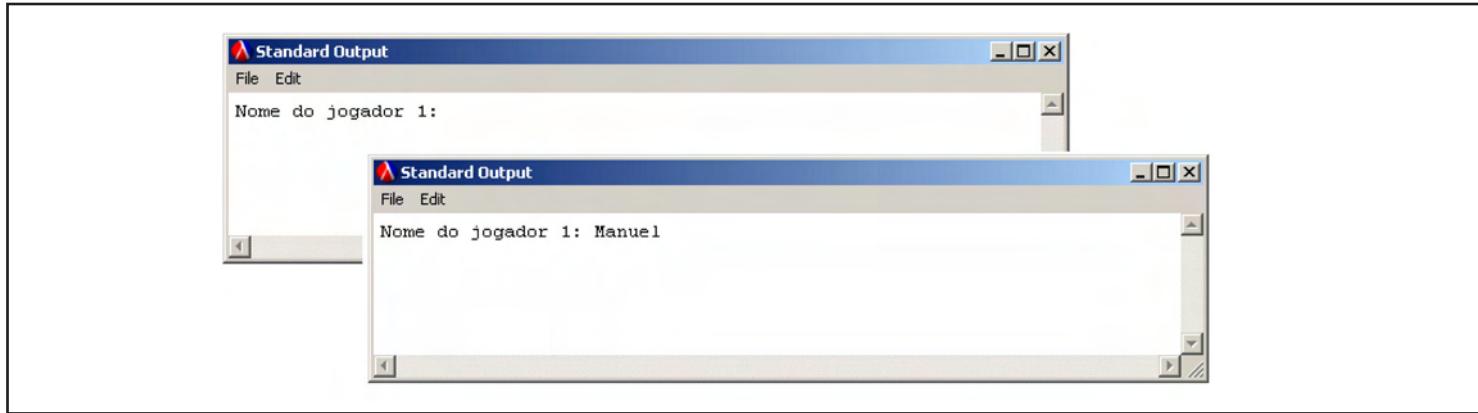
### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

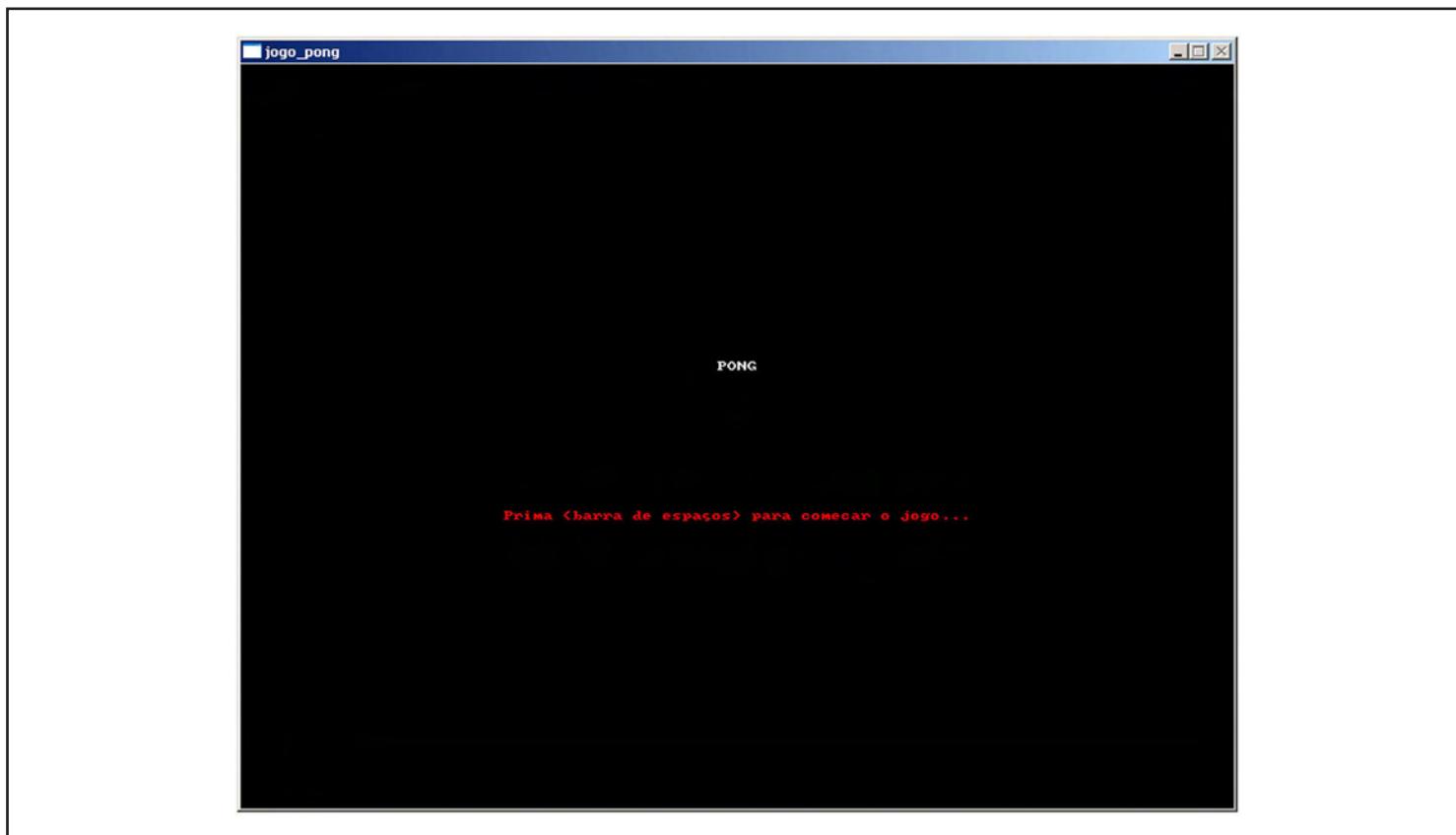
R 1 2 3 4 5

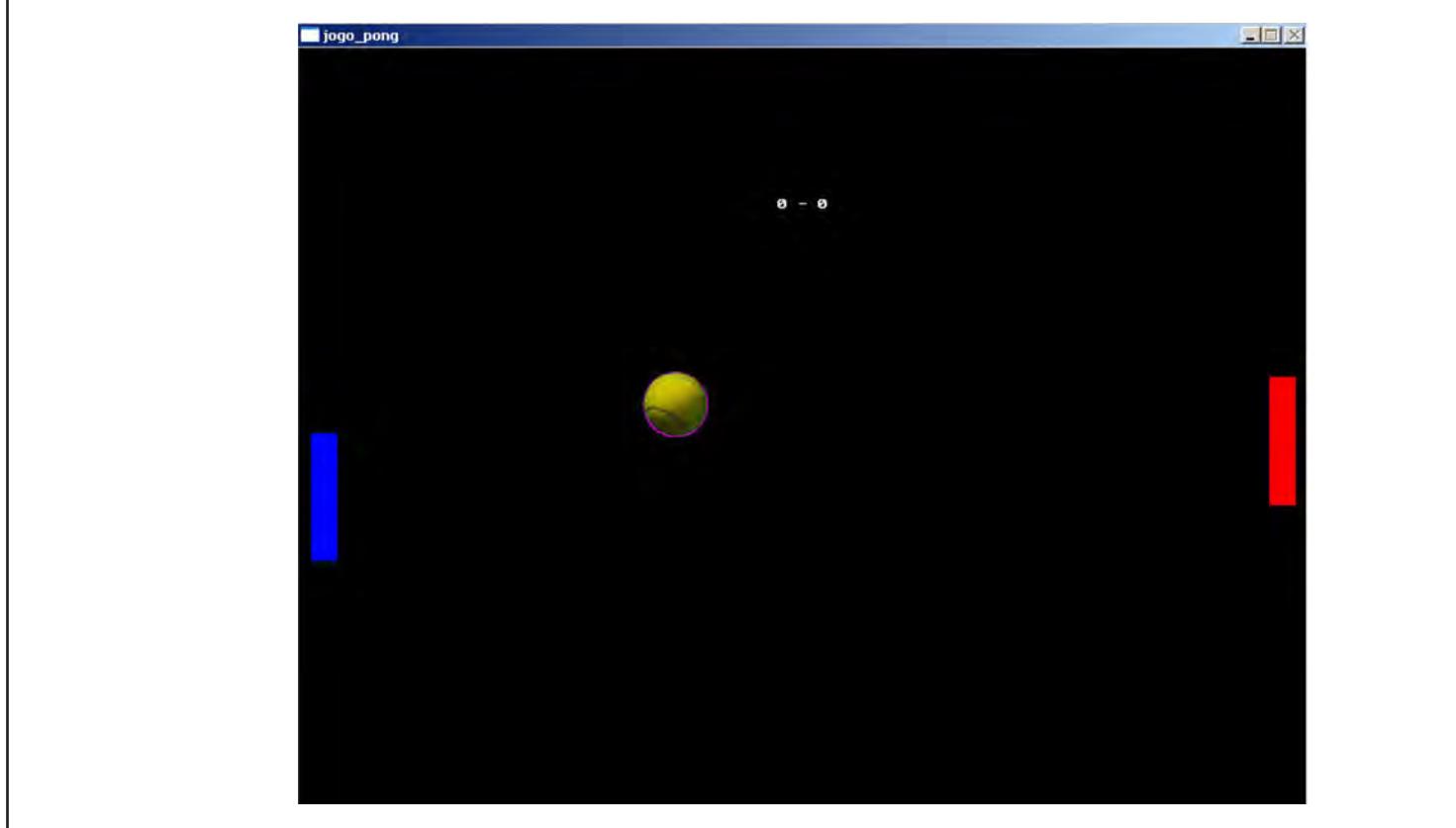
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



A janela gráfica inicial representa o período de preparação antes da bola aparecer. Está nas suas mãos dar início ao jogo, bastando, para tal, actuar o teclado, na barra dos espaços.





• • •

O marcador ainda está a 0 - 0 e a bola aproxima-se perigosamente do seu campo de acção....

• • •

Agora, o marcador já vai em 0 - 2, ou seja, o computador já lhe ganhou!

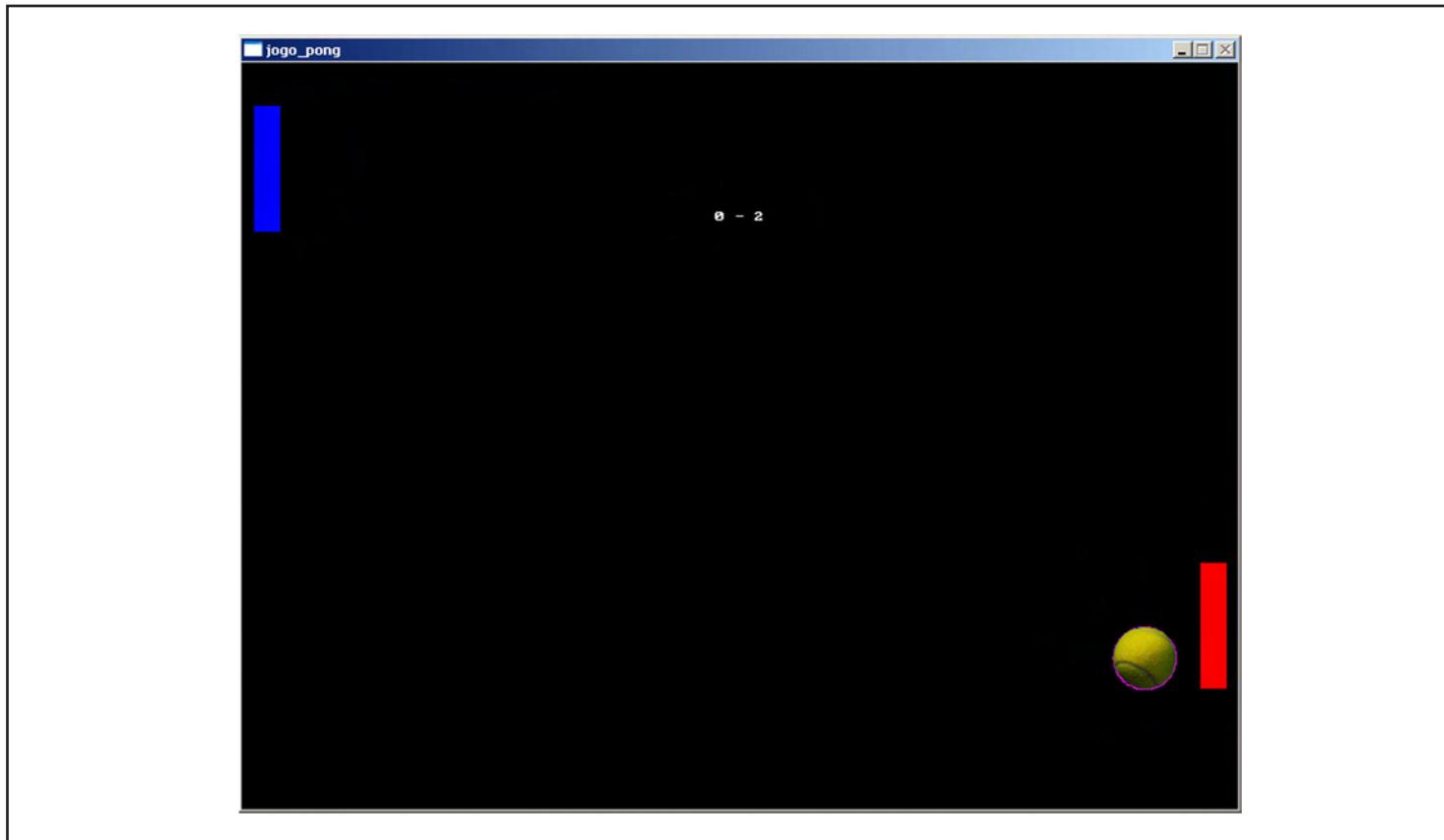
Pode disfarçar com um pouco de música, tecla M (ou m), ou optar por silêncio, tecla S (ou s).

Se o marcador está assim por causa do rato, pode optar por controlar a raquete com o teclado, tecla T (ou t), e a sua raquete passa a obedecer às teclas do cursor UP e DOWN. Mas se achar que assim ainda é mais complicado, pode sempre regressar ao rato, tecla R (ou r).

# **SCHEME**

## na descoberta da programação

- INTRODUÇÃO
  - ESSENCIAL DO SCHEME
  - RECURSIVIDADE
  - STRACÇÃO DE DADOS
  - PROCEDIMENTOS COMO PROECTOS DE 1<sup>a</sup> CLASSE
  - ABSTRACCÓES COM DADOS MÚTÁVEIS
  - SCHEME E OUTRAS TECNOLOGIAS
  - R 1 2 3 4 5
  - 7- EXERCÍCIOS E PROJECTOS
  - ANEXOS



Deve ter chegado a 0 - 5, talvez, e o computador declarou vitória!...



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

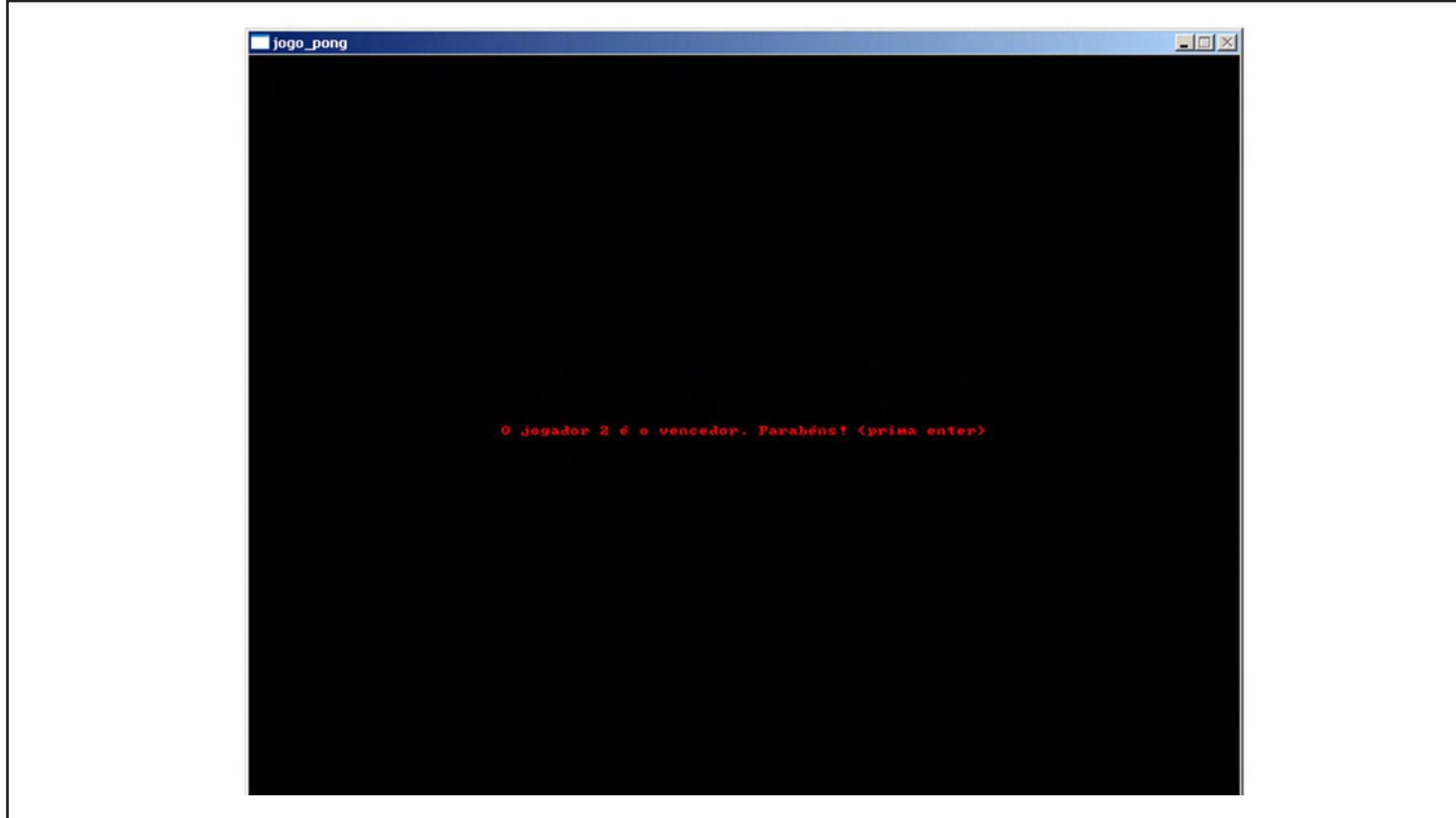
5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

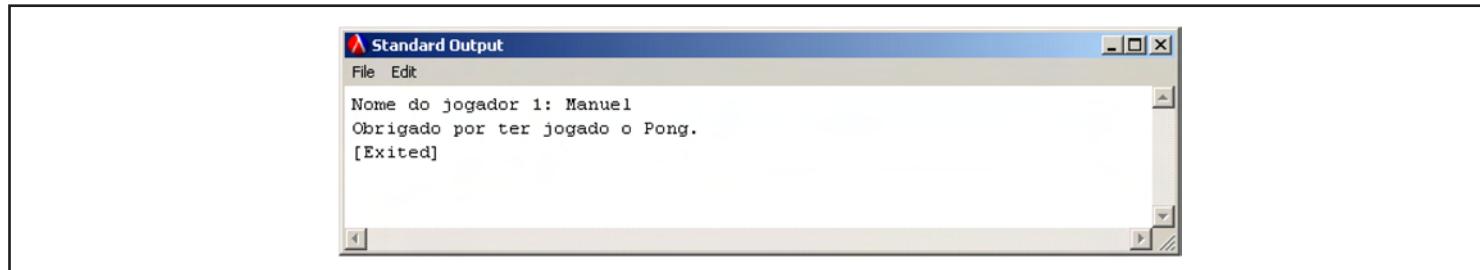
R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



Em vez da tecla `enter`, que lhe facultaria a desforra, poderá optar por actuar a tecla `Esc` (em qualquer altura do jogo), e o programa termina de imediato.



E, agora, está convidado para uns jogos de ténis com o seu computador, pois o ficheiro executável, [jogo\\_pong.exe](#), já está preparado.

Boas jogadas!

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Módulo 6.5 - A programação OO com o Scheme

Na programação orientada por objectos, programação OO, um ou mais objectos são criados com características semelhantes a partir de uma classe de objectos. Por exemplo, da classe rectângulo é possível criar rectângulos de vários tamanhos, cada um com a sua base e altura. A base e a altura serão atributos desta classe rectângulo. Os objectos de uma classe recebem mensagens e estão preparados para responder a cada uma dessas mensagens, através de uma determinada operação ou método. A mensagem área, chegada a um objecto rectângulo, será atendida com a execução do cálculo da sua área. Algo semelhante acontecerá, certamente, para a mensagem perímetro. A área e o perímetro serão operações ou métodos da classe rectângulo.

A partir de uma classe de objectos, a classe base, é possível derivar novas classes, as classes derivadas, que herdam total ou parcialmente os atributos e as operações da classe base. A classe polígono, reunindo o que os rectângulos e os triângulos têm em comum, poderia ser a classe base da qual derivavam as classes rectângulo e triângulo...

O grande interesse da programação OO reside, especialmente, na modularidade que disponibiliza, pois cada objecto surge como um bloco, o qual protege ou encapsula os seus atributos, condicionando o respectivo acesso às suas operações preparadas para esse efeito. Outro interesse da programação OO está na facilidade de reaproveitamento do seu código, provavelmente até desenvolvido por equipas ou programadores diversos.

O Scheme não é uma linguagem vocacionada para a programação orientada por objectos, como acontece com linguagens como o Java, C++, Smalltalk ou Simula, mas permite, com alguma facilidade, a implementação dos principais conceitos deste importante paradigma de programação. Terá assim a oportunidade de contactar com a implementação dos conceitos de classe, objecto, mensagem, operação, herança, entre outros, isto numa primeira fase. Só depois passará à respectiva utilização em vários exemplos e exercícios. Ora, não é isso o que acontece com aquelas linguagens, onde apenas se limita a usar esses conceitos, pois já os encontra implementados...

### Palavras-Chave

Programação OO, classe, objecto, atributo, mensagem, operação ou método, herança, encapsulamento de dados, reutilização de código, *LIFO*, *FIFO*.



# **S C H E M E**

## **na descoberta da programação**

### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

R 1 2 3 4 5

### **7-EXERCÍCIOS E PROJECTOS**

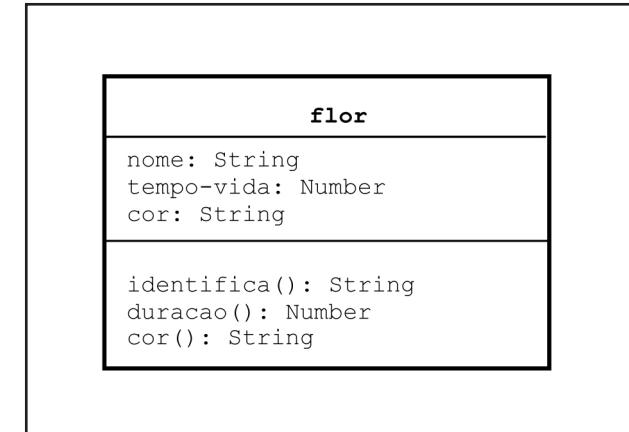
### **ANEXOS**

## **Afinal, uma classe é um procedimento... que devolve um outro procedimento**

Em geral, um objecto de uma classe evidencia um conjunto de características que serão comuns aos objectos dessa classe. Um objecto, com os seus atributos específicos, reage a um certo conjunto de estímulos que lhe chegam, por exemplo, através de mensagens.

Vamos começar com um exemplo simples, a classe `flor`, que tem como atributos um nome para identificação, um valor que representa o tempo médio de vida e uma cor. Esta classe `flor` responde a um conjunto de três mensagens, cada uma das quais fará despoletar uma operação: de identificação, de duração e de cor.

Tudo isto é representado no diagrama da classe `flor`, como se mostra na figura. No topo superior surge a designação da classe, expressa a negrito, na parte central aparecem os atributos e os respectivos tipos e na parte inferior as operações, com a designação seguida da lista de parâmetros e o tipo do valor de retorno.



**Que outros atributos acharia bem incluir na classe `flor`?  
E quanto a operações?**

A partir da classe `flor` será possível criar vários objectos deste tipo (instâncias da classe), com os seus atributos específicos de identificação, duração e cor e com as suas operações.

A classe `flor`, em Scheme, é um procedimento cujos parâmetros estão intimamente associados aos atributos dos objectos desta classe.



# SCHEME

## na descoberta da programação



Analise o procedimento **flor**, no que se refere aos seus parâmetros.  
Veja o que é devolvido por este procedimento: um procedimento!  
O que tem a dizer sobre o parâmetro **msg** deste procedimento devolvido?

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(define flor
  (lambda (nome tempo-vida cor)
  ;
  (let ((n nome)
        (tv tempo-vida)
        (c cor))
    (lambda msg
      (let ((m (car msg)))
        (cond
          ; --- recursos próprios ---
          ((equal? m 'identifica) n)
          ((equal? m 'duracao) tv)
          ((equal? m 'cor) c)
          ; - mensagem desconhecida nesta classe -
          (else
            (display (car msg))
            (display ": mensagem desconhecida!..."))
            (newline)))))))
```

O procedimento **flor** que, em Scheme, implementa a classe **flor**, funciona como um construtor de objectos do tipo **flor**. Passemos a algumas experiências:

```
> (define flor-1 (flor "cravo" 12 "vermelho"))
> (flor-1 'identifica)
"cravo"
> (flor-1 'cor)
"vermelho"
> (flor-1 'duracao)
12
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (define flor-2 (flor "rosa" 4 "amarelo"))
> (flor-2 'cor)
"amarelo"
> (flor-2 'duracao)
4
> (flor-2 'identifica)
"rosa"
>
```



Teste a classe `flor`.

Crie objectos do tipo `flor`, envie-lhes mensagens e observe como eles respondem...



**Uma chamada ao procedimento `flor` cria um novo objecto deste tipo, uma nova instância da classe `flor`. O procedimento `flor` é, por isso, um construtor desta classe.**

No sentido de simplificar a escrita deste tipo de procedimentos, em que surge habitualmente uma sequência de condições, neste caso, um mesmo símbolo (que identifica uma mensagem) e que vai sendo sucessivamente comparado com outros símbolos, introduz-se mais uma estrutura de selecção, designada por `case`, que assim se junta às duas outras estruturas já conhecidas, `if` e `cond`.

Para entender o funcionamento de `case`, que é mais uma forma especial da linguagem Scheme, comece por analisar os dois procedimentos que se seguem (um baseado em `cond` e outro em `case`), em que ambos determinam se uma letra é vogal ou consoante.

```
; recebe um símbolo com uma letra
; e indica se é vogal ou consoante
(define vogal-ou-consoante-cond
  (lambda (letra)
    (cond ((or (equal? letra 'a)
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
(equal? letra 'e)
(equal? letra 'i)
(equal? letra 'o)
(equal? letra 'u))

'vogal)
(else 'consoante))))
```

```
> (vogal-ou-consoante-cond 'a)
vogal
> (vogal-ou-consoante-cond 'f)
consoante
> (vogal-ou-consoante-cond 'a)
vogal
> (vogal-ou-consoante-cond 'f)
consoante
>
```

A mesma situação, mas agora abordada com a forma especial `case`, surge com um aspecto muito mais compacto e simples de ler.

```
(define vogal-ou-consoante-case
  (lambda (letra)
    (case letra
      ((a e i o u) 'vogal)
      (else 'consoante))))
```

```
> (vogal-ou-consoante-case 'a)
vogal
> (vogal-ou-consoante-case 'f)
consoante
>
```

A estrutura de selecção `case` justifica uma pequena explicação adicional.

```
(case expressão
  ((chave1-1 chave1-2 ...) exp1 ...) ; cláusula chave1-i
  ((chave2-1 chave2-2 ...) exp2 ...) ; cláusula chave2-i
  ...
  (else exp-else-1 exp-else-2 ...) ) ; cláusula else
```



# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

Do cálculo de expressão deve resultar um símbolo, um número ou uma cadeia de caracteres, que são os tipos de dados possíveis das chaves indicadas.

A regra de cálculo da forma especial case é a seguinte:

- Calcula expressão;
- O valor resultante vai sendo comparado com as chaves designadas por chave1-i e, se não for igual a nenhuma delas, passará a ser comparado com as chaves chave2-i, e assim sucessivamente até encontrar uma chave igual a esse valor.  
Logo que se encontre uma chave igual ao valor expressão, calcula as expressões correspondentes;
- Se não se encontrar qualquer chave igual ao valor expressão, calcula as expressões correspondentes a else.

E agora o procedimento flor, com case em vez de cond.

```
(define flor
  (lambda (nome tempo-vida cor)
    ;
    (let ((n nome)
          (tv tempo-vida)
          (c cor))
      (lambda msg
        (case (car msg)
          ; --- recursos próprios ---
          ((identifica) n)
          ((duracao) tv)
          ((cor) c)
          ; - mensagem desconhecida nesta classe -
          (else
            (display (car msg))
            (display ": mensagem desconhecida!...")
            (newline)))))))
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

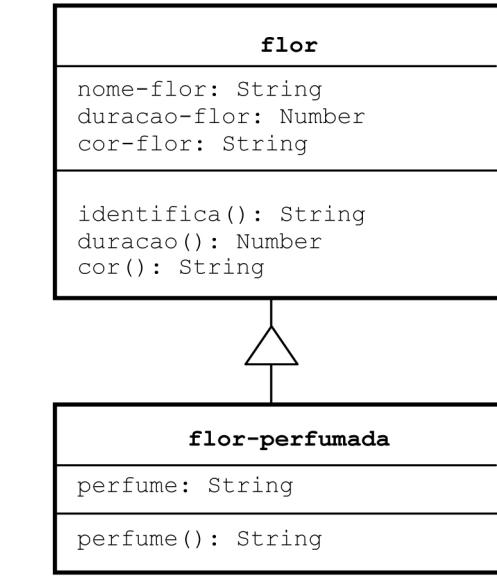


## Classes com heranças

No caso simples da classe flor, seria natural imaginar outras características para além das já consideradas. Imagine, por exemplo, o perfume das flores, a sua fragrância eventualmente definida pelos adjetivos suave, doce e intenso, entre outros.

Esta ou outras novas características poderão ser incluídas sem exigir qualquer alteração à classe flor. Preferiu-se aproveitar o que já está desenvolvido e testado e avançar para a definição de uma nova classe que herde recursos da classe flor e que apareça com outros atributos e operações específicos.

Na figura, a classe flor é a classe base e a classe flor-perfumada é uma classe derivada, pois herda recursos daquela.



Procure descobrir no procedimento flor-perfumada a forma como o Scheme implementa a herança.

```
(define flor-perfumada
  (lambda (nome tempo-vida cor perfume)
    ;
    (let ((flor-base (flor nome tempo-vida cor)))
      (p perfume))
    (lambda msg
      (case (car msg)
        ; --- recursos próprios ---
        ((perfume) p)
        ;
        ; --- recursos herdados ---
        (else (apply flor-base msg)))))))
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Da definição da classe flor-perfumada destaca-se a criação do objecto flor-base, uma instância da classe flor.

Todas as mensagens que cheguem a um objecto da classe flor-perfumada, diferentes de perfume, serão redireccionadas para o objecto flor-base, através de (apply flor-base msg).



Neste exemplo, msg é apenas constituída por um elemento e poderia escrever simplesmente

(flor-base (car msg))

em vez de

(apply flor-base msg).

No entanto, mais à frente, encontrará exemplos em que msg contém vários elementos, onde se justifica plenamente a solução utilizada, pelo que se decidiu desde já adoptá-la.

Não esquecer que (apply flor-base msg) é equivalente a chamar o procedimento flor-base, em que os argumentos são todos os elementos da lista msg, o que não deve confundido com (flor-base msg)...

Podemos assim dizer que a classe flor-perfumada herda métodos da classe flor ou passa mensagens para essa classe. A classe flor funciona como classe base ou ascendente da classe flor-perfumada ou, por outras palavras, a classe flor-perfumada é classe derivada ou descendente da classe flor.

```
> (define f-cravo (flor-perfumada 'cravo 15 'branco 'doce))
> (f-cravo 'perfume)
doce
> (f-cravo 'duracao)
15
> (f-cravo 'identifica)
cravo
> (f-cravo 'cor)
branco
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Alguns atributos foram especificados como símbolos, e não como cadeias de caracteres, mas tudo correu bem na mesma.



Teste a classe flor-perfumada. Crie objectos do tipo flor-perfumada, envie-lhes mensagens e observe como **eles respondem...**

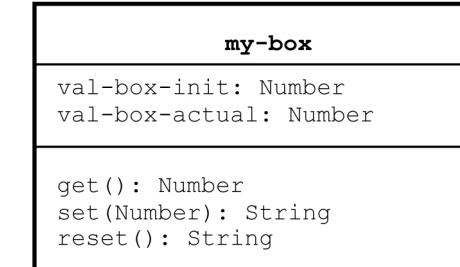
### Classe de objectos com história

Os objectos podem surgir com uma história que é necessário preservar, como acontece no exemplo típico da conta bancária. Neste caso, é fundamental manter actualizado o saldo da conta, pelo menos, e este saldo constituirá a sua história, bastante simplificada é certo, mas que é necessário manter cuidadosamente...

Antes de atacar o problema da conta bancária, começemos por tratar o caso genérico da classe que cria objectos do tipo caixa, que iremos designar por `my-box`. Os objectos da classe `my-box` contêm dois atributos do tipo numérico designados por `val-box-init` (valor inicial) e `val-box-actual` (valor actual), os quais, no momento de criação de um objecto, são inicializados com um mesmo valor, que é dado.

Os objectos da classe `my-box` têm as seguintes operações:

- `get` não tem parâmetros e devolve o conteúdo de `val-box-actual`;
- `set` tem um parâmetro numérico e actualiza `val-box-actual` com esse valor. No final, devolve o símbolo `ok`.
- `reset` não tem parâmetros e repõe em `val-box-actual` o valor inicial, conteúdo guardado em `val-box-init`. No final, devolve o símbolo `ok`.





### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



As mensagens que chegam aos objectos do tipo `my-box` não têm um comprimento fixo. No caso de `get` e `reset` o comprimento é um, apenas o símbolo que representa a mensagem. Mas no caso de `set`, o comprimento é dois, correspondente ao símbolo da mensagem e ao valor numérico da actualização.

Recorde que são as classes cujas mensagens não apresentam um comprimento fixo que exigem objectos definidos através de procedimento com um número não fixo de parâmetros...

```
(define my-box
  (lambda (init-val)
    ;
    (let ((val-box-init init-val)
          (val-box-actual init-val))
      (lambda msg
        (case (car msg)
          ; ----- métodos da classe -----
          ((get) val-box-actual)
          ((set)
           (set! val-box-actual (cadr msg))
           'ok)
          ((reset)
           (set! val-box-actual val-box-init)
           'ok)
          ; - mensagem desconhecida nesta classe -
          (else
            (display (car msg))
            (display ": undefined message!..."))
            (newline))))))

> (define b (my-box 6))
> (b 'get)
6
> (b 'set 56)
ok
> (b 'setttt 6)
setttt: undefined message!...
> (b 'get)
56
> (b 'reset)
ok
> (b 'get)
6
```



Teste a classe `my-box`. Crie objectos do tipo `my-box`, envie-lhes mensagens e observe como eles respondem...

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Neste momento, já terá percebido um outro aspecto importante das classes, que é a protecção dos atributos associados aos objectos. Nos objectos da classe `my-box`, os seus atributos guardados em `val-box-actual` e em `val-box-init` só são acessíveis do exterior por `get`, `set` e `reset`. Os atributos dos objectos encontram-se perfeitamente controlados no que diz respeito a acessos vindos do exterior. Dizendo-se que os atributos estão encapsulados...

A partir da classe `my-box` pode criar objectos cuja história possa ser representada por um dado simples, normalmente um número, um booleano ou um símbolo, que poderá ser, lido e actualizado em qualquer momento, tendo ainda a possibilidade de repôr o seu valor inicial.

Talvez não tenha um grande interesse, mas não fica impedido de colocar, num objecto deste tipo, um dado composto qualquer, como por exemplo, um vector, uma lista, ou uma cadeia de caracteres, mas com uma limitação que é óbvia: esse dado composto, como objecto `my-box`, será tratado como um dado único, não permitindo o acesso individualizado a cada um dos seus elementos. O dado composto entra (`set` ou `reset`) ou sai (`get`) como um todo e o acesso individualizado só poderá ser feito fora...

#### **Exemplo 1 - Conta bancária baseada num objecto `my-box`**

Os objectos da classe `conta-bancaria` têm como atributos um código (uma cadeia de caracteres ou um símbolo) e herdam recursos da classe `my-box`. As operações previstas para esta nova classe são as seguintes:

- `id`: devolve o código associado à conta;
- `saldo`: devolve o saldo actual da conta;
- `levanta-50`: retira 50 ao saldo e devolve-o actualizado;

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

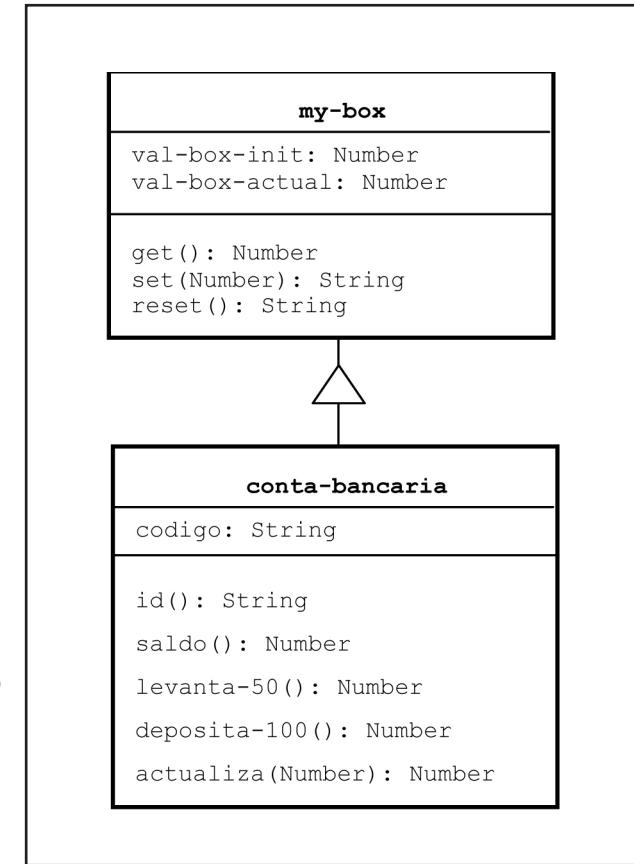
### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

- deposita-100: adiciona 100 ao saldo e devolve-o actualizado;
- actualiza: actualiza o saldo com um montante especificado e devolve-o actualizado.

```
(define conta-bancaria
  (lambda (codigo saldo)
    ;
    (let ((saldo-conta (my-box saldo))
          (cod codigo))
      (lambda msg
        (case (car msg)
          ; --- recursos próprios ---
          ((id) cod)
          ((saldo) (saldo-conta 'get))
          ((levanta-50)
           (saldo-conta 'set (- (saldo-conta 'get) 50))
           (saldo-conta 'get))
          ((deposita-100)
           (saldo-conta 'set (+ (saldo-conta 'get) 100))
           (saldo-conta 'get))
          ((actualiza)
           (saldo-conta 'set (cadr msg))
           (saldo-conta 'get))
          ; --- recursos herdados ---
          ;
          (else (apply saldo-conta msg)))))))
```

```
> (define c (conta-bancaria 'abc567rtp 1000))
> (c 'id)
abc567rtp
> (c 'saldo)
1000
> (c 'levanta-50)
950
> (c 'deposita-100)
1050
> (c 'actualiza 560)
560
> (c 'saldo)
560
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (c 'saaaldo)
saaaldo: undefined message!...
>
```

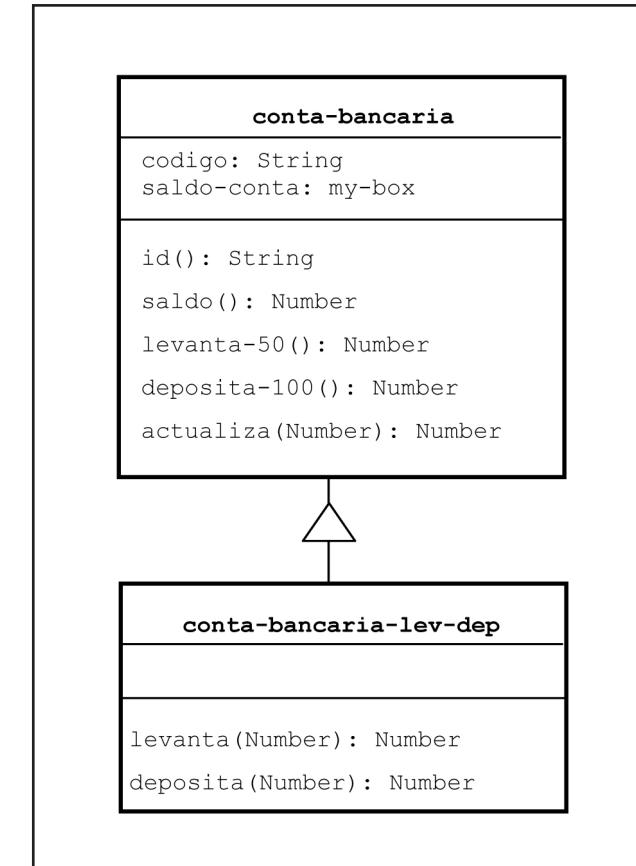


Teste a classe conta-bancaria.

### Exemplo 2 - Conta bancária baseada num objecto de outra conta bancária!

Neste exemplo, define-se a nova classe `conta-bancaria-lev-dep` a partir de outra já existente, a classe `conta-bancaria`, e definem-se também novas operações: `levanta` e `deposita` que funcionam com quantias não fixas, contrariamente ao que acontecia com `levanta-50` e `deposita-100`, da classe base.

```
(define conta-bancaria-lev-dep
  (lambda (ident saldo)
    ;
    (let ((conta-base (conta-bancaria ident saldo)))
      (lambda msg
        (case (car msg)
          ; --- recursos próprios ---
          ((levanta)
           (conta-base 'actualiza
                      (- (conta-base 'saldo) (cadr msg)))
           (conta-base 'saldo))
          ((deposita)
           (conta-base 'actualiza
                      (+ (conta-base 'saldo) (cadr msg)))
           (conta-base 'saldo))
          ; --- recursos herdados ---
          ;
          (else (apply conta-base msg)))))))
```



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
> (define c (conta-bancaria-lev-dep '1abc345g 1500))
> (c 'saldo)
1500
> (c 'id)
1abc345g
> (c 'levanta 200)
1300
> (c 'levanta-50)
1250
> (c 'deposita 50)
1300
> (c 'saldo)
1300
>
```



Teste a classe conta-bancaria-lev-dep.

### Exemplo 3 - uma classe de objectos que sabem contar

A classe de objectos designada por contador cria objectos com características de contador, inicialmente com o valor 0.

Os objectos desta classe têm as seguintes operações:

- inc: incrementa o conteúdo do contador de uma unidade, cujo valor é, seguidamente, devolvido;
- get: devolve o conteúdo do contador;
- load: carrega o contador com um valor fornecido e, seguidamente, devolve o respectivo conteúdo.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

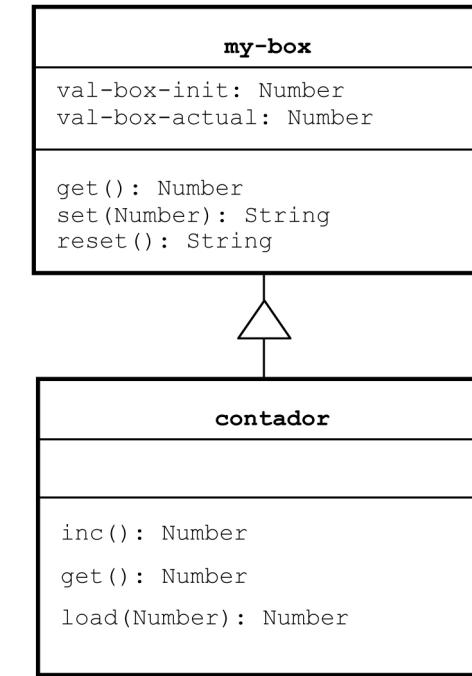
## ANEXOS



A classe contador deriva da classe my-box.

```
(define contador
  (lambda ()
    ;
    (let ((conta (my-box 0)))
      (lambda msg
        (case (car msg)
          ; ---- recursos proprios ----
          ((inc)
            (conta 'set (add1 (conta 'get)))
            (conta 'get))
          ((load)
            (conta 'set (cadr msg))
            (conta 'get))
          ((get)
            (conta 'get))
          ; --- recursos herdados ---
          ;
          (else (apply conta msg)))))))
```

```
> (define cnt (contador))
> (cnt 'inc)
1
> (cnt 'inc)
2
> (cnt 'get)
2
> (cnt 'load 45)
45
> (cnt 'inc)
46
> (cnt 'get)
46
>
```



Teste a classe contador.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 1 - uma classe de objectos que sabem contar nos dois sentidos

Escreva em Scheme a classe contador-completo que herda todas as operações da classe contador e tem como operações próprias:

- dec: decrementa o contador de uma unidade;
- reset: carrega o contador com 0;
- inc-n: incrementa o contador de n unidades;
- dec-n: decrementa o contador de n unidades.

Todas estas operações terminam com a devolução do valor do conteúdo do contador já actualizado.

```
> (define cnt-cmp (contador-completo))
> (cnt-cmp 'inc)
1
> (cnt-cmp 'inc)
2
> (cnt-cmp 'load 56)
56
> (cnt-cmp 'inc)
57
> (cnt-cmp 'dec)
56
> (cnt-cmp 'dec-n 5)
51
> (cnt-cmp 'dec-n 30)
21
> (cnt-cmp 'inc)
22
> (cnt-cmp 'reset)
0
> (c2 'dec)
-1
```



Desenvolva e teste a classe contador-completo.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Classe de objectos com uma história um pouco mais complicada

A história dos objectos nem sempre se representa com dados simples, como acontecia nos exemplos das contas bancárias ou dos contadores. Para as contas bancárias, objectos da classe `my-box` chegavam perfeitamente para guardar o saldo. Mas se pretender representar uma fila de espera, não optará certamente por um objecto do tipo `my-box`...

Introduz-se agora a classe `my-collection`, para criar objectos associados a dados compostos, cuja estrutura possa crescer ou diminuir com o tempo. Um objecto da classe `my-collection` pode crescer com novos elementos, colocados na parte da frente ou na parte de trás, ou diminuir retirando-lhe o elemento da frente ou de trás.

Os métodos a considerar vão ser os seguintes:

- `empty`: devolve `#t`, se colecção vazia, caso contrário, devolve `#f`;
- `push-front`: coloca um novo elemento na frente. Devolve o símbolo `ok`;
- `push-back`: coloca um novo elemento na parte de trás. Devolve o símbolo `ok`;
- `get`: devolve uma lista com o conteúdo de toda a colecção;
- `get-front`: devolve o elemento da frente e deixa a colecção intacta;
- `get-back`: devolve o elemento de trás e deixa a colecção intacta;
- `pop-front`: retira o elemento da frente. Devolve o símbolo `ok`;
- `pop-back`: retira o elemento de trás. Devolve o símbolo `ok`;

Pode assim imaginar uma sessão com objectos criados por instanciação da classe `my-collection`.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

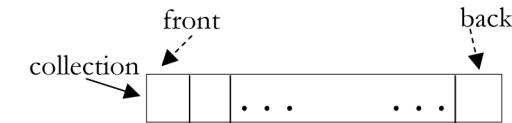
### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (define coll (my-collection))
> (coll 'push-front 6)
ok
> (coll 'push-front 1)
ok
> (coll 'get)
(1 6)
> (coll 'push-back 8)
ok
> (coll 'get)
(1 6 8)
> (coll 'empty)
#f
> (coll 'push-back 12)
ok
> (coll 'get-front)
1
> (coll 'get)
(1 6 8 12)
> (coll 'pop-front)
ok
> (coll 'get)
(6 8 12)
> (coll 'pop-back)
ok
> (coll 'get)
(6 8)
> (coll 'pop-back)
ok
> (coll 'pop-back)
ok
> (coll 'pop-back)
empty collection!...
> (coll 'get)
()
>
```



my-collection
local-coll: List
empty(): Boolean
push-front(Elem): String
push-back(Elem): String
get(): List
get-front(): Elem
get-back(): Elem
pop-front(): string
pop-back(): String

A implementação que se segue recorre a uma lista mutável, com a cabeça `coll`. Fica assim garantido que, mesmo na ausência de qualquer elemento, ou seja, quando a coleção está vazia, que a sua representação é ainda uma lista não vazia, pois terá sempre o elemento cabeça, anteriormente indicado.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
(define my-collection
  (lambda ()
    ;
    (let ((local-coll (list 'coll)))
      (lambda msg
        (case (car msg)
          ; ---- recursos proprios ----
          ((empty) (null? (cdr local-coll)))
          ((push-front)
            (set-cdr! local-coll
              (cons (cadr msg)
                (cdr local-coll)))
            'ok)
          ((push-back)
            (append! local-coll
              (list (cadr msg))))
            'ok)
          ((get) (cdr local-coll))
          ((get-front)
            (if (null? (cdr local-coll))
              (display "empty collection!...")
              (cadr local-coll)))
          ((get-back)
            (if (null? (cdr local-coll))
              (display "empty collection!...")
              (car (reverse local-coll))))
          ((pop-front)
            (if (null? (cdr local-coll))
              (display "empty collection!...")
              (begin
                (set-cdr! local-coll
                  (cddr local-coll))
                'ok)))
          ((pop-back)
            (if (null? (cdr local-coll))
              (display "empty collection!...")
              (begin
                (set-cdr! local-coll
                  (reverse
                    (cdr (reverse (cdr local-coll))))))
                'ok))))
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
; ----- undefined message!... -----
(else
  (display (car msg))
  (display ": undefined message!...")
  (newline))))))
```



Teste a classe my-collection.

### Classe my-stack - um exemplo baseado na classe my-collection

A classe my-collection vai servir de base à classe my-stack (pilha).

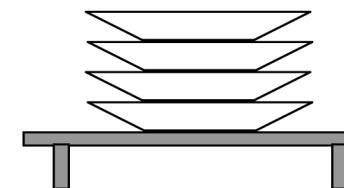
Um objecto criado por instânciação desta classe é composto por uma coleção ordenada de elementos, em que um novo elemento é colocado antes do elemento mais recente da pilha, o topo da pilha, passando ele a ser o novo topo da pilha. O primeiro elemento a sair é o último elemento colocado na pilha, ou seja, o topo da pilha, razão pela qual se justifica a designação LIFO (*last-in-first-out*) para esta estrutura de dados.

Pode fazer a analogia da pilha informática com a pilha de pratos que vai crescendo em cima da mesa da cozinha. Também nesta, o elemento mais facilmente acessível é o que se encontra no topo da pilha dos pratos, ou seja, o último prato que lá foi colocado.

As operações que vão ser associadas à classe my-stack vão ser as seguintes:

- push: coloca um novo elemento no topo da pilha e devolve o símbolo ok;
- pop: retira o elemento do topo da pilha e devolve o símbolo ok;
- top: devolve o elemento do topo da pilha, mantendo-a intacta;

o topo da pilha



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

- empty: devolve #t, se pilha vazia, se não devolve #f;
- visu: visualiza o conteúdo da pilha (operação normalmente apenas para efeitos de teste).

Para clarificar o funcionamento da classe my-stack, siga os exemplos de utilização.

```
> (define stk (my-stack))
> (stk 'visu)
topo:
> (stk 'push 54)
ok
> (stk 'push 35)
ok
> (stk 'visu)
topo: 35 54
> (stk 'pop)
ok
> (stk 'visu)
topo: 54
> (stk 'pop)
ok
> (stk 'empty)
#t
> (stk 'pop)
empty collection!...
>
```



**Na implementação da classe my-stack, verifique como foi bem aproveitado o código da classe my-collection...**

```
(define my-stack
  (lambda ()
    ;
    (let ((local-stack (my-collection)))
      (lambda msg
        (case (car msg)
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
; ---- recursos proprios ----
((push)
  (local-stack 'push-front (cadr msg)))
((pop)
  (local-stack 'pop-front))
((top)
  (local-stack 'get-front))
((empty)
  (local-stack 'empty))
((visu)
  (display "topo: "))
  (for-each (lambda (x)
    (display x)
    (display " ")))
  (local-stack 'get))
  (newline))
; --- recursos herdados ---
;
  (else (apply local-stack msg))))))
```



Teste a classe my-stack.

## Exercício 2

Recorrendo à classe my-stack, escreva a classe my-stack-plus que, para além dos métodos daquela classe, tem um método adicional, designado por `length`, que devolve o número de elementos contidos na pilha.



Desenvolva e teste a classe my-stack-plus.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



### Exercício 3

Escreva a classe `contador-de-n-a-m` que cria objectos contadores, cuja gama de contagem vai de  $n$  a  $m$  (em que  $n < m$ ). Os valores  $n$  e  $m$  são os argumentos a fornecer, na criação de objectos desta classe. Os objectos são inicializados com  $n$ .

As operações consideradas nesta classe são:

- `inc`: incrementa o contador de uma unidade e devolve o conteúdo actualizado. Nesta operação, se atingir  $m+1$ , passa-se automaticamente para  $n$ ;
- `dec`: decrementa o contador de uma unidade e devolve o conteúdo actualizado. Nesta operação, se atingir  $n-1$ , passa-se automaticamente para  $m$ ;
- `load`: carrega o contador com um valor fornecido e devolve o conteúdo actualizado. Nesta operação, se aquele valor se situar fora da gama de contagem, o contador é carregado com  $n$ ;
- `get`: devolve o conteúdo do contador.

```
> (define conta-entre-3-e-15 (contador-de-n-a-m 3 15))
> (conta-entre-3-e-15 'inc)
4
> (conta-entre-3-e-15 'dec)
3
> (conta-entre-3-e-15 'dec)
15
> (conta-entre-3-e-15 'dec)
14
> (conta-entre-3-e-15 'inc)
15
> (conta-entre-3-e-15 'inc)
3
> (conta-entre-3-e-15 'load 23)
3
> (conta-entre-3-e-15 'load 10)
10
> (conta-entre-3-e-15 'inc)
11
>
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Desenvolva e teste a classe contador-de-n-a-m.

### Exercício 4

Recorrendo à classe my-collection, escreva a classe fila.

Um objecto criado por instânciação da classe fila é composto por uma coleção ordenada de elementos, em que um novo elemento é colocado a seguir ao último elemento, passando o novo elemento a ser o último. O primeiro elemento a sair é o primeiro elemento da fila, razão pela qual se justifica a designação FIFO *First-In-First-Out* normalmente associada a esta estrutura de dados.

As operações desta classe fila são agora apresentadas:

- poe-na-fila (*enqueue*): coloca um novo elemento no fim da fila. Devolve o símbolo ok;
- tira-da-fila (*dequeue*): tira o primeiro elemento da fila. Devolve o símbolo ok;
- fila-vazia: devolve #t, se fila vazia, se não devolve #f;
- frente-da-fila: devolve o valor do primeiro elemento da fila, mantendo a fila intacta
- mostra-fila: visualiza o conteúdo da fila.

Segue-se um exemplo de utilização deste tipo de objecto.

```
> (define cantina (fila))
> (cantina 'mostra-fila)
frente:
> (cantina 'poe-na-fila 'jose)
ok
> (cantina 'poe-na-fila 'maria)
ok
> (cantina 'poe-na-fila 'manuel)
ok
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (cantina 'mostra-fila)
frente: jose maria manuel
> (cantina 'frente-da-fila)
jose
> (cantina 'mostra-fila)
frente: jose maria manuel
> (cantina 'tira-da-fila)
ok
> (cantina 'mostra-fila)
frente: maria manuel
>
```



Desenvolva e teste a classe fila.



As classes apresentadas (`my-box`, `contador`, `my-collection` e `my-stack`) estão disponíveis no DrScheme, em `PLT\collects\`, fazendo (`(require (lib "box-cont-coll-stack.scm" "user-feup"))`)



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



Uma das vantagens da programação OO reside na modularidade que oferece, pois cada objecto surge como um bloco, que protege ou encapsula os seus atributos, condicionando o respectivo acesso às suas operações preparadas para esse efeito.

Tomando o exemplo da classe `my-collection`, tente identificar as operações em que esta importante característica da programação OO pode ser violada.

Consegue apresentar uma solução para essas operações?

### Construtor de cópia

Imagine que, numa certa situação, tem um objecto e pretende apenas simular um processamento sobre ele, exigindo, mais tarde, a reposição do seu valor. Numa situação deste tipo, parece vantajosa a possibilidade de criar uma cópia daquele objecto.

Como criar essa cópia?

Em primeiro lugar, veja com atenção o diálogo que se segue.

```
> (define obj1 157)
> (define copia-obj1 obj1)
> copia-obj1
157
> (set! copia-obj1 12)
> copia-obj1
12
> obj1
157
>
```

Ao trabalhar com objectos simples, a cópia surge, naturalmente, como um objecto independente do original, o que já não aconteceria, por exemplo, se o caso envolvesse uma lista.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (define obj10 (list 10 11 12))
> (define copia-obj10 obj10)
> (set-car! copia-obj10 55)
> copia-obj10
(55 11 12)
> obj10
(55 11 12)
```

Os objectos `obj10` e `copia-obj10` não são independentes e as alterações feitas a um reflectem-se no outro. De facto, `obj10` e `copia-obj10` são apontadores para a mesma lista...

Uma forma de ultrapassar o problema seria o procedimento construtor `copia-lista`.

```
(define copia-lista
  (lambda (lis)
    (if (null? lis)
        '()
        (cons (car lis)
              (copia-lista (cdr lis))))))
```

```
> (define obj10 (list 10 11 12))
> (define copia-obj10 (copia-lista obj10))
> (set-car! copia-obj10 55)
> copia-obj10
(55 11 12)
> obj10
(10 11 12)
>
```



Em vez do procedimento construtor `copia-lista`, consegue uma solução muito mais compacta com

`(define copia-obj10 (apply list obj10))`

Concorda?



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Os objectos `obj10` e `copia-obj10` surgem agora como independentes. No entanto, o ideal seria que o construtor `list` tivesse um construtor de cópia que, dada uma lista, forneceria uma cópia independente da lista dada.

```
> (define obj10 (list 10 11 12))
> (define copia-obj10 (list obj10))
> copia-obj10
((10 11 12))
>
```

Sabemos que isto não faz o que desejaríamos! Neste exemplo, `list` devolve uma lista, mas apenas com um elemento, que é a lista dada como argumento...

Na versão que agora se apresenta da classe `my-box`, aparecem duas novidades:

- uma nova operação ou método com a designação `type`, que todas as classes deveriam ter.
- o construtor admite um argumento que pode ser, normalmente, um dado simples (número, booleano ou símbolo), mas, agora, também um objecto do tipo `my-box`. Neste último caso, funcionará como construtor de cópia, devolvendo uma cópia independente do objecto `my-box`.

```
(define my-box
  (lambda (init-val)
    ;
    (let ((i-val
           (if (and (procedure? init-val) ; é procedimento? Objecto = procedimento
                   (string=? "my-box" (init-val 'type))) ; é obj. my-box?
               (init-val 'get)
               init-val))
        ;
        (let ((val-box-init i-val) ; i-val é o argumento ou o valor de um objecto my-box
              (val-box-actual i-val))
            (lambda msg
              (case (car msg)
                ; ----- metodos da classe -----
                ((type) "my-box")
                ((get) val-box-actual)
                ((set)
                  (set! val-box-actual (cadr msg))
                  'ok)
                ((reset)
                  (set! val-box-actual val-box-init)
                  'ok)
                ; - mensagem desconhecida nesta classe -
                (else
                  (display (car msg))
                  (display ": undefined message!..."))
                  (newline)))))))

```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Se ficou com dúvidas sobre o funcionamento do construtor cópia, veja o que acontece no diálogo que se segue, com dois objectos da classe my-box.

```
> (define mb (my-box 45))
> (define mbb (my-box mb))
> (mbb 'get)
45
> (mbb 'set 500)
ok
> (mbb 'get)
500
> (mb 'get)
45
>
```



Teste a nova versão da classe my-box, agora com construtor de cópia.

### Exercício 5 - a classe contador com construtor de cópia

Incluir na classe contador um construtor de cópia.



Desenvolva e teste a nova versão da classe contador, agora com construtor de cópia.

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



### Construtor de cópia na classe my-collection

Como desafio, fica a inclusão de um construtor de cópia na classe my-collection.

O problema não é tão simples como no caso da classe my-box, pois agora encontrará uma lista como base desses objectos...

E se quiser levar o problema até ao fundo, numa segunda volta, verifique que não chega usar uma solução do tipo do procedimento copia-lista, pois a lista poderá ter como elementos outras listas...



Desenvolva e teste a nova versão da classe my-collection, agora com construtor de cópia.

### Projecto 6.1- Jogo de dados

Defina e implemente em Scheme a classe jogo-de-dados, que origina objectos que sabem jogar dados de acordo com as regras definidas mais à frente, e que contempla ainda as seguintes operações:

- apostar: simula um lançamento de 3 dados, visualiza as faces que ficaram voltadas para cima, verifica se o jogador ganhou ou perdeu de acordo com as regras, calcula e visualiza o respectivo montante. Finalmente, actualiza e visualiza o saldo do jogo;
- saldo: devolve o valor do saldo do jogo (pode ser negativo);
- termina: termina o jogo, com a visualização do saldo.

Seguem-se as regras para o cálculo do montante que se ganha ou se perde em cada aposta. Em cada aposta é definido o valor que se arrisca.

- 3 faces iguais: ganha  $72 \times$  valor da aposta;



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

- 2 faces iguais: ganha 36 x valor da aposta;
- 3 faces diferentes e a soma delas é menor que ou igual a 11: ganha soma das faces x valor da aposta;
- 3 faces diferentes e a soma delas é maior que 11: perde soma das faces x valor da aposta.

Para um esclarecimento mais completo do jogo, segue-se um pequeno diálogo em que são criados dois jogos, com alguns comentários ao lado.

```
> (define jogo-1 (jogo-de-dados)) (cria um jogo designado por jogo-1)
> (jogo-1 'saldo) (verifica o saldo de jogo-1)
  saldo: 0
> (jogo-1 'aposta 20) (faz uma aposta de 20 com jogo-1)
  dado 1: 4
  dado 2: 3
  dado 3: 4
  ...
  ganhaste: 36 x 20 = 720 (teve sorte, ganhou 720 pontos)
  saldo: 720

> (jogo-1 'aposta 100) (faz uma aposta de 100 com jogo-1)
  dado 1: 6
  dado 2: 3
  dado 3: 4
  ...
  ...
  perdeste: 13 x 100 = 1300 (teve azar, perdeu 1300 pontos)
  saldo: -580

> (jogo-1 'saldo)
-580

> (define jogo-2 (jogo-de-dados)) (cria novo jogo, jogo-2)
> (jogo-2 'aposta 50) (faz uma aposta de 50 com jogo-2)
  ...
  ...

> (jogo-1 'aposta 200) (é possível retomar o jogo-1...)
  dado 1: 1
  dado 2: 3
  dado 3: 4
  ganhaste: 8 x 200 = 1600
  saldo: 1020

> (jogo-1 'termina)
  O jogo vai terminar
  com o saldo: 1020 (o jogo-1 terminou...)
  ...

> (jogo-2 'aposta 150) (... mas o jogo-2 continua disponível)
  ...
```





Desenvolva e teste a classe jogo-de-dados.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



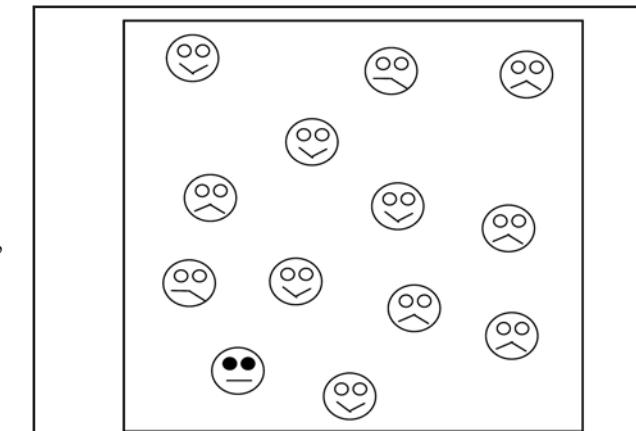
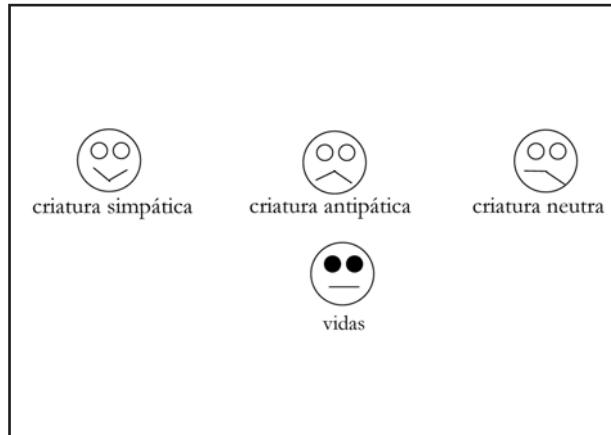
### Projecto 6.2- o-vidas

- projecto de dificuldade elevada

Uma criatura simples, que dá pelo nome de o-vidas, tem alguns problemas de visão. Por isso, o-vidas, desde que nasce e até que morre, não faz outra coisa nos seus passeios que não seja esbarrar com obstáculos vários, paredes ou outras criaturas. Entre estas criaturas, algumas são simpáticas e até lhe oferecem prendas (20 pontos). Mas outras, pelo contrário, são antipáticas e acabam por lhe retirar alguma coisa (10 pontos). Ainda há outras, as neutras, que lhe retiram 50% do que tiver amealhado até essa altura, ou seja, se o-vidas tiver amealhado, por exemplo, 30, passa a ter 15, mas se tiver -30 passa para -15.

Todas as criaturas referidas, incluindo o-vidas, são colocadas aleatoriamente numa praceta.

Mas, contrariamente a o-vidas, que não pára apesar dos seus problemas de visão, as outras criaturas depois de colocadas na praceta permanecem fixas. Limitam-se a esperar que o-vidas, nos seus passeios na praceta, se encontre com elas, para o presentear ou para lhe retirar alguma coisa. Na figura, a praceta mostra 5 criaturas simpáticas, 5 antipáticas, 2 neutras e o-vidas.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Como se desloca o-vidas?

Quando a praceta é visualizada com todos os actores, o utilizador é convidado a indicar um ângulo de partida para o início da trajectória de o-vidas. No caso da figura, o ângulo de partida é -45° (ou 315°).

Depois inicia-se o percurso na praceta, sempre em linha recta, até encontrar uma das 4 paredes ou uma das outras criaturas.

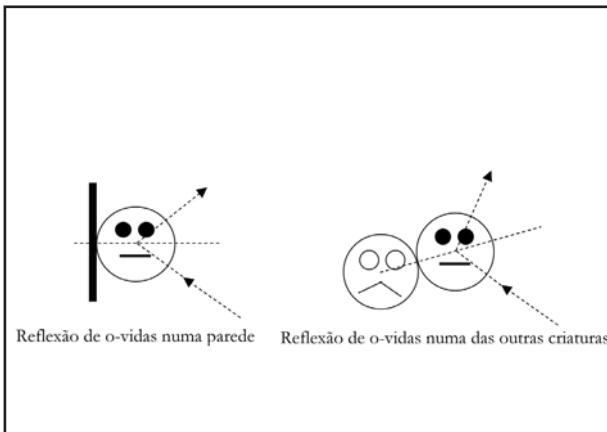
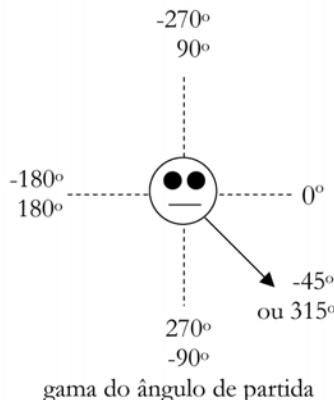
Nesse encontro, a direcção da deslocação sofre uma reflexão.

Quando termina o percurso de o-vidas?

O tempo de vida de o-vidas, especificado em anos, e o comprimento dos pequenos passos que dá são pedidos inicialmente pelo programa. Sempre que esbarra noutra criatura, mas não numa parede, conta como dia de aniversário, e a sua idade avança 1 ano. É por isso que algumas criaturas a presenteiam, se bem que outras, como já se viu, não se importam muito com esse dia festivo, antes pelo contrário. Quando atinge o tempo limite de vida, o percurso de o-vidas termina e é altura de avaliar os haveres que conseguiu amealhar.

Com base na programação orientada por objectos, pretende-se desenvolver um programa para simular o percurso de criaturas da classe vidas. A criatura o-vidas, o herói da história, é uma instância da classe vidas. Sempre que o-vidas esbarra em alguma criatura, é dia de comemorar o aniversário, por isso são actualizados a sua idade (+ 1 ano) e os seus haveres (de acordo com a criatura em que esbarrou), e porque certamente vai beber um pouco mais de champanhe na comemoração, perde o rumo e muda de direcção. Apesar de não contar como dia de aniversário, também muda de direcção quando esbarra numa parede.

Numa janela gráfica são visualizados a praceta com todos os actores, incluindo o-vidas, o percurso que vai fazendo, a sua idade e os seus haveres.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

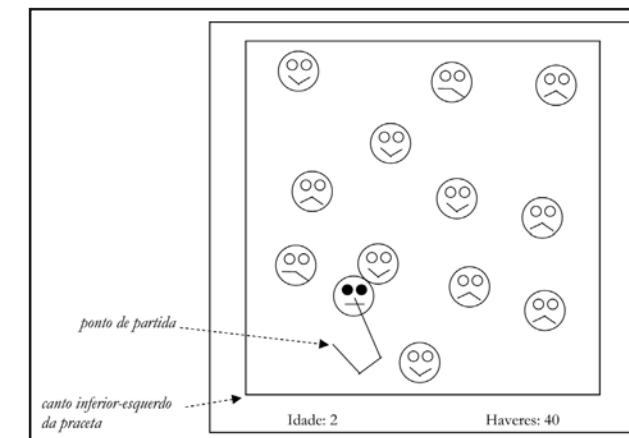
## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

A figura dá uma ideia do que será a visualização ao fim de 2 aniversários. Nota-se, pelo traçado do percurso, que o-vidas iniciou o percurso com um ângulo de, aproximadamente, -45° (equivalente a 315°), com a idade zero e com o saco dos haveres vazio. Começou por esbarrar na parede de baixo e mudou de rumo, encontrando depois uma criatura simpática. Nesse momento, a idade passou para 1 e os haveres aumentaram para 20. Voltou a mudar de rumo e teve a sorte de esbarrar com outra criatura simpática. Passou para 2 anos de idade e o saco dos haveres tem agora 40. É esta a situação representada na figura.



Estruture e escreva em Scheme o programa o-vidas que tem como principal objectivo dar a conhecer qual o montante em haveres com que o-vidas termina o seu percurso. Este programa começa por pedir o tempo de vida do actor principal e o comprimento do seu pequeno passo, normalmente, o valor 1. O programa ainda pede as características da janela gráfica onde decorrerá a simulação do percurso de o-vidas, o raio do círculos que representam as caras e o número de simpáticos, antipáticos e neutros. Depois disto, é visualizada a praceta com todas as criaturas escolhidas e também o-vidas.

O programa acaba por pedir o ângulo de partida e, de imediato, verá no ecrã a praceta onde o-vidas passeia.

```
> (o-vidas)
  características de O-Vidas
  tempo-de-vida largura-do-passo (ex: 30 1): 30 1
  características da janela grafica
  largura altura (ex: 400 350): 400 350

  características da praceta
  raio num-simp num-ant num-neutros (ex: 15 4 3 2): 15 4 3 2

  Angulo de partida: 145
  Fim... do Vidas...
>
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

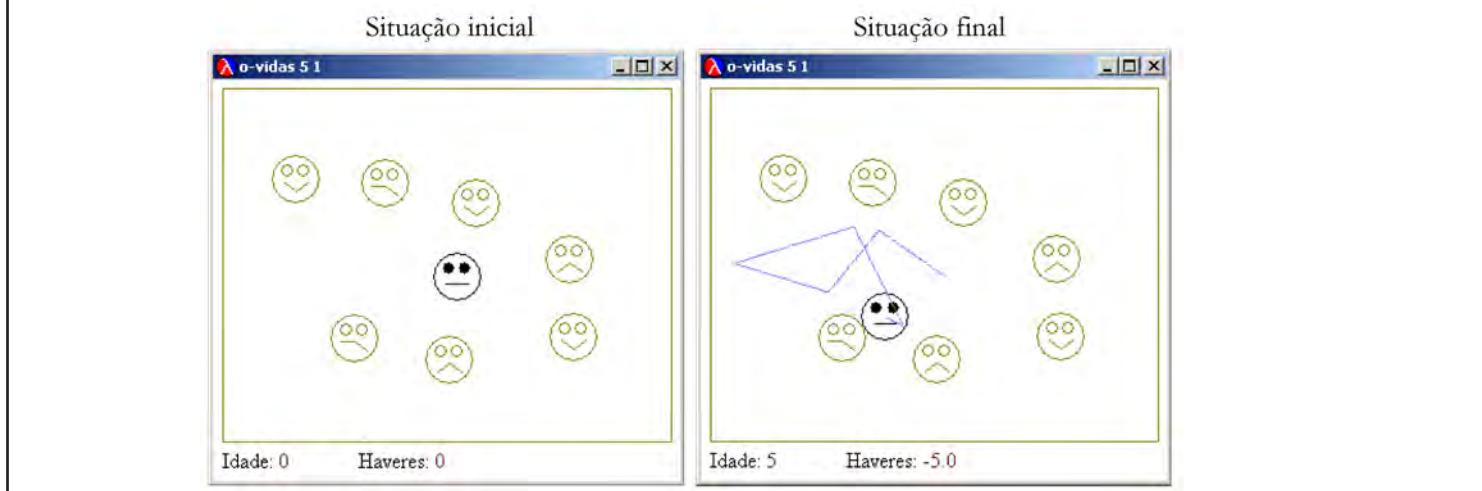
## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Através de uma tabela, apresenta-se a evolução dos atributos de o-vidas desde a situação inicial (figura da esquerda) até à situação final (figura da direita).

acontecimento	idade	haveres
partida	0	0
neutro	1	0
neutro	2	0
parede	2	0
neutro	3	0
antipático	4	-10
neutro	5	-5



Pode experimentar [aqui](#) uma versão executável de o-vidas.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

São agora apresentadas algumas pistas para resolução do problema.

- 1- Analise cuidadosamente o problema exposto e identifique as principais classes a considerar, incluindo as suas operações. Analisar as hipóteses da classe vidas para criar objectos do tipo vidas e da classe praceta para criar objectos do tipo praceta com os respectivos actores, mas excluindo o-vidas. Que operações devem ter?
- 2- Para não atacar inicialmente o problema na sua globalidade, pois já se apresenta com uma razoável complexidade, considere-o por fases.

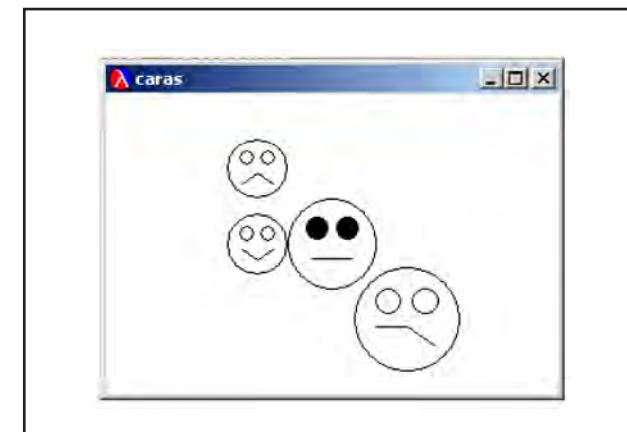
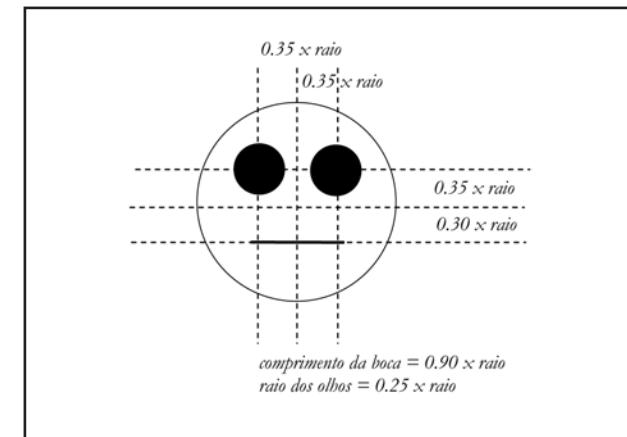
1<sup>a</sup> Fase

Tomando por base as primitivas gráficas apresentadas no **Anexo B**, escreva em Scheme os procedimentos que desenham as caras dos vários actores deste jogo, os procedimentos cara-vidas, cara-simpatico, cara-antipatico e cara-neutro. Os parâmetros poderão ser o centro da cara na janela gráfica e o seu raio. É aconselhável que o posicionamento e dimensão, quer dos olhos quer da boca, sejam calculados em função do raio da cara. Se assim for, a visualização das caras apresentar-se-á sempre proporcionada, independentemente da dimensão do raio. Por exemplo, no caso de o-vidas, o dimensionamento poderia ser como se indica na figura.

```
> (janela 300 200 "caras")
> (cara-simpa (list 100 100) 20)
> (cara-vidas (list 150 100) 30)
> (cara-anti (list 100 150) 20)
> (cara-neutro (list 200 50) 35)
>
```

2<sup>a</sup> Fase

Escreva em Scheme a classe praceta, com os parâmetros origem (o canto inferior-esquerdo da praceta, em relação à origem da janela gráfica), largura (da praceta), altura (da praceta), raio-caras (raio das circunferências que definem as caras), e o número de simpáticos, antipáticos e neutros.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Considere, para já, as operações que permitem seleccionar as principais características dos objectos gerados a partir desta classe e uma variável interna a considerar nestes objectos, que se pode designar por *praceta* (do tipo *my-collection*), que conterá a definição das paredes da *praceta* e a posição dos actores (os simpáticos, os antipáticos e os neutros).

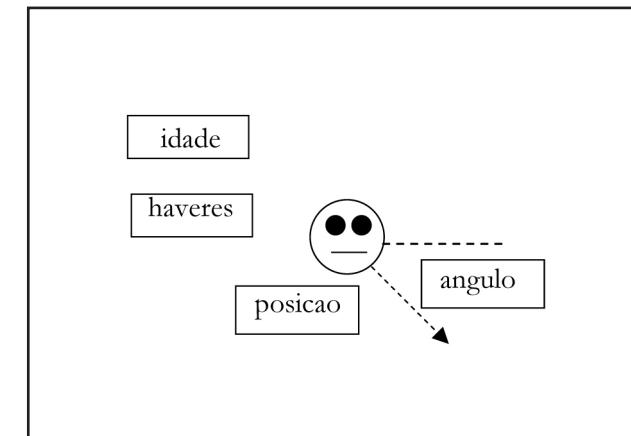
```
> (define praceta-1 (praceta (list 10 30) 250 200 15 2 2 1))
> (praceta-1 'largura)
250
> (praceta-1 'altura)
200
> (praceta-1 'raio)
15
> (praceta-1 'origem)
(10 30)
> (praceta-1 'as-paredes)
((p . 25) (p . 215) (p . 245) (p . 45))
> (praceta-1 'as-criaturas)
((s 112 114) (s 192 79) (a 137 183) (a 73 182) (n 207 130))
> (praceta-1 'perto-das-outras? 110 113)
#t
> (praceta-1 'perto-das-outras? 50 113)
#f
>
```

Verifique, na parte final do diálogo, que uma das operações é *perto-das-outras?*, que devolve um booleano que reflecte a proximidade de um ponto dado às criaturas da *praceta*. Esta operação é muito importante no posicionamento aleatório das criaturas na *praceta*, incluindo *o-vidas*, para evitar que se sobreponham.

3<sup>a</sup> Fase

Escreva em Scheme a classe *vidas*, com os parâmetros *tempo-de-vida* (define a duração do percurso de *o-vidas*), *passo* (define o passo elementar dado por *o-vidas*) e *praca* (*praceta* onde *o-vidas* vai passear), com as operações que encontra no diálogo que se segue.

Nesta altura será importante identificar quais as variáveis internas a considerar nos objectos desta classe, sugerindo-se *idade* (do tipo *contador*), *haveres* (do tipo *my-box*), *angulo* (do tipo *my-box*), *posicao* (do tipo *my-box*), pois não está impedido de associar a este tipo, uma lista de comprimento fixo, neste caso, de 2 elementos, *x* e *y*.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

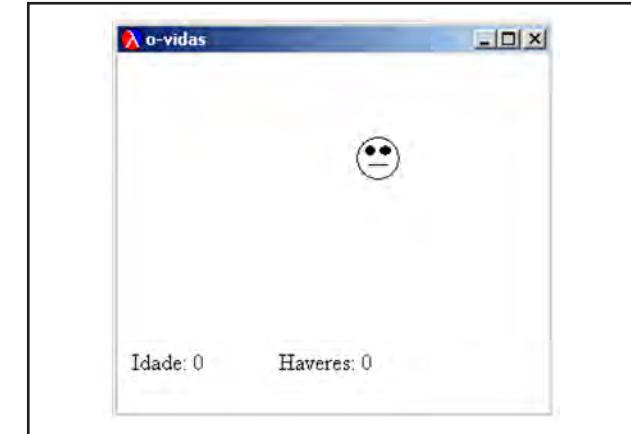
## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
> (janela 300 250 "o-vidas")
> (define praca (praceta (list 10 50) 270 190 15 3 2 1))
> (define o-vidas (vidas 30 1 praca))
> (o-vidas 'posicao)
(181 177)
> (o-vidas 'visu)
> (o-vidas 'visu-idade-haveres)
> (o-vidas 'idade)
0
> (o-vidas 'idade+1!)
1
> (o-vidas 'idade)
1
> (o-vidas 'poe-angulo! 35)
35
> (o-vidas 'angulo)
35
> (o-vidas 'poe-angulo! 0)
0
> (o-vidas 'angulo)
0
> (o-vidas 'andar!)
> (o-vidas 'posicao)
(182 177)
> (o-vidas 'andar!)
> (o-vidas 'posicao)
(183 177)
>
```



Alguns dos pontos deste diálogo merecem uma breve referência, já que a maioria é auto-esclarecedora. A operação `andar!` faz mover `o-vidas` na direcção do ângulo actual e o comprimento da deslocação é função do passo definido para esta criatura. Para se conseguir uma visualização dinâmica de `o-vidas`, deve utilizar cor `temp`. Um objecto visualizado com esta cor "desaparece" ao ser visualizado pela segunda vez, no mesmo sítio, podendo depois ser visualizado mais à frente. Às operações `visu` e `visu-idade-haveres` corresponde, respectivamente, a visualização de `o-vidas` e dos campos correspondentes à idade e aos haveres.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

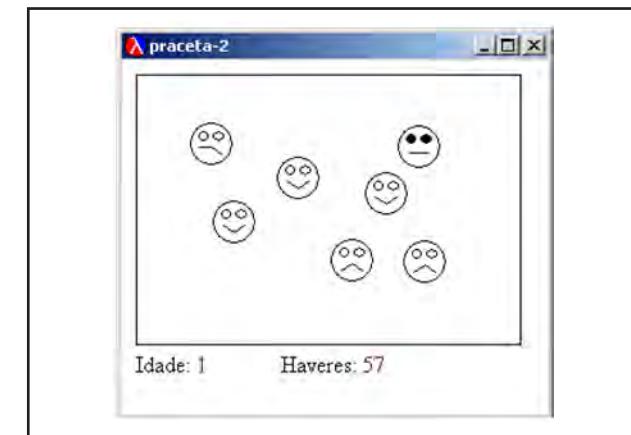
## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

### 4<sup>a</sup> Fase

Escreva a classe `praceta-2`, com os mesmos parâmetros da classe `praceta`, que, para além das operações que herda desta classe, ainda possui a operação `visu`, que visualiza a `praceta` e os vários actores associados, com exceção de `o-vidas`, e a operação `colisao?` com um parâmetro do tipo `vidas` e que devolve `#t` ou `#f`, conforme o objecto `vidas` tenha colidido ou não, no último deslocamento, com qualquer elemento da `praceta` (criatura ou parede). Mais pormenorizadamente, sugere-se que o valor devolvido por esta última operação, caso não haja colisão, seja a lista `(#f)` ou, se ocorrer colisão, a lista `(#t 'p/ 'c novos-haveres novo-angulo)`. O segundo elemento desta lista, ou seja, `'p/ 'c`, permite distinguir se a intersecção se deu com uma parede, `'p`, ou com uma das outras criaturas, `'c`.

```
> (janela 300 250 "praceta-2")
> (define praca (praceta-2 (list 10 50) 270 190 15 3 2 1))
> (define o-vidas (vidas 30 1 praca))
> (o-vidas 'idade+1!)
1
> (o-vidas 'poe-haveres! 57)
57
> (o-vidas 'idade)
1
> (o-vidas 'haveres)
57
> (o-vidas 'angulo)
0
> (o-vidas 'posicao)
(209 189)
> (praca 'visu)
> (o-vidas 'visu)
> (o-vidas 'visu-idade-haveres)
> (praca 'colisao? o-vidas)
(#f)
>
```



### 5<sup>a</sup> Fase

Criar objectos com as classes anteriormente desenvolvidas e testar as várias operações de cada uma delas.

É natural que venha a detectar um problema com os objectos da classe `praceta-2`.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

A operação `colisao?` permite verificar se algum dos elementos de um objecto desta classe se encontra em colisão com o-vidas e, se assim for, procede-se à actualização dos seus atributos de idade, haveres e ângulo. Ora, devido ao passo que o-vidas dá, poderá acontecer que, ao colidir com um elemento da praceta, "entre por ele a dentro"... O caso não seria grave se, no passo seguinte, já depois das actualizações anteriormente citadas, o-vidas deixasse de intersectar o elemento com que colidiu. Mas nem sempre assim acontece, e nova colisão seria identificada, ocorrendo novas actualizações de idade, haveres e ângulo. A confusão seria grande, pois tratar-se-ia da mesma colisão processada mais do que uma vez. É por este motivo que se vai considerar uma variável do tipo `my-box` na classe `praceta-3`, que se pode designar por `em-interseccao`, colocada a `#t` quando o-vidas colide com um elemento da praceta e colocada a `#f` quando deixa de o intersectar. Sendo assim, mesmo que o método `colisao?` devolva (`#t 'p/ 'c novos-haveres novo-angulo`), a situação de o-vidas não se altera se `em-interseccao` for `#t`...

Considerando o que acaba de se apresentar, escreva a classe `praceta-3`, com as mesmas características da classe `praceta-2` e que, para além dos métodos que herda desta classe, ainda possui `interseccao` que devolve o valor da `my-box` designada por `em-interseccao` e `poe-interseccao!` que coloca o argumento `#t` ou `#f` nesta mesma `my-box`.

6<sup>a</sup> Fase

Criar objectos com as classes anteriormente desenvolvidas e testar as várias operações de cada um deles. Introduza as alterações que venha a considerar necessárias.

```
> (define praca-3 (praceta-3 (list 10 30) 300 280 15 3 3 2))
> (praca-3 'poe-interseccao! #t)
#t
> (praca-3 'interseccao)
#t
> (praca-3 'poe-interseccao! #f)
#f
> (praca-3 'interseccao)
#f
> (praca-3 'as-paredes)
((p . 25) (p . 295) (p . 295) (p . 45))
> (praca-3 'as-criaturas)
((s 166 125) (s 59 131) (s 126 263) (a 213 210) (a 136 171) (a 59 82)
 (n 57 242) (n 236 142))
>
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

R 1 2 3 4 5

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### 7<sup>a</sup> Fase

Desenvolva o programa principal `o-vidas`, sem parâmetros, que pede as características de `o-vidas` e também as características da `praceta`, cria um objecto `praceta` e um objecto `vidas`, visualiza `praceta` e logo de seguida `o-vidas`, visualiza os campos da idade e dos haveres, pede o ângulo de partida (em graus, internamente convertido em radianos) e, finalmente, dá ordem a `o-vidas` para fazer o seu percurso.

### 8<sup>a</sup> Fase

Experimente o programa `o-vidas`... e introduza-lhe os melhoramentos para eliminar eventuais problemas que venha a detectar.



Desenvolva e teste o projecto `o-vidas`.

## Conclusão

Neste Módulo fez uma abordagem inicial à programação orientada por objectos utilizando o Scheme. O Scheme, apesar de não ser uma linguagem pensada para este paradigma de programação, permitiu mostrar como podem ser implementados os principais conceitos deste importante paradigma de programação: classes, objectos, atributos, mensagens, operações (ou métodos) e herança.

Ficou claro que os dados de cada objecto deveriam ficar perfeitamente protegidos de acessos vindos do exterior, só sendo admitidos os acessos com origem em operações próprias de cada objecto. Nem sempre assim aconteceu, mas foram-lhe deixadas pistas para ultrapassar este problema. Também verificou, certamente, que é fácil reutilizar classes previamente implementadas na criação de novas classes.

A propósito dos objectos com história, foi introduzida a classe `my-box` e com base nela a classe `contador`. A classe `my-box` está mais vocacionada para guardar dados simples, e foi por isso que se apresentou a classe `my-collection`, esta orientada para dados compostos e dinâmicos, que crescem ou diminuem conforme as necessidades dos programas. Foi com base na classe `my-collection` que se introduziu a classe `stack`, *LIFO*, e que num dos exercícios do final do capítulo foi proposta a implementação da classe `fila`, *FIFO*.

# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

R 1 2 3 4 5

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

A lista de exercícios encerra com dois projectos que deverão ser abordados através da programação OO. Um, mais simples, tem a ver com um jogo de dados, o outro, de complexidade média/alta que, simula, através de uma animação gráfica, o percurso de uma criatura numa praça fechada por quatro paredes, onde se encontram outras criaturas de vários tipos...



## **INTRODUÇÃO**

**1 - O ESSENCIAL DO SCHEME**

**2 - RECURSIVIDADE**

**3 - ABSTRACÇÃO DE DADOS**

**4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE**

**5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS**

**6 - SCHEME E OUTRAS  
TECNOLOGIAS**

**7-EXERCÍCIOS  
E  
PROJECTOS**

**ANEXOS**

## **Exercícios e Projectos**

O conjunto de enunciados de exercícios e projectos que se segue, apesar de alguns indicarem pistas de solução, tenta colocá-lo perante problemas de programação desligados de qualquer contexto ou capítulo. Quando se propõe um exercício no final de um capítulo, uma primeira pista de resolução fica implicitamente estabelecida. Na vida real, os problemas de programação não aparecem rotulados ou associados a capítulos de qualquer livro. O problema surge e torna-se necessário encontrar uma ideia de resolução para ele, vaga no início, mas sucessivamente mais refinada. Só à medida que as várias tarefas e sub-tarefas do problema vão sendo identificadas é que se clarificam as associações com os capítulos e matérias tratadas em cada um deles. Surge então a forma de representar os dados do problema, a abstracção de dados acompanhada de um certo conjunto de construtores, selectores e até modificadores. E é sobre a abstracção idealizada e com as tarefas e sub-tarefas identificadas que se refina a construção da solução.

Tendo em conta o elevado número de exercícios e projectos propostos, recomenda-se o uso do Scheme nas actividades do módulo.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Exercício 1

O programa matriz estabelece um diálogo com o utilizador e, através desse diálogo, constrói uma matriz. Com a matriz construída o diálogo continua, permitindo escolher uma linha ou uma coluna e, seguidamente, somar ou visualizar os elementos da linha ou da coluna escolhida.

```
> (matriz)
Número de linhas: 2
Número de colunas: 3
próximo elemento da matriz: 1
próximo elemento da matriz: 2
próximo elemento da matriz: 3
próximo elemento da matriz: 4
próximo elemento da matriz: 5
próximo elemento da matriz: 6
O que deseja? linha/coluna: linha
Número da linha/coluna: 2
Que pretende? somar/visualizar: somar
15
O que deseja? linha/coluna: coluna
Número da linha/coluna: 2
Que pretende? somar/visualizar: visualizar
2 5
O que deseja? linha/coluna: coluna
Número da linha/coluna: 3
Que pretende? somar/visualizar: somar
9
O que deseja? linha/coluna: linha
Número da linha/coluna: 3
erro...
O que deseja? linha/coluna: linha
Número da linha/coluna: 1
Que pretende? somar/visualizar: somaaaaaaaar
O programa vai terminar...
```

Imagine que o programa referido é implementado em Scheme através do procedimento `matriz`.

```
(define matriz
  (lambda ()
    (let* ((n-lin (le-num 'linhas)) ; leitura nº de linhas
           (n-col (le-num 'colunas)) ; leitura nº de colunas
           (mat (le-matriz (* n-lin n-col)))) ; constrói lista com os
           ; elementos da matriz
           (processa-matriz mat n-lin n-col))))
```



### INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



- 1- Faça uma abordagem de-cima-para-baixo ao problema exposto, identifique as suas tarefas e sub-tarefas principais e defina uma abstracção de dados adequada.
- 2- Escreva em Scheme a abstracção de dados definida.
- 3- Escreva em Scheme o programa matriz, apoiado na abstracção desenvolvida e nas tarefas e sub-tarefas identificadas.



Desenvolva e teste o programa matriz.

### Exercício 2

Numa língua muito estranha, as palavras são constituídas por letras de um abecedário, como acontece normalmente, mas neste caso não se admitem repetições de letras na mesma palavra!

Assim, o comprimento das palavras poderá variar entre 1 e n, em que n representa o número de letras do abecedário. Por exemplo, com um abecedário constituído apenas pelas letras a e b, as palavras possíveis serão a, b, ab e ba, com comprimentos que variam entre 1 e 2. Escreva em Scheme o procedimento comprimento-max-dodicionario com o parâmetro n e que devolve o número máximo de palavras do dicionário de uma língua com um abecedário de n letras.

```
> (comprimento-max-do-dicionario 1)  
1  
> (comprimento-max-do-dicionario 2)  
4
```



Desenvolva e teste o programa comprimento-max-do-dicionario.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

## Exercício 3

Um programa designado por teclado suporta uma interacção como se indica. O programa começa por interrogar o utilizador sobre a operação aritmética que pretende executar (so - somar; su - subtrair; mu - multiplicar; di - dividir). Caso o utilizador opte por uma operação fora deste elenco, o programa voltará a interrogá-lo sobre a operação a executar. Depois de identificada a operação, o programa interroga o utilizador sobre os valores que pretende operar. A série de valores a operar deve terminar com zero, que funciona apenas como marca e não como valor a operar.

```
> (teclado)
operacao (so su mu di): so
numero (zero para terminar): 3
numero (zero para terminar): 4
numero (zero para terminar): 0
resultado: 7
Terminou o programa

> (teclado)
operacao (so su mu di): di
numero (zero para terminar): 5
numero (zero para terminar): 1
numero (zero para terminar): 2
numero (zero para terminar): 0
resultado: 2.5
Terminou o programa

> (teclado)
operacao (so su mu di): s
operacao (so su mu di): su
numero (zero para terminar): 4
numero (zero para terminar): 56
numero (zero para terminar): 6
numero (zero para terminar): 0
resultado: -58
Terminou o programa
```

- 1- Faça uma abordagem de-cima-para-baixo ao problema exposto, identifique as suas tarefas e sub-tarefas principais e defina uma abstracção de dados adequada.
- 2- Escreva em Scheme a abstracção de dados definida.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

- 3- Escreva em Scheme o programa teclado, apoiado na abstracção desenvolvida e nas tarefas e sub-tarefas identificadas.



Desenvolva e teste o procedimento teclado.

### Exercício 4

O programa classificacoes pede as classificações dos estudantes de uma turma e, no final, imprime o gráfico das classificações positivas.

```
> (classificacoes 'fp)
classificacao= 15
classificacao= 12
classificacao= 16
classificacao= 11
classificacao= 13
classificacao= 14
classificacao= 12
classificacao= 16
classificacao= 14
classificacao= 13
classificacao= 17
classificacao= 10
classificacao= 7
classificacao= 18
classificacao= 15
classificacao= 14
classificacao= 8
classificacao= 13
classificacao= -3
Classificacoes de fp:
10  *
11  *
12  * *
13  * * *
```

um valor negativo  
indica fim



<b>INTRODUÇÃO</b>
<b>1 - O ESSENCIAL DO SCHEME</b>
<b>2 - RECURSIVIDADE</b>
<b>3 - ABSTRACÇÃO DE DADOS</b>
<b>4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE</b>
<b>5 - ABSTRACÇÕES COM DADOS MUTÁVEIS</b>
<b>6 - SCHEME E OUTRAS TECNOLOGIAS</b>
<b>7-EXERCÍCIOS E PROJECTOS</b>
<b>ANEXOS</b>

```
14  * * *
15  *
16  *
17  *
18  *
19
20
```

- 1- Faça uma abordagem de-cima-para-baixo ao problema exposto, identifique as suas tarefas e sub-tarefas principais e defina uma abstracção de dados adequada.
- 2- Escreva em Scheme a abstracção de dados definida.
- 3- Escreva em Scheme o programa `classificacoes`, apoiado na abstracção desenvolvida e nas tarefas e sub-tarefas identificadas.



Desenvolva e teste o procedimento `classificacoes`.

### Exercício 5

Imagine um alvo constituído por uma matriz de 5 linhas por 5 colunas.

Esse alvo é representado computacionalmente por uma lista de 5 sublistas, em que cada uma destas sublistas, compostas por 5 elementos, representa uma linha do alvo.

Numa jogada é escolhida uma célula do alvo (através da linha e coluna correspondentes) e o número de pontos que se pretende somar aos existentes nessa célula.

1	2	3	4	5
1				
2				
3				
4				
5				

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



O programa `tiro-ao-alvo!` permite escolher um alvo e o número de jogadas consecutivas. No final das jogadas é visualizado o estado do alvo. É possível continuar a jogar com o mesmo alvo na situação em que se encontra ou então limpá-lo com `limpar-alvo!` e recomeçar.

```
> (define alvo-1 (criar-alvo))
> alvo-1
((0 0 0 0 0) (0 0 0 0 0) (0 0 0 0 0) (0 0 0 0 0) (0 0 0 0 0))
> (tiro-ao-alvo! alvo-1 2)
Jogada 1
Linha: 3
Coluna: 5
Pontos: 3
jugar com alvo-1 em 2 jogadas

Jogada 2
Linha: 2
Coluna: 4
Pontos: 6
Pontos a somar aos existentes

(0 0 0 0 0)
(0 0 0 6 0)
(0 0 0 0 3)
(0 0 0 0 0)
(0 0 0 0 0)
situação do alvo-1

ok

> (tiro-ao-alvo! alvo-1 1)
Jogada 1
Linha: 3
Coluna: 8
Pontos: 5
continua com alvo-1
em mais 1 jogada

Jogada 1
Linha: 2
Coluna: 4
Pontos: 1
situação do alvo-1

(0 0 0 0 0)
(0 0 0 7 0)
(0 0 0 0 3)
(0 0 0 0 0)
(0 0 0 0 0)
ok

> ...
```

- 1- Represente alvo-1, sob a forma caixa-e-apontador, no final do segundo conjunto de jogadas indicadas anteriormente.
  - 2- Faça uma abordagem de-cima-para-baixo ao problema exposto, identifique as suas tarefas e sub-tarefas principais e defina uma abstracção de dados adequada.
  - 3- Escreva em Scheme a abstracção de dados definida.
  - 4- Escreva em Scheme o programa `tiro-ao-alvo!`, apoiado na abstracção desenvolvida e nas tarefas e sub-tarefas identificadas.



Desenvolva e teste o programa *tiro-ao-alvo*!

## Exercício 6

O procedimento `elem-divisor` toma dois argumentos, `num` e `vec`, respectivamente, um número inteiro e um vector, e procura o primeiro elemento de `vec` que divide `num` (um elemento divide `num` se for zero o resto da divisão de `num` por esse elemento), devolvendo o seu índice, ou -1 caso não encontre. Para melhor entender a funcionalidade de `elem-divisor`, analise os exemplos que se seguem:

```
> (define v1 (vector))
> v1
#()
> (define v2 (vector 7 6 5 4 3 2))
> v2
#(7 6 5 4 3 2)
> (elem-divisor 10 v1)
-1 ←
nem um dos elementos do vector divide 10
> (elem-divisor 10 v2)
2 ←
2 é o índice de 5, número que divide 10
> (elem-divisor 13 v2)
-1
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



Escreva em Scheme o procedimento `elem-divisor`, integrando localmente os procedimentos auxiliares necessários.



Desenvolva e teste o procedimento `elem-divisor`.

### Exercício 7

Analise, nos exemplos que se seguem, o funcionamento do procedimento `actualizar`:

```
> (actualizar 5 '((3 1) (6 2)))
((3 1) (5 1) (6 2))
> (actualizar 3 '((3 1) (6 2)))
((3 2) (6 2))
> (actualizar 3 '((3 2) (6 2)))
((3 3) (6 2))
> (actualizar 5 '())
((5 1))
```

Escreva em Scheme o procedimento construtor `actualizar`.



Desenvolva e teste o procedimento `actualizar`.

### Exercício 8

O programa `matriz` toma um número não definido de argumentos, com os quais começa por definir uma matriz quadrada. Quando o número de argumentos não chega para uma matriz quadrada, a matriz é completada com zeros.

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

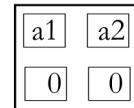
ANEXOS

Exemplos de chamadas de matriz, com as matrizes constituídas em cada caso, são indicados na figura:

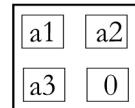
(matriz a1)



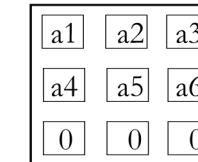
(matriz a1 a2)



(matriz a1 a2 a3)



(matriz a1 a2 a3 a4 a5 a6)



O programa matriz responde da seguinte forma, num exemplo em que são fornecidos 7 argumentos:

```
> (matriz 1 2 3 4 5 6 7)

O que deseja? linha/coluna/fim: coluna
Número de coluna, 1..3: 1
1 4 7
O que deseja? linha/coluna/fim: linha
Número de linha, 1..3: 2
4 5 6
O que deseja? linha/coluna/fim: linha
Número de linha, 1..3: 3
7 0 0
O que deseja? linha/coluna/fim: coluna
Número de coluna, 1..3: 3
3 6 0
O que deseja? linha/coluna/fim: linha
Número de linha, 1..3: 4
erro...
O que deseja? linha/coluna/fim: fim
fim...
```

- 1- Faça uma abordagem de-cima-para-baixo ao problema exposto, identifique as suas tarefas e sub-tarefas principais e defina uma abstracção de dados adequada.
- 2- Escreva em Scheme a abstracção de dados definida.



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

- 3- Escreva em Scheme o procedimento matriz, apoiado na abstracção desenvolvida e nas tarefas e sub-tarefas identificadas.



Desenvolva e teste o programa matriz.

### Exercício 9

O predicado tres-juntos? tem dois argumentos, simb e lis, sendo o primeiro um símbolo e o segundo uma lista. Este procedimento devolve #t quando encontra em lis, pelo menos, três símbolos seguidos e iguais a simb.

```
> (define lista1 '(o o x - x o o o x x o - x))  
> (tres-juntos? 'x lista1)  
#f  
> (tres-juntos? 'o lista1)  
#t  
> (tres-juntos? 'u lista1)  
#f
```

- 1- Escreva em Scheme o predicado tres-juntos?.
- 2- Escreva em Scheme o predicado n-juntos? com três argumentos, num, simb e lis, sendo o primeiro o número de símbolos iguais a simb que se pretendem encontrar juntos na lista lis.

```
> (define lista1 '(o o x - x o o o x x o - x))  
> (n-juntos? 2 'x lista1)  
#t
```



Desenvolva e teste os procedimentos tres-juntos? e n-juntos?.



INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



## Exercício 10

d1 e d2 são listas com três elementos, que representam datas, (dia mes ano).

O procedimento `idade-anos` toma dois argumentos, d1 e d2, e calcula a idade, à data d2, de uma pessoa que tenha nascido em d1. A resposta é dada em número inteiro de anos.

```
> (define hoje (list 18 01 2006))
> (idade-anos (list 01 02 2003) hoje)
idade = 2 anos
> (idade-anos (list 01 04 2004) hoje)
idade = 1 ano
> (idade-anos (list 01 11 2005) hoje)
idade = 0 anos
> (idade-anos (list 01 12 2006) hoje)
idade = !!!
```

quando d1 é posterior a d2  
a resposta é !!!

- 1- Apresente uma abstracção de dados para esta situação (estruturas de dados, construtores, selectores e, eventualmente, modificadores).
- 2- Sobre a abstracção referida, escreva em Scheme o procedimento `idade-anos`.
- 3- Escreva em Scheme o procedimento `idade-anos-meses`, que difere do procedimento `idade-anos`, essencialmente no tipo de resposta, pois considera também, para além do número de anos, o número de meses.

```
> (idade-anos-meses (list 01 04 2004) (list 16 01 2006))
idade = 1 ano e 9 meses
> (idade-anos-meses (list 15 11 2005) (list 16 01 2006))
idade = 0 anos e 2 meses
> (idade-anos-meses (list 16 11 2005) (list 16 01 2006))
idade = 0 anos e 1 mes
> (idade-anos-meses (list 17 11 2005) (list 16 01 2006))
idade = 0 anos e 1 mes
> (idade-anos-meses (list 01 12 2006) (list 16 01 2006))
idade = !!!
```



Desenvolva e teste os procedimentos `idade-anos` e `idade-anos-meses`.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 11

Cada uma das figuras mostra duas rodas concêntricas, com igual número de sectores, onde são representados valores inteiros. Em fig. 1, as duas rodas apresentam valores coincidentes apenas num caso (valor 3). A roda exterior é fixa e a interior pode rodar no sentido dos ponteiros do relógio. Em fig. 2, a roda interior avançou um passo, e o número de valores coincidentes continua a ser um (valor 1). Se a roda interior avançar mais um passo, fig. 3, o número de valores coincidentes passa a ser 2 (valores 1 e 2). Finalmente, se a roda interior avançar mais um passo, fig. 4, o número de coincidências passa a zero.

O procedimento `max-coincidencias` toma duas listas de comprimento igual, que representam duas rodas, e determina a situação com o maior número de coincidências.

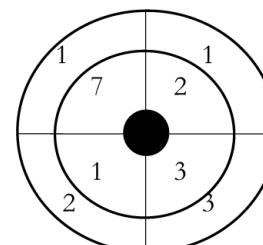


fig. 1

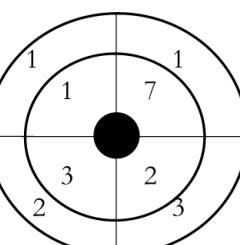


fig. 2

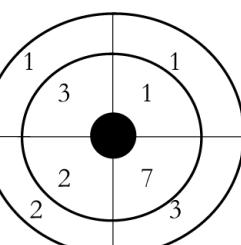


fig. 3

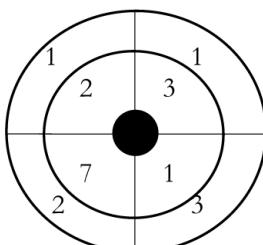


fig. 4

Para as rodas indicadas na fig. 1, devolveria a seguinte mensagem:

número de passos= 2      número máximo de coincidências= 2

- 1- Escreva em Scheme o procedimento `mais-um-passo`, que recebe uma lista e devolve uma lista de igual comprimento, rodada uma posição.

```
> (mais-um-passo (list 1 2 3 4))  
(4 1 2 3)
```

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS



- 2- Escreva em Scheme o procedimento num-coincidencias, que recebe duas listas de igual comprimento e devolve o número de valores coincidentes.

```
> (num-coinc (list 1 2 3 4) (list 2 3 4 3 5))  
1
```

3 será uma vez coincidente

- 3- Escreva em Scheme o procedimento max-coincidencias, baseado em mais-um-passo e em num-coincidencias.



Desenvolva e teste os procedimentos max-coincidencias, mais-um-passo e num-coincidencias.

### Exercício 12

Segundo a conjectura *Goldbach*, os inteiros pares maiores que 2 são iguais à soma de 2 números primos.

Exemplos:  $4 = 2 + 2$ ;  $6 = 3 + 3$ ;  $8 = 5 + 3$ ;  $10 = 7 + 3$ ;  $12 = 7 + 5$ ;  $14 = 11 + 3$ ; ...

Considerar agora um programa em Scheme que suporta diálogos do seguinte tipo:

```
> (goldback)  
Apresentar um inteiro par e positivo, maior que 2: 3  
3 não é par!!!
```

```
Apresentar um inteiro par e positivo, maior que 2: 4  
4 = 2 + 2
```

```
Apresentar um inteiro par e positivo, maior que 2: -1  
-1 não é positivo!!!
```

```
Apresentar um inteiro par e positivo, maior que 2: 20  
20 = 17 + 3
```

```
Apresentar um inteiro par e positivo, maior que 2: 0  
FIM!
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



- 1- Escreva em Scheme o predicado `e-primo?`, que recebe um número positivo e determina se é ou não primo, devolvendo, respectivamente, `#t` ou `#f`.  
Pista: Procurar um divisor do número, de 2 até metade do número.
- 2- Escreva em Scheme o procedimento `gold` que recebe um número par e positivo, maior que 2, e visualiza os primos correspondentes, de acordo com a conjectura de *Goldbach*. Por exemplo, a chamada `(gold 20)` visualiza  $17 + 3$ .  
Pista:
  - a) de número - 1 a 2 procurar um primo `p1`.
  - b) de `p1` a 2 procurar um primo `p2`, tal que  $\text{número} = p1 + p2$ .
  - c) se não encontrar `p1` e `p2` naquela situação, repetir a) de `p1-1` a 2 ...
- 3- Escreva em Scheme o programa `goldback`, que suporta o diálogo acima indicado.



Desenvolva e teste os procedimentos `goldback`, `gold` e `e-primo?`.

### Exercício 13

Para a compilação de programas recorre-se muitas vezes a uma estrutura de dados para armazenar temporariamente os símbolos relativos às constantes, variáveis e tipos que vão sendo definidos e utilizados ao longo do programa, estrutura de dados normalmente designada por tabela de símbolos.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

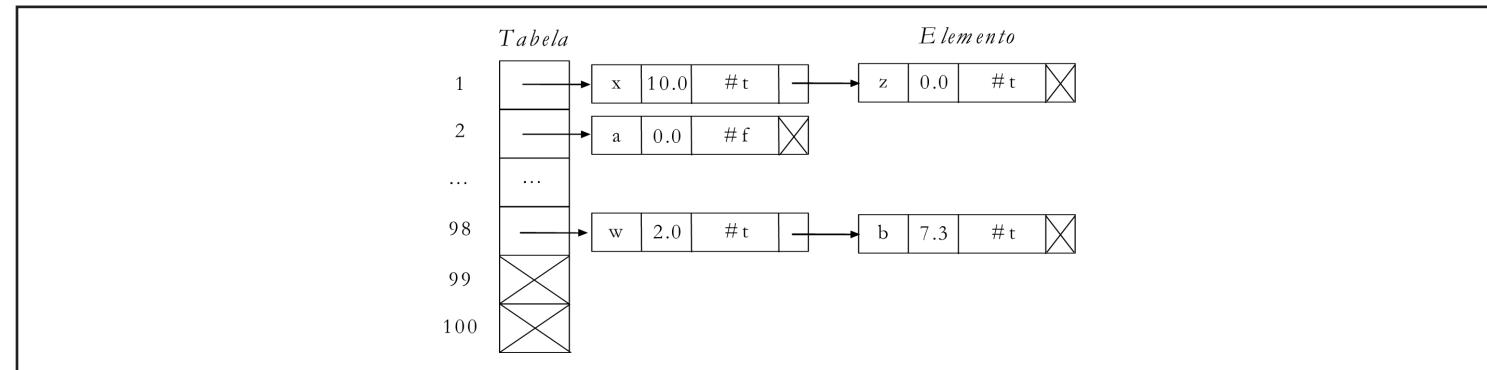
### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



Na figura, apresenta-se uma das suas configurações possíveis: um vector de  $n$  apontadores para listas de elementos, em que cada elemento contém o símbolo, o seu valor, e um campo indicativo do seu estado, activo (#t) ou inactivo (#f).



Em torno desta estrutura de dados criou-se a abstracção que vamos designar por tabela de símbolos, definindo alguns procedimentos, entre os quais de distinguem:

#### Construtor

O procedimento `criar-tabela-de-simbolos` recebe um inteiro, `num`, e devolve uma tabela de simbolos vazia, ou seja, um vector de `num` posições todas elas preenchidas com '()' .

#### Selector

O procedimento `ler-símbolo` recebe uma tabela de símbolos, `tab`, um inteiro, `n-int`, e um símbolo, `simb`, e procura `simb` a partir da posição `n-int` de `tab`. Devolve `#f` se não encontrar `simb` ou, se o encontrar, devolve uma lista em que o primeiro elemento é o valor associado a `simb` e o segundo é um booleano que representa o seu estado de activação.

#### Modificador

O procedimento `inserir-símbolo!` aceita cinco argumentos (uma tabela de símbolos, um símbolo a inserir na tabela, um inteiro que indica a posição do vector onde inserir o símbolo, um valor associado ao símbolo e um booleano que representa o seu estado de activação) e devolve o símbolo `ok`. O elemento é inserido no princípio da lista apontada por essa posição, mesmo que outros elementos já lá se encontrem. Caso já exista um elemento na posição dada com o símbolo a inserir, então ocorrerá, simplesmente, a actualização dos campos do valor associado ao símbolo e do seu estado de activação.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



O exemplo de tabela apresentado na figura resulta da seguinte sequência de invocações de inserir-símbolo!: o símbolo b na posição 98, o símbolo z na posição 1, o símbolo x na posição 1, o símbolo a na posição 2 e o símbolo w na posição 98.

- 1- Apresente a estrutura indicada na figura, numa representação gráfica caixa-e-apontador.
- 2- Escreva em Scheme o procedimento criar-tabela-de-símbolos.
- 3- Escreva em Scheme o procedimento inserir-símbolo!.
- 4- Escreva em Scheme o procedimento ler-símbolo.



Desenvolva e teste a abstracção tabela de símbolos, com o construtor, selector e modificador indicados.

### Exercício 14

Projecto - A lista telefónica

Suponha que uma lista telefónica, com a designação lista-telefonica, é representada por um vector, como se indica no exemplo que se segue:

```
#((('joao 1835) ('maria 1823) ('joaquim 1845) ('mafalda 1805)  
  ('carlos 1856) ('joana 1830)))
```

Sobre lista-telefonica, o programa operar-lista permite interacções do tipo:

```
> (operar-lista lista-telefonica)  
  
O que deseja? telefone/alterar/acrescentar/retirar/fim: telefone  
indicar o nome: joana  
joana tem o telefone 1830  
  
O que deseja? telefone/alterar/acrescentar/retirar/fim: telefone  
indicar o nome: pedro  
pedro não tem telefone!!!
```

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



O que deseja? telefone/alterar/acrescentar/retirar/fim: qualquer comando não previsto!!!

O que deseja? telefone/alterar/acrescentar/retirar/fim: alterar indicar o nome: joana  
indicar o novo número: 1900  
joana com número alterado

O que deseja? telefone/alterar/acrescentar/retirar/fim: telefone  
indicar o nome: joana  
joana tem o telefone 1900

O que deseja? telefone/alterar/acrescentar/retirar/fim: alterar  
indicar o nome: pedro  
indicar o novo número: 2000  
pedro não tem telefone

O que deseja? telefone/alterar/acrescentar/retirar/fim: fim  
Trabalho finalizado...

- 1- Tendo em conta o diálogo apresentado, defina rigorosamente como se comportará o programa face a todos os comandos, incluindo os que não foram considerados naquela interacção, e também como se comportará quando receber respostas inadequadas.
- 2- Faça uma abordagem de-cima-para-baixo ao problema exposto, identifique as suas tarefas e sub-tarefas principais e defina uma abstracção de dados adequada.
- 3- Escreva em Scheme a abstracção de dados definida.
- 4- Escreva em Scheme o programa operar-lista, apoiado na abstracção desenvolvida e nas tarefas e sub-tarefas identificadas.



Desenvolva e teste o projecto A lista telefónica.

Numa segunda fase deverá prever que as listas telefónicas poderão ser guardadas em ficheiro de texto.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



### Exercício 15

#### Projecto - O jogo dos 14 fósforos

Vamos considerar que o *jogo-dos-14-fósforos* vai ter apenas dois jogadores em cada sessão. Inicialmente, são colocados 14 fósforos em cima de uma mesa e cada jogador pode, alternadamente, retirar 1 ou 2 fósforos. Ganha o jogador que retirar o último dos 14 fósforos.

- 1- Imagine o funcionamento de um programa que permita que dois humanos joguem o *jogo-dos-14-fósforos*.
- 2- Identifique as principais tarefas e sub-tarefas do problema exposto e, apoiado nelas, escreva em Scheme o programa imaginado, que será designado por *jogo-dos-14-fósforos*.
- 3- Escreva uma nova versão do programa que permita o *jogo-dos-14-fósforos* entre um jogador humano e o computador, que será designado por *jogo-dos-14-fósforos-com-computador*. Pretende-se que o computador tenha alguma estratégia nas suas jogadas, de tal forma que perante as 2 hipóteses, retirar 1 ou retirar 2 fósforos, escolha a que lhe proporcione mais possibilidades de vitória.



Desenvolva e teste o projecto O jogo dos 14 fósforos.

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

### Exercício 16

Projecto - O caminho mais curto

Um tabuleiro de 5 x 5 tem as suas células identificadas de 1 a 25. A cada uma das células do tabuleiro é associado aleatoriamente um valor inteiro situado entre 0 e 9.

identificação das células

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

um tabuleiro preenchido

1	5	7	3	0
1	2	8	2	1
0	6	1	0	2
3	7	3	9	4
5	2	5	2	5

Dada uma célula de partida, pretende-se determinar o caminho mais curto entre a célula dada e a célula de chegada, que é a 25. Entende-se por caminho mais curto o que corresponde à soma mínima dos valores associados às células por onde passa. Por exemplo, sendo 18 a célula de partida, o caminho mais curto passará pelas células: 18, 23, 24 e 25, a que corresponde um comprimento de  $3 + 5 + 2 + 5 = 15$ .

- Sabendo que os deslocamentos possíveis, a partir de uma posição pos, são pos+1 (se pos não estiver na coluna 5) e pos+5 (se pos não estiver na linha 5), escreva em Scheme o programa com a designação procura-caminho-min que começa por gerar um tabuleiro com pesos aleatórios entre 0 e 9 e, de seguida, visualiza um tabuleiro com a identificação das posições e outro com os pesos respectivos. O programa pede ainda a célula de partida, determina o caminho mais curto entre ela e a célula 25 e visualiza o respectivo comprimento e as células por onde passa.

```
> (procura-caminho-min)
 1 2 3 4 5      4 4 1 9 3
 6 7 8 9 10     2 6 3 7 5
 11 12 13 14 15 3 4 8 3 0
 16 17 18 19 20 1 6 4 2 3
 21 22 23 24 25 6 1 0 0 6
partida: 8
Comprimento do percurso mais curto: 21
O percurso passa por:
8 9 14 19 24 25
ok
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

```
> (procura-caminho-min)
 1  2  3  4  5      5  4  9  2  6
 6  7  8  9  10     5  8  0  6  0
 11 12 13 14 15    5  8  6  8  8
 16 17 18 19 20    1  1  2  1  7
 21 22 23 24 25    4  2  5  8  6
partida: 1
Comprimento do percurso mais curto: 33
O percurso passa por:
1 6 11 16 17 18 19 20 25
ok
```



Desenvolva e teste o projecto O caminho mais curto.

### Exercício 17

Desenvolva um programa para gerar apostas do Totoloto, baseado num procedimento com um só parâmetro, n-apostas, relacionado com o número de apostas a gerar.

```
> (preenche-totoloto 10)
Apostas seleccionadas:
(5 22 26 31 41 46)
(2 12 24 25 41 43)
(10 19 20 23 48 49)
(15 19 22 41 42 48)
(12 32 33 35 42 46)
(10 19 26 29 43 49)
(4 7 21 41 43 44)
(6 11 33 35 42 47)
(3 4 17 18 19 48)
(18 19 21 22 28 37)
```

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



> (preenche-totoloto 4)

Apostas seleccionadas:

(5 8 9 10 18 43)

(5 23 26 34 40 42)

(3 10 17 19 34 41)

(8 10 19 23 36 44)



Desenvolva e teste o programa preenche-totoloto.

### Exercício 18

Projecto - O jogo master mind

No jogo master mind, um jogador pensa numa combinação secreta de cores e outro jogador vai procurar adivinhar, por tentativas, a combinação de cores pensada pelo primeiro jogador.

Os jogadores utilizam pinos coloridos e, à partida, sabe-se quantos pinos compõem uma combinação e quantas cores diferentes de pinos existem.

Por exemplo, atribuindo um número a cada cor, vamos considerar pinos azuis (1), vermelhos (2) e verdes (3) e combinações de 4 pinos. Uma combinação secreta pode ser 2 1 3 1.

Imaginemos que a primeira tentativa do outro jogador foi 1 2 3 2.

O jogador que escolheu a combinação secreta responde que, na jogada efectuada, há um pino preto e dois brancos. O preto corresponde ao 3 pois a cor está certa na posição certa. Os pinos brancos correspondem ao 1 e a um dos 2, uma vez que a cor está certa, mas em posição incorrecta. O jogador que adivinha tentará, com a informação que vai recebendo do outro jogador, outras combinações, procurando chegar à combinação secreta.

- 1- Propõe-se o desenvolvimento de um programa em que o computador faz de primeiro jogador e o utilizador faz de jogador que adivinha. Este programa reveste a forma de um procedimento com o nome `mm-utilizador-adivinha`, com os parâmetros `n-pin`s e `n-cores`, em que o primeiro representa o número de pinos de cada combinação e o segundo o número de cores possíveis.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

```
> (mm-utilizador-adivinha 4 3)
```

Ja tenho uma combinacao secreta... ← diz o computador...

A tua vez (4 pinos com cores de 1 a 3): 1 1 1 1  
pinos pretos: 2      pinos brancos: 0 ← 2 cores na posição correcta

A tua vez (4 pinos com cores de 1 a 3): 1 1 2 2  
pinos pretos: 1      pinos brancos: 2 ← 1 cor na posição correcta e  
2 em posição incorrecta

A tua vez (4 pinos com cores de 1 a 3): 1 2 1 2  
pinos pretos: 3      pinos brancos: 0

A tua vez (4 pinos com cores de 1 a 3):

cores: ...

...

pinos pretos: 4      pinos brancos: 0 ← 4 cores na posição correcta...

Numero de jogadas para acertar: 7 acertou!!!



Desenvolva e teste o projecto O jogo master mind na versão em que é o jogador a adivinhar.

2- Propõe-se agora o desenvolvimento de um programa em que o utilizador faz de primeiro jogador e o computador faz de jogador que adivinha. Este programa reveste a forma de um procedimento com o nome mm-computador-adivinha, com os parâmetros n-pinôs e n-cores, em que o primeiro representa o número de pinos de cada combinação e o segundo o número de cores possíveis.

No diálogo que se segue, imagine que o utilizador escolheu como combinação secreta 2 1 3 1.

```
> (mm-computador-adivinha 4 3)
```

O utilizador escreve num papel a combinacao secreta.

Ja' está? (actuar uma tecla, seguida de return) s

jogada realizada: (1 2 2 1)

pinos pretos: 1

pinos brancos: 2



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

jogada realizada: (2 2 1 3)

pinos pretos: 1

pinos brancos: 2

jogada realizada: (2 1 2 2)

pinos pretos: 2

pinos brancos: 0

jogada realizada: (2 1 3 1)

pinos pretos: 4

pinos brancos: 0

Número de jogadas para acertar: 4

Pista

Por se tratar de um problema com alguma complexidade, adianta-se uma pista para auxiliar o computador a adivinhar a combinação secreta. Toma-se, como ponto de partida, a lista de todas as combinações possíveis que inclui, obviamente, a combinação secreta que o computador vai tentar adivinhar.

Em cada jogada, o computador elimina algumas combinações ou, de outra forma, apenas mantém as que, comparadas com a combinação jogada, originarem uma resposta idêntica à obtida nessa jogada. Por exemplo, se de uma combinação jogada resultarem 2 pinos pretos e 1 branco, então, das combinações ainda existentes, apenas serão guardadas as que comparadas com a combinação jogada originarem também 2 pinos pretos e 1 branco. Entre elas estará, certamente, a combinação secreta...



Desenvolva e teste o projecto O jogo master mind na versão em que é o computador a adivinhar.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MÚTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

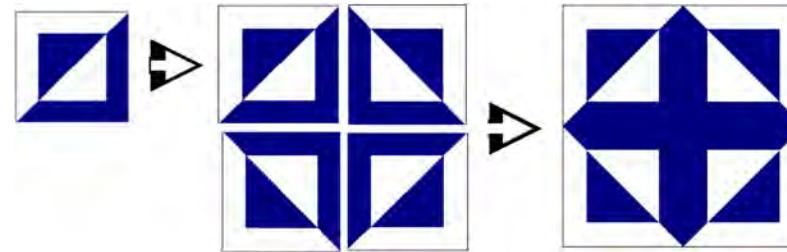
### ANEXOS

### Exercício 19 - exercício de grande complexidade

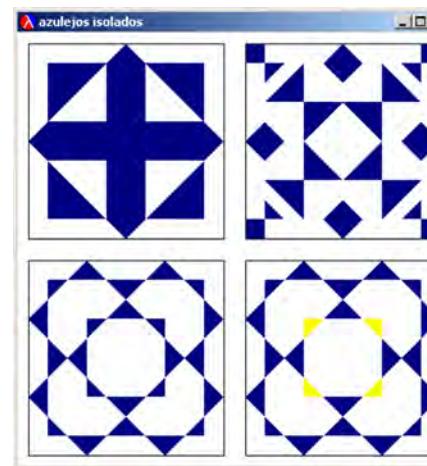
#### Projecto - Pintura de azulejos

Como pode observar nas figuras, alguns azulejos são concebidos a partir de um dos seus quadrantes. Vamos começar por pintar o quadrante superior-esquerdo do azulejo. Depois fazem-se 3 cópias desse quadrante, as quais são rodadas em torno do respectivo ponto central. Rodando 90°, obtém-se o quadrante superior-direito. Rodando 180°, obtém-se o quadrante inferior-direito. Rodando 270°, obtém-se o quadrante inferior-esquerdo.

Colando convenientemente os 4 quadrantes, 3 deles rodados como foi indicado, obtém-se o azulejo que vamos designar por azul-cruz.



Na figura seguinte, pode observar quatro tipos de azulejos, todos eles criados a partir da técnica descrita, ou seja, pinta-se o quadrante superior-esquerdo e os restantes 3 são obtidos por rotação.



# **S C H E M E**

## **na descoberta da programação**

### **INTRODUÇÃO**

### **1 - O ESSENCIAL DO SCHEME**

### **2 - RECURSIVIDADE**

### **3 - ABSTRACÇÃO DE DADOS**

### **4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE**

### **5 - ABSTRACÇÕES COM DADOS MUTÁVEIS**

### **6 - SCHEME E OUTRAS TECNOLOGIAS**

### **7-EXERCÍCIOS E PROJECTOS**

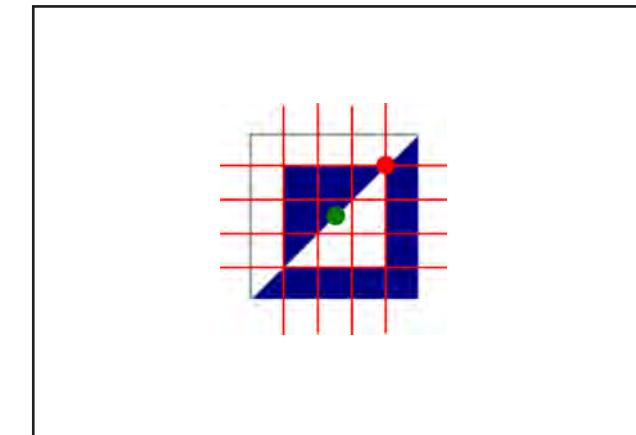
### **ANEXOS**

Imagine que o quadrante superior-esquerdo do azul-cruz corresponde ao padrão1, codificado como se indica mais à frente.

Note que este quadrante é composto por duas manchas poligonais. A codificação de um padrão tem em consideração que a caneta se encontra no ponto central desse mesmo padrão, antes de iniciar a pintura de cada mancha poligonal. Isto implica, normalmente, uma deslocação da caneta, sem desenhar, daquele ponto central para o início da mancha e, no final, o regresso ao ponto central também sem desenhar.

Um padrão é representado por uma lista de sublistas, existindo uma sublista por cada mancha poligonal do padrão. Em cada sublista, o primeiro elemento é o índice da cor atribuída à mancha, o segundo elemento é o deslocamento desde o ponto central do padrão (pequeno círculo verde na figura que se segue) até ao início da mancha (pequeno círculo vermelho na figura que se segue) e o terceiro elemento representa a linha poligonal que serve de fronteira à mancha.

```
(define padrao1
  (lambda (lado) ; o parâmetro representa o comprimento do lado do azulejo em pixels
    (let ((u (/ lado 5))) ; definição de variáveis locais
      (u3 (/ (* 3 lado) 5))
      (u1.5 (/ (* 3 lado)
                 10)))
      (list
        ; mancha de 6 lados
        (list 1 ; cor de índice 1
              (list u1.5 u1.5) ; desloca do centro
              (list ; do padrao para o
                    (list 0 0) ; início do polígono
                    (list u u)
                    (list 0 (- lado))
                    (list (- lado) 0)
                    (list u u)
                    (list u3 0)
                    (list 0 u3))))
        ; mancha triangular
        (list 1
              (list u1.5 u1.5)
              (list
                (list 0 0)
                (list (- u3) (- u3))
                (list 0 u3)
                (list u3 0)))))))
```



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

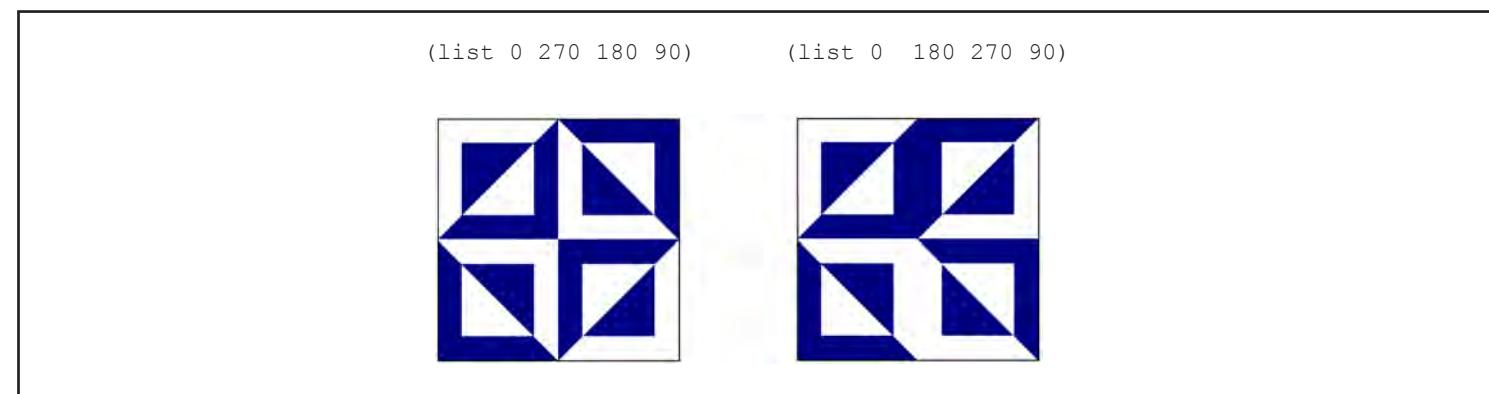
## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

Um procedimento, designado por desenha-azulejo, tem como parâmetros lado-azul, padrao, lista-de-4-angulos e lista-de-indices-de-cor. Este procedimento supõe que a caneta se encontra no ponto sobre o qual vai ser pintado o ponto central do azulejo e desenha o padrão 4 vezes de acordo com o que se indicou.

Os 2 últimos parâmetros recomendam uma explicação adicional. O parâmetro lista-de-4-angulos é normalmente uma lista constituída pelos valores 0, 90, 180 e 270, como seria de esperar, pois correspondem ao valor de rotação do padrão nos 4 sectores. Decidiu-se fornecer estes ângulos através de uma lista de 4 valores, pois assim obtém-se uma flexibilidade maior para, a partir de um dado padrão, produzir vários tipos de azulejos. Na figura, podem observar os azulejos obtidos, a partir de padrao1, quando se altera a rotação do padrão em cada um dos 4 sectores.



Com o parâmetro lista-de-indices-de-cor também se procurou uma flexibilidade maior em relação às cores utilizadas na pintura de cada um dos padrões de um azulejo. Assim, a cor que inicia a codificação da primeira mancha poligonal de um padrão é substituída pela primeira cor da lista, a cor da segunda mancha é substituída pela segunda cor da lista. Se esgotarem as cores da lista e ainda houver manchas para pintar, retoma-se a primeira cor da lista, depois a segunda, e assim sucessivamente até pintar todas as manchas. Todos os azulejos baseados em padrao1, anteriormente apresentados, utilizaram uma lista-de-indices-de-cor equivalente a (list 1), uma vez que todas as manchas foram pintadas com a cor de índice 1, o azul. Sugere-se a observação dos azulejos obtidos com o mesmo padrao1, alterando apenas o conteúdo de lista-de-indices-de-cor.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

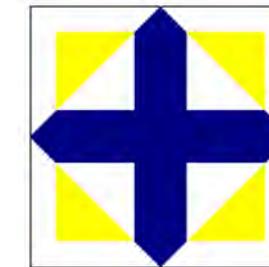
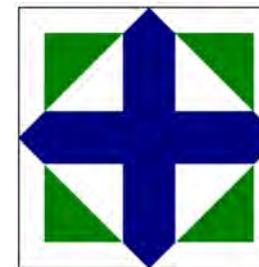
### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS



(list 0 90 180 270) (list 1 3) (list 0 90 180 270) (list 1 24)



Com o procedimento desenha-azulejo torna-se relativamente fácil criar vários tipos de azulejos, tendo como ponto de partida um conjunto de padrões previamente definidos. Indica-se, como exemplo, a criação do azulejo designado por azul-cruz.

```
(define azul-cruz
  (lambda (lado)
    (desenha-azulejo lado padrao1 (list 0 90 180 270) (list 1))))
```

Para não obrigar a uma incursão no domínio da Trigonometria e antes de se passar à especificação dos procedimentos que se pretendem desenvolver, dá-se conhecimento do procedimento roda-ponto, com os parâmetros lis-ponto, cos-a e sin-a. Sendo lis-ponto uma lista com as coordenadas x e y de um ponto e cos-a e sin-a o cosseno e o seno, respectivamente, do ângulo de rotação do ponto lis-ponto em torno da origem dos eixos, roda-ponto devolve o ponto correspondente a lis-ponto depois de rodado.

```
(define roda-ponto
  (lambda (lis-ponto cos-a sin-a)
    (let ((x (car lis-ponto))
          (y (cadr lis-ponto)))
      (list (+ (* x cos-a) (* y sin-a))
            (- (* y cos-a) (* x sin-a))))))
```

# SCHEME na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

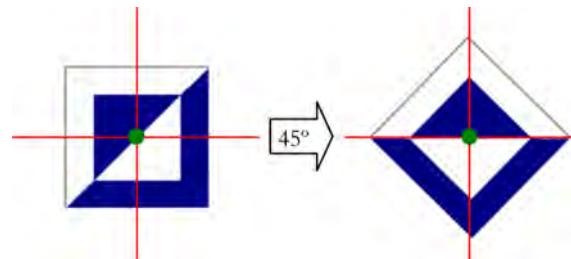
5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

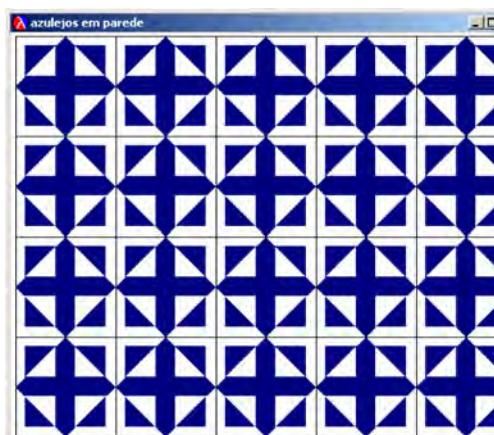
Tendo em consideração que as manchas dos padrões são codificadas em movimentos relativos, sendo o centro do padrão a origem do primeiro movimento, não será difícil admitir que da aplicação de roda-ponto, com um ângulo ang, a todos os pontos que definem a fronteira de uma mancha, fará rodar essa mancha de um ângulo ang em torno do centro do padrão.



Depois deste enunciado já tão longo, apresentam-se os procedimentos que deve desenvolver.

- 1- O procedimento desenha-parede tem como parâmetros x-org, y-org, n-col, n-lin, tipo-de-azulejo e lado e desenha uma parede com uma matriz de n-col colunas por n-lin linhas, utilizando para tal azulejos tipo-de-azulejo, com o comprimento do lado especificado por lado. Os 2 primeiros parâmetros indicam, na janela gráfica activa, a posição do ponto central do azulejo situado no canto superior-esquerdo.

> (janela 500 400 "azulejos em parede")  
> (desenha-parede 54 346 5 4 azul-cruz 98)



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

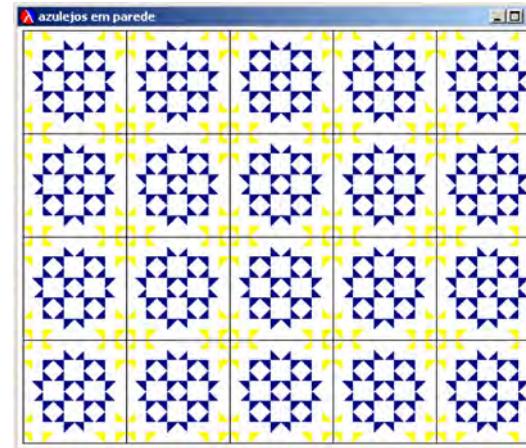
5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

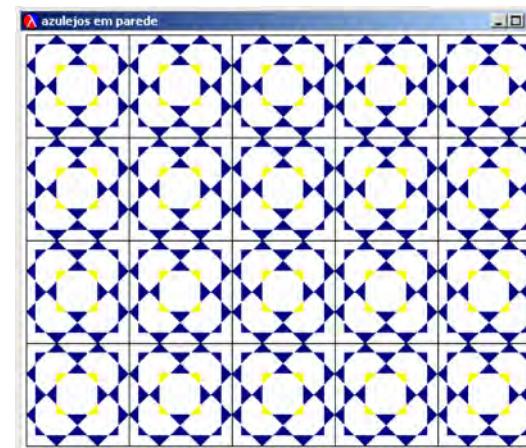
7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

```
> (limpa)
> (desenha-parede 54 348 5 4 azul-amare-flor 98)
```



```
> (limpa)
> (desenha-parede 54 348 5 4 azul-amare3-estrela 98)
```



- 2- O procedimento desenha-catalogo tem como parâmetros x-org, y-org, n-col, n-lin, lado e lista-de-azulejos e desenha um catálogo de azulejos numa parede de n-col colunas por n-lin linhas, retirando aleatoriamente azulejos da lista lista-de-azulejos, com o comprimento do lado especificado por lado. Pode usar azulejos repetidos.



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

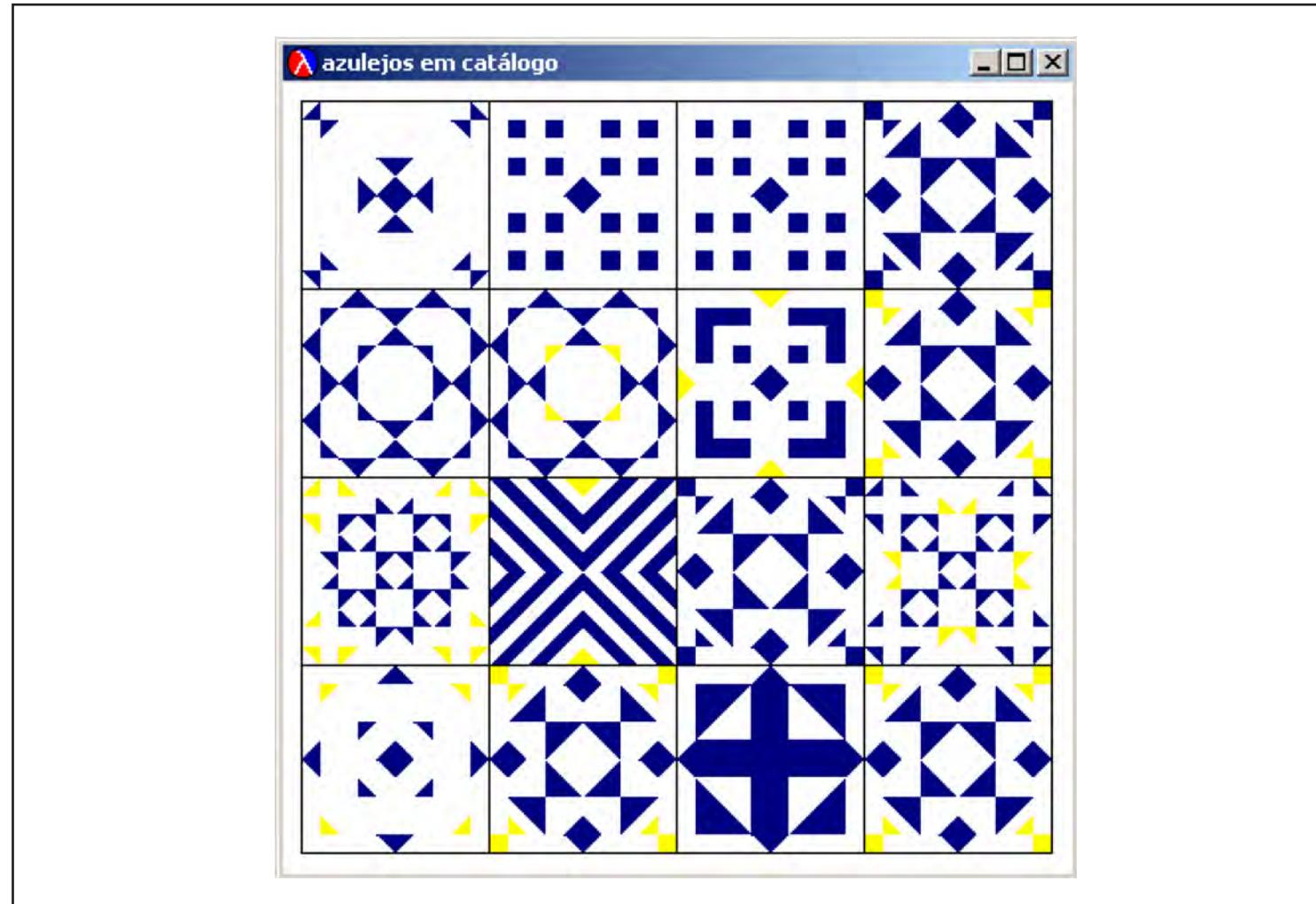
6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

Os 2 primeiros parâmetros indicam, na janela gráfica activa, a posição do ponto central do azulejo situado no canto superior-esquerdo.

```
> (janela 400 400 "azulejos em catálogo")
> (desenha-catalogo 57 343 4 4 95 lista-azulejos)
```



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

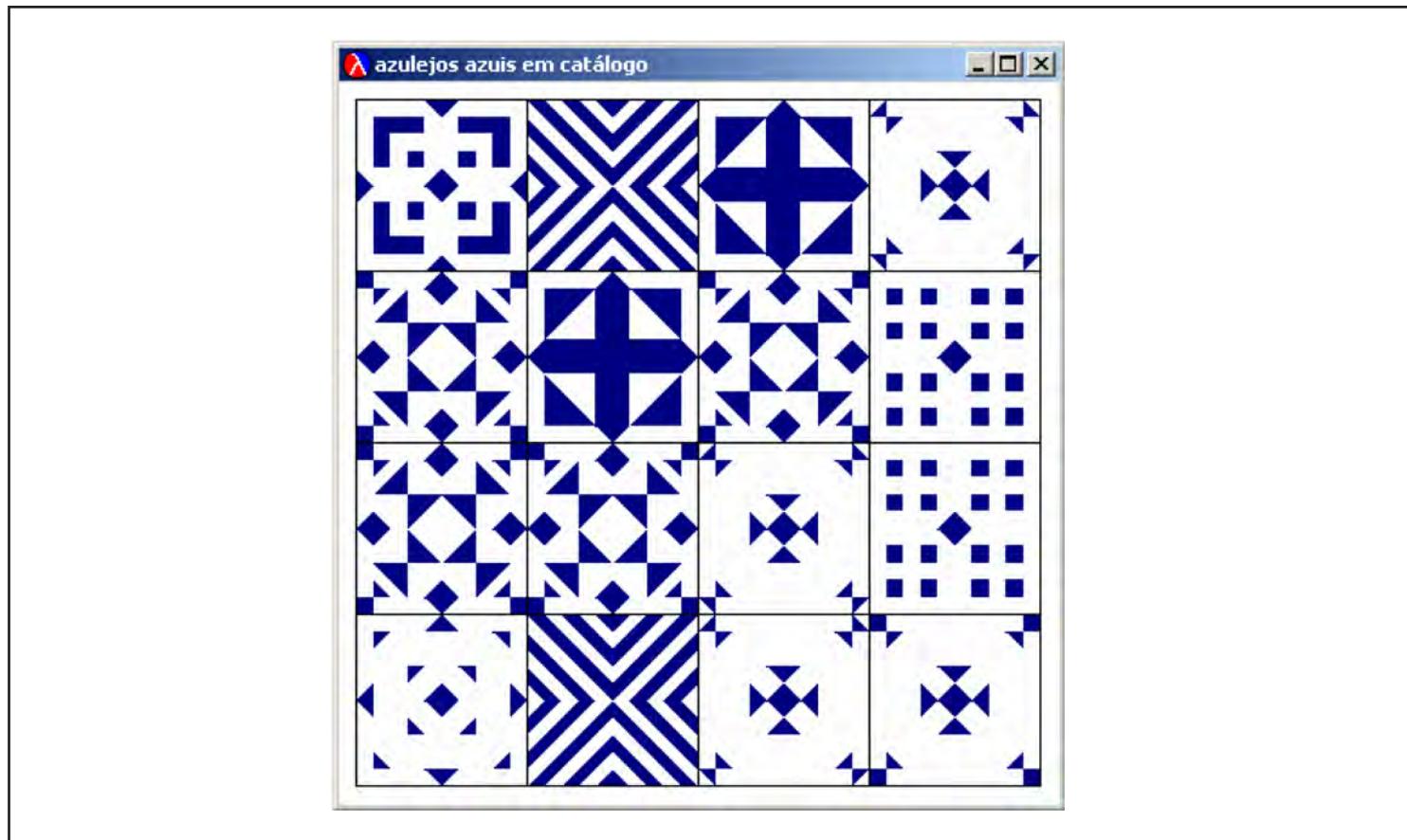
4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS



Finalmente, conceba e implemente um programa para desenhar padrões interactivamente sobre o ecrã e, a partir deles, novos tipos de azulejos. Os padrões e os tipos de azulejos deverão poder ser guardados em ficheiros.



Desenvolva e teste o projecto pintura de azulejos.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D



## Anexo A - Principais procedimentos da linguagem Scheme

Neste anexo resumem-se os principais procedimentos da linguagem Scheme, acompanhados de exemplos e agrupados em:

- Processamento booleano
- Processamento numérico
- Processamento trigonométrico
- Controlo de sequência
- Entrada/Saída
- Processamento de Pares e Listas
- Processamento de Caracteres e Cadeia de caracteres (strings)
- Processamento de Vectores
- Processamento de Ficheiros
- Vários

Uma descrição completa desta linguagem encontra-se em

*Revised (5) Report on the Algorithmic Language Scheme*

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

**A B C D**

## Processamento booleano

```
#f ; valor booleano falso.
#t ; valor booleano verdadeiro.
(boolean? x) ; qualquer valor diferente de #f é considerado #t. (not 5) devolve #f.
(and x1 x2 x3 ...) ; se x for booleano, devolve #t, se não, devolve #f.
; calcula x1, x2, x3, ..., até encontrar um
; valor #f, e devolve #f.
(or x1 x2 x3 ...) ; calcula x1, x2, x3, ..., até encontrar um
; valor #t, e devolve #t.
; não encontrando qualquer valor #t, devolve #f.
(not x) ; não encontrando qualquer valor #t, devolve #f.
; se x = #t, devolve #f, se não, devolve #t.
```

## Processamento numérico

```
(number? x) ; devolve #t, se x for número, se não, #f.
(integer? x) ; devolve #t, se x inteiro, se não, #f.
(real? x) ; devolve #t, se x real, se não, #f.
(+ x1 x2 x3 ...) ; devolve soma de x1, x2, x3, ...
(- x1 x2 x3 ...) ; subtrai a x1, sucessivamente, x2, x3, ...
(* x1 x2 x3 ...) ; com um só argumento: (- x) devolve -x.
(/ x1 x2 x3 ...) ; devolve produto de x1, x2, x3, ...
(< x1 x2 x3 ...) ; divide x1, sucessivamente, por x2, x3, ...
(> x1 x2 x3 ...) ; com um só argumento: (/ x) devolve 1/x.
(<= x1 x2 x3 ...) ; devolve #t, se x1, x2, x3,... em ordem crescente,
; se não, #f.
(>= x1 x2 x3 ...) ; devolve #t, se x1, x2, x3,... em ordem decrescente,
; se não, #f.
(= x1 x2 x3 ...) ; devolve #t, se x1, x2, x3, ... iguais, se não, #f.
(<= x1 x2 x3 ...) ; devolve #t, se x1, x2, x3, ... em ordem
; não decrescente, se não, #f.
(>= x1 x2 x3 ...) ; devolve #t, se x1, x2, x3, ... em ordem
; não crescente, se não, #f.
(add1 x) ; devolve x+1.
(sub1 x) ; devolve x-1.
(sqrt x) ; devolve raiz quadrada de x, sendo x >= 0.
(exp x) ; devolve ex.
(expt x y) ; devolve xy.
(log x) ; devolve logaritmo de x na base e.
(abs x) ; devolve valor absoluto de x.
; (abs 5.1) devolve 5.1 e (abs -5.1) devolve 5.1.
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

```
(round x) ; devolve o inteiro mais próximo de x. Existindo 2
; inteiros igualmente distantes, devolve o que for par.
; (round 2.7) devolve 3.0, (round 2.5) devolve 2
; e (round 3.5) devolve 4.
; devolve inteiro igual a x ou imediatamente acima.
; (ceiling 5.3) devolve 6.0 e (ceiling -5.3) devolve -5.0.
; devolve inteiro igual a x ou imediatamente abaixo.
; (floor 5.3) devolve 5.0, (floor -5.3) devolve -6.0.
; devolve inteiro constituído pela parte inteira de x.
; (truncate 2.7) devolve 2.0; (truncate -5.3) devolve -5.0.
; devolve o valor do maior argumento.
; devolve o valor do menor argumento.
; devolve o quociente da divisão inteira x/d.
; (quotient 7 3) devolve 2.
; (quotient -17 3.0) devolve -5.0.
; devolve o resto (com o mesmo sinal de x)
; da divisão inteira x/d.
; (remainder -7 3) devolve -1, pois -7= 3 * -2 + -1.
; (remainder 7 -3) devolve 1, pois 7= -3 * -2 + 1.
; (remainder 7 3) devolve 1, pois 7= 3 * 2 + 1.
; (remainder -7 -3) devolve -1, pois -7= -3 * 2 + -1.
; devolve o resto (com o mesmo sinal de d) da
; divisão inteira x/d.
; quando x e d têm o mesmo sinal,
; modulo devolve o mesmo resultado que remainder.
; (modulo -7 3) devolve 2 e (modulo 7 -3) devolve -2.
; (modulo 7 3) devolve 1 e (modulo -7 -3) devolve -1.
; devolve máximo divisor comum dos argumentos.
; devolve menor múltiplo comum dos argumentos.
; devolve #t, se x < 0, se não, #f.
; devolve #t, se x > 0, se não, #f.
; devolve #t, se x = 0, se não, #f.
; devolve #t, se x par, se não, #f.
; devolve #t, se x ímpar, se não, #f.
; devolve um inteiro pseudo-aleatório,
; situado entre 0 e n-1.
; não faz parte da definição do Scheme.
; em certas implementações, random não tem parâmetros.
```



## Processamento trigonométrico

(degrees->radians x)	; x em graus, devolve valor correspondente em radianos (não existe no DrScheme-FullScheme).
(radians->degrees x)	; x em radianos, devolve valor correspondente em graus (não existe no DrScheme-FullScheme).
(sin x)	; devolve seno de x, estando x em radianos. ; ( $\sin(\text{degrees}->\text{radians } 90)$ ) devolve 1.
(cos x)	; devolve cosseno de x, estando x em radianos.
(tan x)	; devolve tangente de x, estando x em radianos.
(acos x)	; devolve arco em radianos, cujo cosseno é x.
(asin x)	; devolve arco em radianos, cujo seno é x.
(atan x)	; devolve arco em radianos, cujo tangente é x. ; ( $\text{radians}->\text{degrees } (\text{atan } 1)$ ) devolve 45.0.

## Controlo de sequência

```

(if test-exp
    then-exp
    else-exp)
(if test-exp
    then-exp)
(cond (pred-1 exp1-1 exp1-2 ...)
      (pred-2 exp2-1 exp2-2 ...)
      ...
      (pred-n expn-1 expn-2 ...))
(cond (pred-1 exp1-1 exp1-2 ...)
      (pred-2 exp2-1 exp2-2 ...)
      ...
      (else e-else-1 e-else-2 ...))
(begin exp1 exp2 ...)

(case expressão
  ((chave1-1 chave1-2 ...) exp1-1 ...)
  ((chave2-1 chave2-2 ...) exp2-1 ...)
  ...
  (else exp-else-1 ...))

; (if (< 5 3)
;       (- 45 40)
;       (- 40 45)) devolve -5.
; (if (< 5 3)
;       (- 45 40)) nada acontece.
; (cond ((< n 3) (display "< 3"))
;       ((> n 4) (display "> 4")))
;       ...
;       ((> n 10) (display "> 10")))
; (cond ((< n 3) (display "< 3"))
;       ((> n 4) (display "> 4")))
;       ...
;       (else (display "outros")))
; calcula exp1, depois exp2, ...
; e devolve o valor da última expressão.
; (define vogal-ou-consoante-case
;   (lambda (letra)
;     (case letra
;       ((a e i o u) 'vogal)
;       (else 'consoante))))
;
; > (vogal-ou-consoante-case 'a)
; vogal
; > (vogal-ou-consoante-case 'f)
; consoante

```



# SCHEME

## na descoberta da programação

INTRODUÇÃO

1 - O ESSENCIAL DO SCHEME

2 - RECURSIVIDADE

3 - ABSTRACÇÃO DE DADOS

4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE

5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS

6 - SCHEME E OUTRAS  
TECNOLOGIAS

7-EXERCÍCIOS  
E  
PROJECTOS

ANEXOS

A B C D

## Entrada/Saída

```
(display arg) ; (begin (display "E' ")  
;           (display 5)  
;           (display " um numero par?"))  
; visualiza: E' 5 um numero par?  
; (display "Ele disse \"Ola'\".")  
; visualiza: Ele disse "Ola'".  
; (begin (display "E' ")  
;           (display 5)  
;           (newline)  
;           (display " um numero par?"))  
; E' 5  
; um numero par?  
; (let ((x (read)))  
; introduzindo 13 pelo teclado, x vale 13  
; mas introduzindo abcd, x vale abcd.
```

## Processamento de Pares e Listas

```
(cons x1 x2) ; constrói um par.  
; (cons 45 2) devolve par (45 . 2).  
(car x) ; selecciona e devolve o elemento da esquerda de um par.  
; (car (cons 45 2)) devolve 45.  
(cdr x) ; selecciona e devolve o elemento da direita de um par.  
; (cdr (cons 45 2)) devolve 2.  
(list x1 x2 x3 ...) ; constrói uma lista.  
; (list 1 2 3) devolve lista (1 2 3).  
; (list 1 (cons 34 2) 7) devolve (1 (34 . 2) 7).  
(cadr x) ; combina um cdr com um car.  
; (cadr (list 1 2 3)) devolve 2.  
; o Scheme disponibiliza os seguintes 28  
; procedimentos, que são composições de car e cdr:  
caar caaar cdadr caadar cadddr cddar caadr cddar  
caaddr cdaaar cddadr cdar cadar cdddr cadaar cdaadr cdddar  
cddr caddr caaaar cadadr cdadar cdddr cdaar caaadrr caddar cdaddr  
; (caddr x) equivalente a (car (cdr (cdr x))).  
(quote x) ; devolve argumento sem o processar. (equivalente a 'x)  
; (quote (1 2 3)) devolve lista (1 2 3).  
; '(1 2 3) devolve lista (1 2 3).  
; (cons elem lista) devolve lista composta por  
; elem e todos elementos de lista.
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

```
; (cons 1 '(4 all)) devolve lista (1 4 all).
; (car '((a b) c d)) devolve (a b).
; (cdr '(a . 1)) devolve 1.
; (cddr '(a ((b c) d) e)) devolve (e).
; (caadr '(a ((b c) d) e)) devolve (b c).
; devolve uma lista que inclui os elementos
; de list1, list2, ...
; (append (list 1 2) '(4 outros))
; devolve lista (1 2 4 outros)
; devolve comprimento de lista.
; (length (list 12 34 1 (list 1 2) 3)) devolve 5.
; devolve o elemento de lista na posição x.
; 1º elemento está na posição 0, 2º na posição 1, ...
; (list-ref (list 1 2 3 4) 2) devolve 3.
; devolve lista com elementos de lista em ordem inversa.
; (reverse (list 1 2 3 4)) devolve a lista (4 3 2 1).
; devolve a cauda de lista.
; se lista-1 for (1 2 3 4 5)
; (list-tail lista-1 2) devolve (3 4 5)
; aplica a operação op ao primeiro elemento das listas, depois ao segundo, ...
; (map + '(1 2 3) '(8 9 10)) devolve a lista (9 11 13).
; e devolve a lista resultante.
; (map add1 '(1 3 5 7)) devolve (2 4 6 8).
; (map (lambda (x) (+ 2 x)) '(1 3 5)) devolve (3 5 7).
; aplica a operação op a cada elemento da lista
; mas só interessam os efeitos laterais, como
; seja, a visualização.
; (for-each display '(1 3 5)) visualiza 135
; (for-each display '("pri" 35 "seg")) visualiza
; pri 35 seg
; aplica a operação op aos elementos de lista
; e devolve o valor resultante.
; (apply + '(1 2 3)) devolve 6.
; (map + '(1 2 3)) devolve (1 2 3).
; (apply + '(1 2 3 2 -2 1)) devolve 7.
; (apply max '(1 2 3 2 -2 1)) devolve 3.
; (apply min '(1 2 3 2 -2 1)) devolve -2.
; utiliza equal? para comparar x com os
; elementos de lista e devolve #f se lista não
; contém x, se não, devolve a sublistá que vai
; desde a ocorrência de x até fim de lista.
; (member 'a '(b c d e)) devolve #f.
; (member '(a) '(b (a) (b a))) devolve ((a) (b a)).
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

(assoc x lista)

; devolve o 1º elemento de lista que é uma lista e  
; cujo 1º elemento é x. Devolve #f, se x não existir.  
; (assoc 5 '((2 df) (6) (5 r t) (7 a b c)))  
; devolve (5 r t).  
; (assoc 6 '((2 df) (6) (7 a b c)))  
; devolve (6).

(null? x)

; devolve #t se x é uma lista vazia, se não, devolve #f.  
; (null? (list 1 2 3)) devolve #f.

(pair? x)

; (null? '()) devolve #t.  
; devolve #t, se x for um par, se não, devolve #f.  
; (pair? (cons 2 3)) devolve #t.  
; (pair? '()) devolve #f.

(set-car! par obj)

; obj substitui a parte esquerda de par.  
; (define lista '(a b c))

(set-cdr! var obj)

; (set-car! lista 'xyz)  
; lista passa a ser (xyz b c).

(set! var exp)

; obj substitui a parte direita de par.  
; (define lista '(a b c))

; (set-cdr! lista '(xyz))  
; lista passa a ser (a xyz).

; é atribuído a var, variável previamente  
; definida, o valor de exp.

; (define x 9)  
; (set! x (\* x x)) a variável x toma o valor 81.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

## Processamento de Caracteres e Cadeia de caracteres (strings)

```
#\B ; representa o carácter B.  
#\b ; representa o carácter b.  
#\7 ; representa o carácter 7.  
\space ; representa o carácter "espaço".  
\newline ; representa o carácter "nova linha".  
(char? cx) ; devolve #t se cx é carácter, se não, #f.  
 ; (char? #\5) devolve #t.  
 ; (char? 5) devolve #f.  
 ; (char->integer #\5) devolve 53.  
 ; (char->integer #\space) devolve 32.  
 ; (integer->char 37) devolve #\%.  
 ; devolve #t se os caracteres c1 e c2 forem iguais.  
 ; (char=? #\a #\A) devolve #f.  
 ; devolve #t se os caracteres c1 e c2  
 ; forem iguais, sem ter em conta se são  
 ; letras maiúsculas ou minúsculas.  
 ; (char-ci=? #\a #\A) devolve #t.  
 ; (char>? #\a #\A) devolve #t.  
 ; (char-ci>? #\a #\A) devolve #f.  
 ; devolve #t se c é letra maiúscula.  
 ; devolve #t se c é letra minúscula.  
 ; se c for letra, devolve c maiúscula, se não, devolve c.  
 ; se c for letra, devolve c minúscula, se não, devolve c.  
 ; devolve #t se c é uma das letras.  
 ; devolve #t se c é um dos dígitos decimais.  
 ; devolve #t se c for "espaço" ou "nova linha".  
 ; os argumentos são caracteres e devolve  
 ; uma cadeia de caractares.  
 ; (string #\a #\b #\c) devolve "abc".  
 ; devolve #t, se arg for cadeia de caracteres.  
 ; (string? "abc") devolve #t.  
 ; devolve comprimento de cadeia.  
 ; (string-length "This is a string") devolve 16.  
 ; (string-length "") devolve 0.  
 ; devolve uma cadeia igual ao argumento cad.  
 ; devolve uma cadeia com caracteres c, de comprimento num.  
 ; (make-string 3 #\a) devolve "aaa".  
 ; (make-string 3) devolve " ".
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

```
(string-append cad1 cad2 ...) ; devolve uma cadeia de caracteres,  
; concatenando as cadeias cad1, cad2 ...  
; (string-append "This is " "a string")  
; devolve "This is a string".  
; devolve o elemento da posição k de cadeia.  
; (string-ref "abcd 1234" 2) devolve #\c.  
; (string-ref "abcd 1234" 0) devolve #\a.  
; devolve uma sub-cadeia.  
; (substring "This is a string" 0 6) devolve "This i".  
; (substring "This is a string" 5 6) devolve "i".  
; constrói uma cadeia a partir de simbolo.  
; (symbol->string 'hello) devolve "hello".  
; constrói um símbolo a partir de cadeia.  
; (string->symbol "abc") devolve abc.  
; constrói uma cadeia a partir de uma lista de caracteres.  
; (list->string (list #\a #\b)) devolve "ab".  
; constrói uma lista a partir de cadeia  
; (string->list "ab") devolve (#\a #\b).  
; constrói uma cadeia a partir de numero.  
; (number->string 123) devolve "123".  
; constrói um número a partir de cadeia.  
; (string->number "abc") devolve #f.  
; (string->number "12345") devolve 12345.  
; (string=? "abc" "abc") devolve #t.  
; (string=? "abc" "ABC") devolve #f.  
; (string-ci=? "abc" "ABC") devolve #t.  
; (string-ci>? "abc" "ABC") devolve #t.  
; (string-ci>? "abc" "ABC") devolve #f.  
; a cadeia cad já existente, é preenchida com caracteres c.  
; o carácter da cadeia cad na posição  
; ind é substituído pelo carácter c.
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

## Processamento de Vectores

```
(vector obj1 obj2 ...)  
; constrói um vector com obj1, obj2, ...  
  
(vector? obj)  
; devolve #t, se obj é vector.  
  
(make-vector comp)  
; constrói um vector de comprimento comp,  
; em que os seus elementos são todos () .  
; (make-vector 3) devolve #(0 0 0).  
  
(make-vector comp elem)  
; constrói um vector de comprimento comp,  
; em que os seus elementos são todos elem.  
; (make-vector 3 'a) devolve #(a a a).  
; constrói um vector a partir da lista lis.  
; (list->vector '(1 6 a 7)) devolve #(1 6 a 7).  
; constrói uma lista a partir do vector vec.  
; (vector->list (vector 'ab 4)) devolve (ab 4).  
; devolve comprimento do vector vec.  
; (define v1 (vector 'a 6 'abc 90))  
; (vector-length v1) devolve 4.  
  
(vector-ref vec k)  
; devolve o elemento de índice k do vector vec.  
; (define v1 (vector 'a 6 'abc 90))  
; (vector-ref v1 1) devolve 6.  
; (vector-ref v1 2) devolve abc.  
; o vector vec existente, é preenchido com elem.  
; (define v-3-elementos (vector 1 2 3))  
; (vector-fill! vec-3-elementos "ac")  
; vec-3-elementos passa a ser #("ac" "ac" "ac").  
; modifica o vector vec, trocando o elemento  
; de índice k por elem.  
; (define v1 (vector 0 2 4 6 8))  
; v1 devolve #(0 2 4 6 8).  
; (vector-set! v1 2 5)  
; faz v1 igual a #(0 2 5 6 8).
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

## Processamento de Ficheiros

```
(load nome-fich) ; carrega o ficheiro designado por nome-fich
; e calcula as expressões nele contidas.
; se se tratar do ficheiro em c:\\expfnf\\lixo111.txt
; (load c:\\expfnf\\lixo111.txt)
; atenção à duplicação do carácter \
; pois o Scheme elimina um deles
; devolve uma porta de saída que fica
; associada ao ficheiro de saída nome-fich.
; (define f1 (open-output-file "c:\\expfnf\\lixo111.txt"))
; todas as chamadas de newline e display que
; refiram f1 vão para o ficheiro associado.
; (display "isto vai para o ficheiro" f1)
; (newline f1)
; (display 456 f1)

(open-output-file nome-fich 'append) ; uma variante de open-output-file, que permite
; acrescentar mais elementos a partir do fim
; do ficheiro
(open-output-file nome-fich 'replace) ; uma variante de open-output-file, que permite
; refazer o conteúdo de um ficheiro existente,
; limpando-o previamente

(close-output-port porta-s) ; (close-output-port porta-s) fecha
; o ficheiro de saída associado a porta-s.
(open-input-file nome-fich) ; devolve uma porta de entrada que fica
; associada ao ficheiro de entrada nome-fich.
; (define porta-e (open-input-file nome-ficheiro))
; todas as chamadas de read que refiram porta-e
; vão buscar dados ao ficheiro associado.
; (read porta-e)

(eof-object? ultimo-elem-lido) ; sempre que um ficheiro é acedido em
; leitura, através de read, o elemento
; lido deverá ser testado a fim de se
; verificar se já se atingiu o fim do ficheiro.
; (close-output-port porta-e) fecha o
; ficheiro de entrada associado a porta-e.

(close-input-port porta-e) ; verifica se o ficheiro nome-fich existe.
; elimina o ficheiro nome-fich.
(file-exists? nome-fich) ; (file-exists? "c:\\expfnf\\lixo111.txt")
; devolve #t se o ficheiro existir.
(delete-file nome-fich) ; (delete-file "c:\\expfnf\\lixo111.txt")
; (file-exists? "c:\\expfnf\\lixo111.txt")
; devolve #f
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

## Vários

```
(symbol? x)
(procedure? x)
(define (proc arg1 ...)
  exp1 exp2 ...)

; devolve #t, se x for símbolo, se não, devolve #f.
; devolve #t, se x procedimento, se não, devolve #f.
; equivalente a (define proc
;                 (lambda (arg1 ...)
;                   exp1 exp2 ...))

(let ((var1 init-exp1)
      (var2 init-exp2)
      ...
      )
  exp1 exp2 ...)

; define as variáveis var com o valor das
; expressões init-exp. Depois, no corpo de
; let, calcula as expressões exp
; e devolve valor da última.

(let* ((a 2)           ; a toma o valor 2
       (b 3))           ; e b o valor 3.
  (let ((a 4)           ; c toma o valor 2-3
        (c (- a b)))   ; e a o valor 4.
    (* c a)))          ; devolve -4.
; define, em sequência, as variáveis var com
; o valor das expressões init-exp. Ou seja,
; é possível utilizar numa expressão init-exp
; uma variável já definida no próprio let*.
; depois, no corpo de let, calcula as
; expressões exp e devolve valor da última.

(let* ((a 2)
       (b (* 2 a)))
  (+ a b))
; a toma o valor 2 e b 4. É devolvido 4.

(letrec ((var1 init-exp1)
         (var2 init-exp2)
         ...
         )
  exp1 exp2 ...)

; define as variáveis var com o valor das
; expressões init-exp. Estas definições
; também podem ser procedimentos recursivos
; ou mutuamente recursivos.
; depois, no corpo de letrec, calcula as
; expressões exp e devolve valor da última.

;
; (let ((a 2)           ; a toma o valor 2
;       (b 3))           ; e b o valor 3
;   (letrec ((soma-todos
;             (lambda (x)
;               (if (zero? x)
;                   0
;                   (+ x (soma-todos (- x 1)))))))
;     (+ a (soma-todos (- b 1))))))

; devolve 0.75.
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

(time exp)

```
;           (+ x
;           (soma-todos
;           (sub1 x))))))
;           (soma-todos (* a b))))
; calcula exp, visualiza em ms o tempo de cpu
; gasto no cálculo, o tempo real e o tempo
; de recolha de lixo (garbage collection) e,
; finalmente, devolve resultado do cálculo.
; (define fib
;   (lambda (n)
;     (if (< n 2)
;         n
;         (+ (fib (- n 1))
;             (fib (- n 2)))))))
; > (time (fib 31))
; cpu time: 3922 real time: 3953 gc time: 0
; 1346269
; >
; (current-inexact-milliseconds) ; devolve um positivo (não obrigatoriamente um inteiro)
; correspondente ao número de milisegundos que passaram
; desde uma certa data (normalmente desde que a máquina foi
; ligada). Este número nunca diminui enquanto a máquina
; permanecer ligada.
; exemplo de procedimento que implica uma certa espera,
; especificada em milisegundos...
; (define espera
;   (lambda (t-ms)
;     (let ((limite (+ (current-inexact-milliseconds)
;                      t-ms)))
;       (letrec ((ciclo
;                 (lambda ()
;                   (if (< (current-inexact-milliseconds)
;                           limite)
;                       (ciclo)
;                       (ciclo)))))))
```



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D



## Anexo B - Abstracção Janela gráfica

Os procedimentos gráficos disponibilizados pelas várias implementações da linguagem Scheme são, em geral, sofisticados e exigem alguma experiência para uma utilização correcta. Para os exercícios e projectos propostos que requeiram interacção gráfica, optou-se pela definição e implementação de um conjunto de procedimentos simples que funcionam numa janela gráfica. Trata-se de uma abstracção, a janela gráfica, que esconde os pormenores dos procedimentos gráficos do DrScheme, dando ao programador a hipótese de controlar, de uma forma simples, uma caneta que pode desenhar, pintar, ou escrever texto, com cores à escolha. O ponto da janela onde se encontra a caneta é o chamado ponto-corrente e as suas coordenadas e as do cursor comandado pelo rato podem ser lidas sempre que for necessário. Também está disponível a funcionalidade de leitura de tecla, logo que actuada.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

Tabela de cores utilizada				
índice	R	G	B	cor
0	0.0	0.0	0.0	preto
1	0.0	0.0	0.5	azul escuro
2	0.0	0.0	1.0	azul
3	0.0	0.5	0.0	verde escuro
4	0.0	0.5	0.5	cyan escuro
5	0.0	0.5	1.0	azul cyan
6	0.0	1.0	0.0	verde
7	0.0	1.0	0.5	verde cyan
8	0.0	1.0	1.0	cyan
9	0.5	0.0	0.0	vermelho escuro
10	0.5	0.0	0.5	magenta escuro
11	0.5	0.0	1.0	azul magenta
12	0.5	0.5	0.0	amarelo escuro
13	0.5	0.5	0.5	cinzento
14	0.5	0.5	1.0	azul cinza
15	0.5	1.0	0.0	verde amareulado
16	0.5	1.0	0.5	verde cinza
17	0.5	1.0	1.0	cyan pálido
18	1.0	0.0	0.0	vermelho
19	1.0	0.0	0.5	vermelho magenta
20	1.0	0.0	1.0	magenta
21	1.0	0.5	0.0	vermelho amareulado
22	1.0	0.5	0.5	vermelho cinza
23	1.0	0.5	1.0	magenta pálido
24	1.0	1.0	0.0	amarelo
25	1.0	1.0	0.5	amarelo pálido
26	1.0	1.0	1.0	branco

Esta abstracção esconde os pormenores dos procedimentos gráficos do DrScheme e ficará acessível com a inclusão, no seu código, de

```
(require (lib "swgr.scm" "user-feup"))
```

- (janela largura altura titulo)

Cria janela gráfica com a dimensão largura x altura e com um título, (cadeia de caracteres especificada por titulo).

Devolve uma lista com as características da janela, ou seja, a lista (largura altura titulo).

- (cor indice-ou-rgb)

De uma tabela de 27 cores, escolhe uma delas para cor da caneta, através de indice-ou-rgb, inteiro situado entre 0 e 26. Se indice-ou-rgb = 'temp (temporário), o efeito de uma primeira passagem da caneta é anulado por uma segunda passagem.

É com esta cor que se deve desenhar ou pintar um objecto animado, sem se perder os gráficos visualizados e sobre os quais o objecto se desloca. Para tal, bastará desenhar o objecto animado com a cor 'temp e, quando ele sair, deverá ser novamente desenhado com a cor 'temp, no local onde se encontra, reaparecendo o que ele tapava.



# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

A B C D



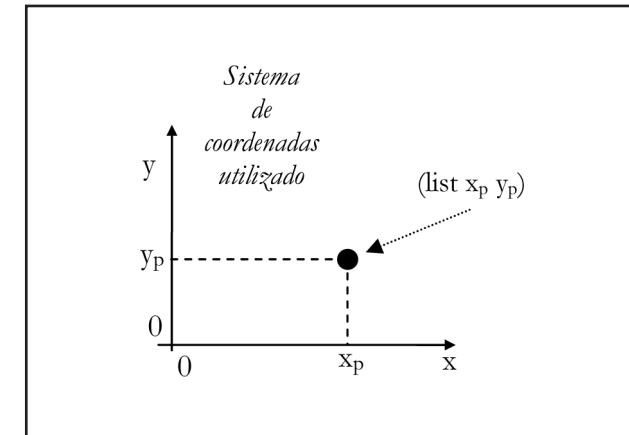
Ao parâmetro `índice-ou-rgb` também se podem associar três valores entre 0.0 e 1.0, que traduzem, respectivamente, os pesos R, G e B das cores Vermelho (*Red*), Verde (*Green*) e Azul (*Blue*). Por exemplo, com R=G=1.0 e B a variar de 0.0 a 1.0, codificam as cores desde o amarelo, até ao branco, passando pelo amarelo pálido.

- `(move pt)`

Move a caneta, sem desenhar, para o ponto definido por `pt` (`pt` é uma lista com as coordenadas `x` e `y` do ponto).

- `(move-rel pt)`

Procede como `move`, mas o ponto `pt` é definido em relação ao ponto corrente. Sendo `Px` e `Py` as coordenadas do ponto corrente, e o argumento `pt` uma lista com `dx` e `dy`, a caneta é movida, sem desenhar, para o ponto definido pela lista `(Px + dx Py + dy)`.



- `(le-cor ponto)`

Devolve uma lista com os coeficientes RGB da cor lida no pixel da janela referido por `ponto`.

- `(le-índice-cor ponto)`

Devolve índice (da tabela de cores) da cor lida no pixel da janela referido por `ponto`.

Devolve -1 se não está na tabela de cores.

- `(desenha sequencia-de-pontos)`

Sendo `sequencia-de-pontos` uma lista de pontos, desenha a linha poligonal definida pelos pontos em `sequencia-de-pontos`. A caneta fica sobre o último ponto da sequência.

- `(desenha-oval dois-pontos)`

Sendo `dois-pontos` uma lista de dois pontos, desenha a oval inscrita no rectângulo cujo vértice superior-esquerdo é o primeiro ponto da lista `dois-pontos` e o vértice inferior-direito é o segundo ponto. No final, a caneta retoma a posição que tinha à partida.

# SCHEME na descoberta da programação

## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

A B C D

- (pinta sequencia-de-pontos)

Procede como desenha, mas preenche com a cor da caneta todo o interior do polígonos definido por sequencia-de-pontos. Se o último ponto de sequencia-de-pontos não coincidir com o primeiro, o polígonos é fechado ligando o último ponto com o primeiro, ficando, no entanto, a caneta no último ponto.

- (pinta-oval dois-pontos)

Procede como desenha-oval, mas preenche com a cor da caneta o interior da oval.

- (ponto pt)

Procede como move, mas pinta com a caneta o ponto definido por pt e, no final, a caneta fica sobre esse ponto.

- (desenha-rel sequencia-de-pontos)

Procede como desenha, mas cada ponto em sequencia-de-pontos é definido relativamente ao ponto onde se encontra a caneta, antes desta começar a mover-se na sua direcção. No final, a caneta ficará sobre o último ponto da sequência.

- (desenha-oval-rel dois-pontos)

Procede como desenha-oval, mas os pontos da lista dois-pontos são definidos relativamente ao ponto corrente.

- (pinta-rel sequencia-de-pontos)

Procede como pinta, mas cada ponto da sequencia-de-pontos é definido relativamente ao ponto onde se encontra a caneta, antes desta se começar a mover na sua direcção.

- (ponto-rel pt)

Procede como ponto, mas pt é relativo ao ponto corrente.

- (desenha-txt texto)

A partir do ponto corrente, escreve a cadeia de caracteres texto e deixa a caneta no canto inferior direito do espaço onde for escrito o último carácter.

- (caneta)

Devolve a posição actual da caneta, ou seja, o ponto corrente.



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

- (cursor)

Devolve o ponto do interior da janela apontado pelo cursor, quando é actuado o botão esquerdo do rato.

- (limpa)

Limpa a janela.

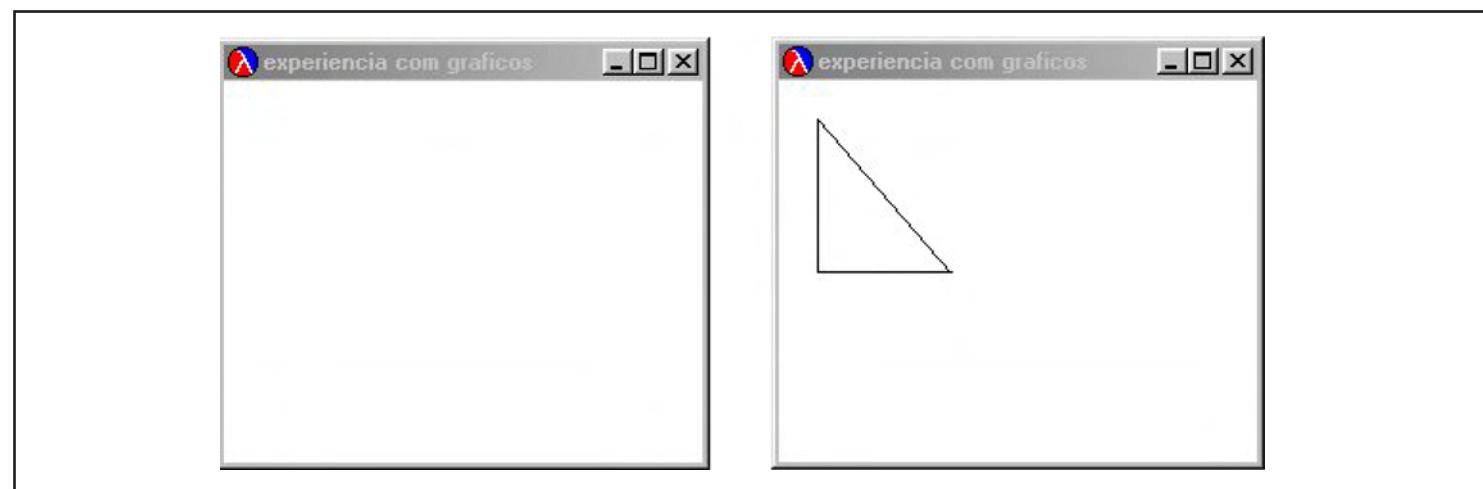
- (tecla-pressionada espera)

Se espera for #t, aguarda que uma tecla seja activada e devolve o respectivo código.

- (fecha-janela)

Fechá a janela gráfica.

```
> (require (lib "swgr.scm" "user-feup"))
> (janela 250 200 "experiencia com graficos")
(250 200 "experiencia com graficos")
> (desenha (list '(20 100) '(20 180) '(90 100) '(20 100)))
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

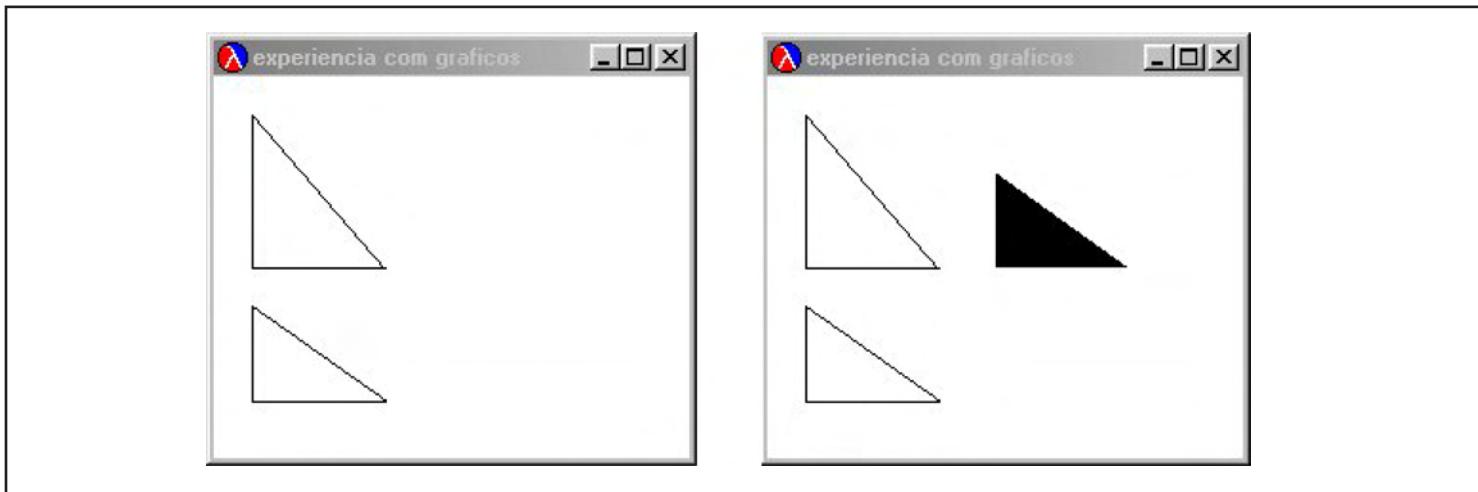
### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

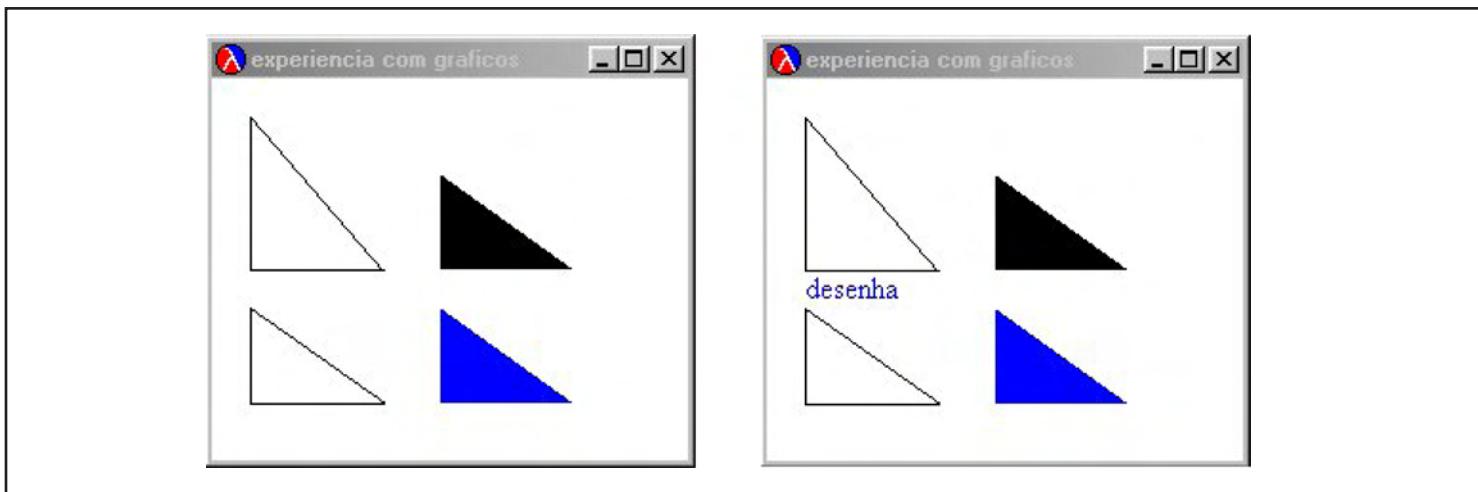
### ANEXOS

A B C D

```
> (move (list 20 30))
> (desenha-rel (list '(0 0) '(0 50) '(70 -50) '(-70 0)))
> (move (list 120 100))
> (pinta-rel (list '(0 0) '(0 50) '(70 -50) '(-70 0))))
```



```
> (cor 2)
> (pinta '((120 30) (120 80) (190 30)))
> (move '(20 85))
> (desenha-txt "desenha")
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

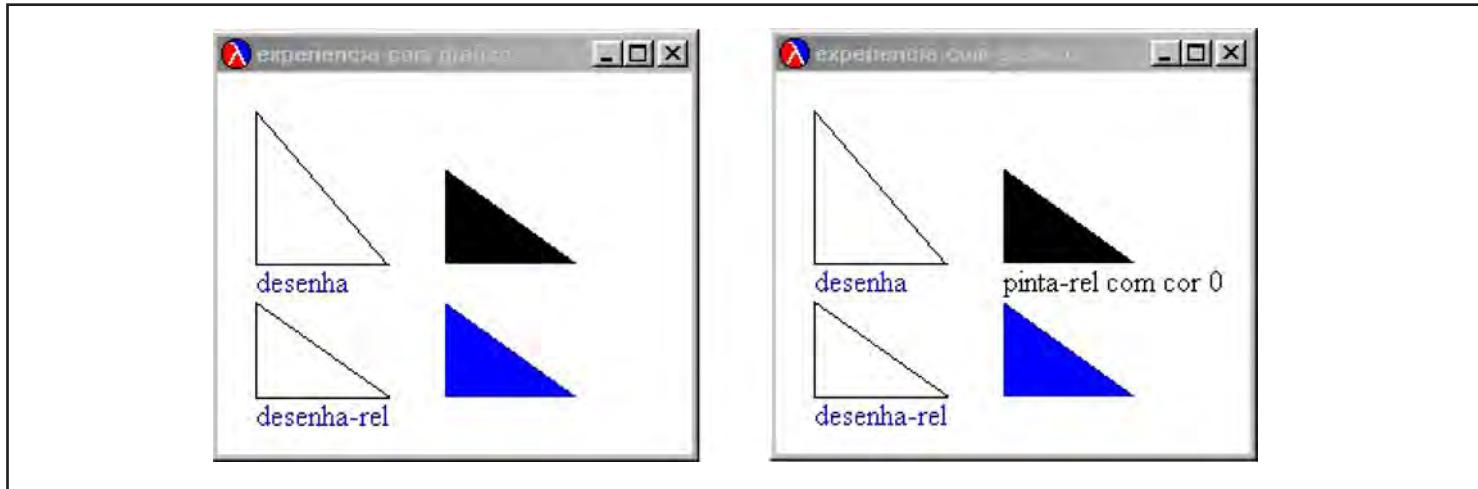
### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

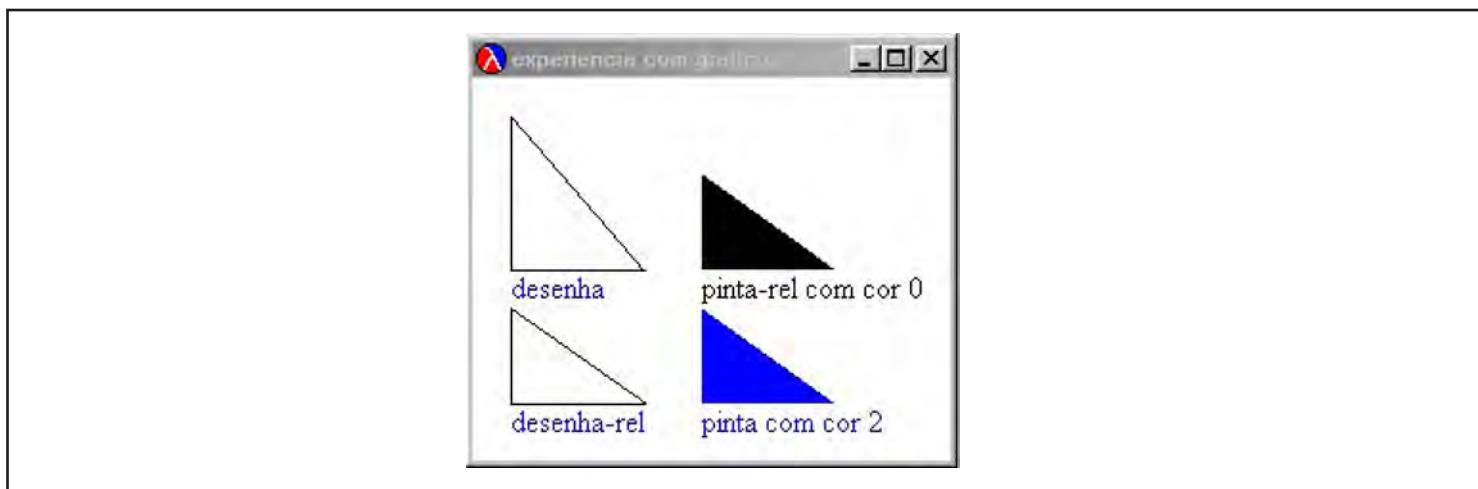
### ANEXOS

A B C D

```
> (move '(20 15))
> (desenha-txt "desenha-rel")
> (move '(120 85))
> (cor 0)
> (desenha-txt "pinta-rel com cor 0")
```



```
> (move (list 120 15))
> (cor 2)
> (desenha-txt "pinta com cor 2")
```



## INTRODUÇÃO

## 1 - O ESSENCIAL DO SCHEME

## 2 - RECURSIVIDADE

## 3 - ABSTRACÇÃO DE DADOS

## 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

## 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

## 6 - SCHEME E OUTRAS TECNOLOGIAS

## 7-EXERCÍCIOS E PROJECTOS

## ANEXOS

A B C D



## Anexo C - Reutilização de código

O DrScheme tem vindo a oferecer esquemas simples, mas suficientemente flexíveis e poderosos, utilizados na reutilização de código. Utilizar um código previamente desenvolvido e testado é uma atitude normal, que se aconselha, especialmente quando se enfrentam problemas que exigem programas de envergadura média ou acima da média.

Pode ver, neste anexo, como estruturar um código desenvolvido para ser posteriormente reutilizado, por si ou por outros, e também recolher ideias sobre algumas abstracções interessantes que surgiram no decorrer dos nossos trabalhos e que foram colocadas num dos directórios do DrScheme: PLT\collects\user-feup.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D



Em versões mais antigas do DrScheme, era necessário começar por colocar os ficheiros, com o código a reutilizar, num directório criado em `PLT\collects\`. Se esse directório se designasse, por exemplo, `user-feup`, e se fosse `tartaruga.scm` o nome de um desses ficheiros, bastaria escrever, no início do programa

```
(require-library "tartaruga.scm" "user-feup")
```

para usufruir de todos os procedimentos e outras entidades incluídos naquele ficheiro.

Em versões mais recentes do DrScheme, o esquema oferecido é um pouco mais sofisticado, pois permite descriminar, entre aqueles ficheiros, as partes que interessam, mantendo tudo o resto escondido e inacessível. É assim possível controlar com rigor as entidades que podem ser reutilizadas, escondendo, por exemplo, procedimentos auxiliares que não interessa divulgar, o que contribui para minimizar a ocorrência de conflitos de nomes.

Nestas versões mais recentes, os ficheiros a reutilizar são também colocados num directório de `PLT\collects\`. A requisição das entidades faz-se também de uma forma muito parecida com a anterior e quase se confundem. Assim, se o directório a criar continuar a ser designado por `user-feup` e se o nome do ficheiro for `tartaruga.scm`, bastará agora escrever, no início do programa

```
(require (lib "tartaruga.scm" "user-feup"))
```

para usufruir não de todas, como acontecia em versões mais antigas do DrScheme, mas apenas de determinadas entidades daquele ficheiro. A principal diferença reside na forma como o ficheiro é constituído, pois deverá ser organizado como um módulo no qual são claramente especificadas as entidades a disponibilizar.

Sugere-se agora a análise cuidada do exemplo que se segue, em especial as características que se destacam:

- `eixos1` é o nome do módulo, que é seguido por `mzscheme`.
- dos vários procedimentos do módulo, apenas o procedimento `eixos` é disponibilizado, ficando os restantes escondidos.
- supondo ser `eixos1.scm` o nome do ficheiro com este módulo, colocado no directório `user-feup`, a sua reutilização ou, mais especificamente, a reutilização do procedimento `eixos` é possível escrevendo no início do programa

```
(require (lib "eixos1.scm" "user-feup"))
```

# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

```
(module eixos1
  mzscheme

  (define eixos
    (lambda (origem escala eixo-xx eixo-yy)
      ... ; definição do procedimento eixos

  (define origem-eixos
    (lambda (sistema-de-eixos)
      ... ; definição do procedimento origem-eixos

  (define escala-eixos
    (lambda (sistema-de-eixos)
      ... ; definição do procedimento escala-eixos

  (define num-divisorias
    (lambda (sistema-de-eixos)
      ... ; definição do procedimento num-divisorias

  (provide eixos))
```

pormenor de sintaxe na  
criação de um módulo

disponibilizado apenas o  
procedimento eixos

No exemplo que se segue, o módulo é designado por `eixos2` e supõe que o próprio módulo em definição requeira entidades de outro módulo, colocado no directório `user-feup`, no ficheiro com o nome `swgr.scm`.

```
(module eixos2
  mzscheme

  (require (lib "swgr.scm" "user-feup"))

  (define eixos
    (lambda (origem escala eixo-xx eixo-yy)
      ... ; definição do procedimento eixos

  (define origem-eixos
    (lambda (sistema-de-eixos)
      ... ; definição do procedimento origem-eixos

  (define escala-eixos
    (lambda (sistema-de-eixos)
      ... ; definição do procedimento escala-eixos

  (define num-divisorias
    (lambda (sistema-de-eixos)
      ... ; definição do procedimento num-divisorias
```

reutilização de `swgr.scm`  
no módulo `eixos2`.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

(provide eixos)

; a entidade janela é suposta  
; disponível em "swgr.scm"  
(provide janela))

As entidades disponibilizadas por swgr.scm, apesar de estarem acessíveis em eixos2, não são automaticamente disponibilizadas para outros módulos que venham a requerer eixos2. No entanto, entre as entidades disponibilizadas por eixos2 podem também ser incluídas as que requereu a swgr.scm, situação ilustrada com o procedimento janela.

### O directório PLT\collects

No decorrer dos nossos trabalhos, surgiram algumas abstracções interessantes que foram colocadas em PLT\collects\, num directório criado para esse efeito, com a designação user-feup. As duas primeiras abstracções que tiveram este destinado foram a abstracção tartaruga (ficheiro tartaruga.scm) e a abstracção que simula os deslocamentos e calcula as distâncias entre tartarugas (ficheiro simula-desloca.scm).

Assim, bastará escrever no início do seu programa

```
(require (lib "tartaruga.scm" "user-feup"))
```

para usufruir das operações

```
faz-tartaruga, linha-tar, coluna-tar, cor-tar, visu-tar
```

e escrever

```
(require (lib "simula-desloca.scm" "user-feup"))
```

para usufruir das operações

```
vai-para-norte, vai-para-sul, vai-para-oeste, vai-para-este,  
distancia-horizontal, distancia-vertical, distancia.
```



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

Apenas por curiosidade, junta-se agora o código das abstracções tartaruga e simulação de deslocamentos, colocadas em PLT\collects\user-feup.

```
(module tartaruga
  mzscheme

  (define faz-tartaruga          ; Três parâmetros, inteiros positivos.
    (lambda (col lin cor)         ; Devolve uma tartaruga situada em col
      (cons cor (cons col lin)))) ; e lin, com a cor dada. É um construtor.

  (define linha-tar              ; parâmetro do tipo tartaruga. Devolve um inteiro
    (lambda (tart)                ; positivo que identifica a linha onde se
      (cdr (cdr tart))))        ; encontra a tartaruga. É um selector.

  (define coluna-tar             ; parâmetro do tipo tartaruga. Devolve um inteiro
    (lambda (tart)                ; positivo que identifica a coluna onde se
      (car (cdr tart))))        ; encontra a tartaruga. É um selector.

  (define cor-tar                ; Um parâmetro do tipo tartaruga. Devolve um
    (lambda (tart)                ; inteiro positivo que identifica a cor da
      (car tart)))               ; tartaruga. É um selector.

  (define visu-tar
    (lambda (tart)
      (display "tartaruga em col: ")
      (display (coluna-tar tart))
      (display ", lin: ")
      (display (linha-tar tart))
      (display ", cor: ")
      (display (cor-tar tart))
      (newline)))

  (provide faz-tartaruga)
  (provide linha-tar)
  (provide coluna-tar)
  (provide cor-tar)
  (provide visu-tar))
```



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

A abstracção que simula os deslocamentos das tartarugas requer a abstracção tartaruga.

```
(module simula-desloca
  mzscheme

  (require (lib "tartaruga.scm" "user-feup"))

  (define vai-para-norte           ; deslocamento para Norte de n posições.
    (lambda (tart n)
      (let ((destino (- (linha-tar tart)
                         n)))
        (faz-tartaruga (coluna-tar tart)
                        (if (< destino 1)
                            1
                            destino)
                        (cor-tar tart)))))

  (define vai-para-sul           ; deslocamento para Sul de n posições.
    (lambda (tart n)
      (faz-tartaruga (coluna-tar tart)      ; neste caso, não há limitações ao
                      (+ (linha-tar tart) n)      ; deslocamento.
                      (cor-tar tart)))))

  (define vai-para-oeste          ; ver comentários do procedimento
        ; vai-para-norte.
    (lambda (tart n)
      (let ((destino (- (coluna-tar tart)
                         n)))
        (faz-tartaruga (if (< destino 1)
                           1
                           destino)
                        (linha-tar tart)
                        (cor-tar tart)))))

  (define vai-para-este           ; ver comentários do procedimento
        ; vai-para-sul.
    (lambda (tart n)
      (faz-tartaruga (+ (coluna-tar tart) n)
                     (linha-tar tart)
                     (cor-tar tart)))))

  (define distancia-horizontal
    (lambda (tart1 tart2)
      (abs (- (coluna-tar tart1)
               (coluna-tar tart2))))))

  (define distancia-vertical
    (lambda (tart1 tart2)
      (abs (- (linha-tar tart1)
               (linha-tar tart2))))))

  ; a distância horizontal entre tart1 e tart2
  ; é dada pelo valor absoluto da diferença
  ; entre as colunas de tart1 e tar2
```



# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1ª CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

```
(define distancia
  (lambda (tart1 tart2)
    (+ (distancia-horizontal tart1 tart2) ; este procedimento recorre aos dois
        (distancia-vertical tart1 tart2)))) ; procedimentos anteriores

(provide vai-para-norte)
(provide vai-para-sul)
(provide vai-para-oeste)
(provide vai-para-este)
(provide distancia-horizontal)
(provide distancia-vertical)
(provide distancia))
```

Requerendo as duas abstracções, isoladamente, ficam acessíveis, obviamente, os recursos disponibilizados por ambas.

```
(require (lib "simula-desloca.scm" "user-feup"))
(require (lib "tartaruga.scm" "user-feup"))

> (define tar1 (faz-tartaruga 10 20 3))
> (define tar2 (vai-para-norte tar1 2))
> tar1
(3 10 . 20)
> tar2
(3 10 . 18)
```

### O que poderá encontrar no directório PLT\collects\user-feup

Das abstracções que foram desenvolvidas no decorrer dos nossos trabalhos, colocadas em PLT\collects\user-feup, salientam-se:

- (require (lib "audio.scm" "user-feup"))

A abstracção audio permite a introdução de efeitos sonoros. Foi utilizada na Parte 1, no módulo - Introdução à programação através de abstracções.

Caso não possua equipamento para escutar os sons reproduzidos no seu computador ou se não quiser incomodar as pessoas que estejam próximas, substitua

```
(require (lib "audio.scm" "user-feup"))
por
(require (lib "noaudio.scm" "user-feup"))
```

para "ver" no ecrã os sons reproduzidos...



# S C H E M E

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

- (require (lib "box-cont-coll-stack.scm"))

Esta abstracção permite a utilização das classes my-box, contador, my-collection e my-stack. Foi desenvolvida e utilizada na Parte 6, no Módulo - A programação OO com o Scheme.

- (require (lib "conjuntos.scm"))

Esta abstracção permite a criação de conjuntos e dá acesso às respectivas operações. Foi desenvolvida na Parte 3 e utilizada no Módulo - Um jogo para testar a memória.

- (require (lib "naves.scm"))

Esta abstracção baseia-se na abstracção tabuleiro e permite a criação e deslocação de naves. Foi muito utilizada na Parte 1, no Módulo - Introdução à programação através de abstracções.

- (require (lib "swgr.scm"))

Esta abstracção permite a criação de janelas gráficas em que uma caneta desenha e pinta pontos, rectas e elipses, e também desenha texto. Aparece na Parte 3, no Módulo - Utilizar e construir abstracções para reutilizar código e é o tema do Anexo B.

- (require (lib "tabuleiro.scm"))

Esta abstracção baseia-se na abstracção janela gráfica e permite a criação de tabuleiros para jogos ou para pistas onde se deslocam naves. Aparece na Parte 3, no Módulo - Utilizar e construir abstracções para reutilizar código e é o tema do Anexo D.

- (require (lib "varios.scm"))

Aqui encontra uma mistura de procedimentos utilitários, dos quais se distinguem:

- pi
- roleta-1-n
- degrees->radians
- radians->degrees
- espera

Introduz uma paragem especificada pelo argumento, em milissegundos.



# **S C H E M E**

## **na descoberta da programação**

**INTRODUÇÃO**

**1 - O ESSENCIAL DO SCHEME**

**2 - RECURSIVIDADE**

**3 - ABSTRACÇÃO DE DADOS**

**4 - PROCEDIMENTOS COMO  
OBJECTOS DE 1<sup>a</sup> CLASSE**

**5 - ABSTRACÇÕES COM  
DADOS MUTÁVEIS**

**6 - SCHEME E OUTRAS  
TECNOLOGIAS**

**7-EXERCÍCIOS  
E  
PROJECTOS**

**ANEXOS**

**A B C D**



## **Anexo D - Abstracção Tabuleiro**

Para responder a situações que fazem uso de um tabuleiro visualizado no ecrã, como acontece com o jogo de damas e labirintos, desenvolveu-se a abstracção tabuleiro.

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

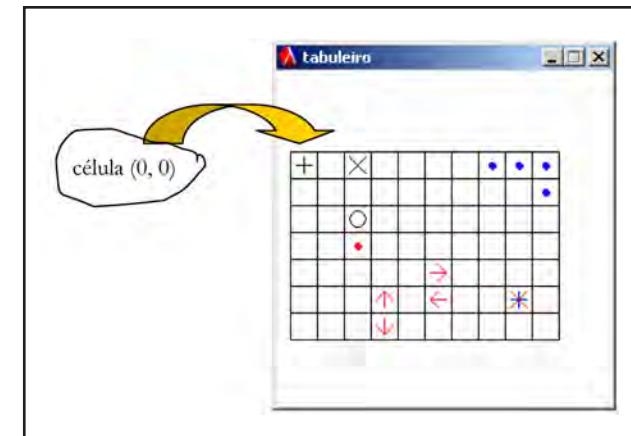
### ANEXOS

A B C D



## Abstracção tabuleiro

Para responder a situações que fazem uso de um tabuleiro visualizado no ecrã, como acontece com o jogo de damas e labirintos, desenvolveu-se a abstracção tabuleiro. Um tabuleiro é criado e visualizado com  $nx * ny$  células quadradas e o conteúdo visual destas células pode ser alterado, colocando em cada uma delas um símbolo à escolha num conjunto de vários símbolos gráficos. Cada célula é identificada por dois índices, situados entre 0 e  $nx - 1$  e 0 e  $ny - 1$ . A célula colocada na parte superior-esquerda correspondem os índices 0 e 0.



Esta abstracção esconde os pormenores dos procedimentos gráficos do DrScheme e permite este tipo de visualização.

Para a utilizar, deverá incluir no seu código

```
(require (lib "tabuleiro.scm" "user-feup"))
```

e abrir uma janela gráfica, suficientemente ampla para conter o tabuleiro.

Desta abstracção, fazem parte vários procedimentos.

- `(tabuleiro org-x org-y lado num-cel-x num-cel-y)`

em que `org-x` e `org-y` representam a origem do vértice superior-esquerdo do tabuleiro, em coordenadas da janela, `lado` é o comprimento do lado de cada célula em unidades do ecrã e `num-cel-x` e `num-cel-y` representam o número de células, na horizontal e na vertical, respectivamente.

Visualiza um tabuleiro com as características especificadas e devolve uma lista com essas características, ou seja, a lista `(org-x org-y lado num-cel-x num-cel-y)`. Trata-se de um construtor.

- `(cel-x tab)`
- `(cel-y tab)`

em que `tab` é um tabuleiro.

Devolvem, respectivamente, o número de células na horizontal e o número de células na vertical do tabuleiro. São ambos selectores.

# SCHEME

## na descoberta da programação

### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D

- (celula tab i-x i-y simb cor-actual)

em que tab é um tabuleiro, i-x e i-y são os índices que identificam uma célula, simb é um símbolo e cor-actual é uma cor (conforme a especificação de cor usada na **abstracção janela gráfica**).

Trata-se de um modificador, pois altera o aspecto visual de uma célula, desenhando nela um símbolo à escolha, de acordo com a seguinte codificação:

+	desenha uma cruz +	x	desenha uma cruz x
c	desenha um círculo	p	desenha um ponto
n	desenha uma seta para norte	s	desenha uma seta para sul
e	desenha uma seta para este	o	desenha uma seta para oeste
l	limpa a célula		

- (celulas tab lis simb espera cor-actual)

em que tab é um tabuleiro, lis é uma lista de células, cada uma das especificada por um par cujos elementos são os índices da célula, simb é um símbolo, espera é um valor inteiro e cor-actual é uma cor (conforme a especificação de cor usada na abstracção janela gráfica).

As células especificadas por lis são alteradas visualmente com o símbolo simb, a um ritmo condicionado por espera (espera expressa o tempo em milissegundos entre a visualização de células). Trata-se de um modificador.

- (limpa-tab tab cor-actual)

em que tab é um tabuleiro e cor-actual é uma cor (conforme a especificação de cor usada na abstracção janela gráfica).

Todas as células do tabuleiro tab são pintadas com a cor especificada. Trata-se de um modificador.

- (indice-celula tab i-x i-y)

em que tab é um tabuleiro e i-x e i-y são os índices que identificam uma célula.

Devolve o índice (da tabela de cores) da cor lida no ponto central da célula.

Devolve -1 se a essa cor não estiver na tabela de cores. Trata-se de um selector.



### INTRODUÇÃO

### 1 - O ESSENCIAL DO SCHEME

### 2 - RECURSIVIDADE

### 3 - ABSTRACÇÃO DE DADOS

### 4 - PROCEDIMENTOS COMO OBJECTOS DE 1<sup>a</sup> CLASSE

### 5 - ABSTRACÇÕES COM DADOS MUTÁVEIS

### 6 - SCHEME E OUTRAS TECNOLOGIAS

### 7-EXERCÍCIOS E PROJECTOS

### ANEXOS

A B C D



A abstracção gráfica também disponibiliza os procedimentos:

- (janela largura altura titulo)

Cria uma janela gráfica com a dimensão largura x altura e com um título (cadeia de caracteres especificada por titulo).

Devolve uma lista com as características da janela, ou seja, a lista (largura altura titulo).

- (limpa)

Limpa a janela.

O tabuleiro da figura anterior foi criado numa sessão equivalente à que se segue.

```
> (require (lib "tabuleiro.scm" "user-feup"))
> (janela 250 250 "tabuleiro")
(250 250 "tabuleiro")

> (define tab1 (tabuleiro 10 190 20 10 7))
> (cel-x tab1)
10
> (cel-y tab1)
7
> (celula tab1 0 0 '+ 0)      visualiza + a preto, na célula 0,0;
> (celula tab1 2 0 'x 0)      visualiza x a preto, na célula 2,0;
> (celula tab1 3 0 'X 0)      símbolo não previsto... erro;
erro: comando desconhecido!
> (celula tab1 2 2 'c 0)      visualiza 0 a preto, na célula 2,2;
> (celula tab1 2 3 'p 18)      visualiza . a vermelho, na célula 2,3;
> (celula tab1 3 5 'n 18)      visualiza seta para Norte a vermelho, na célula 3,5;
> (celula tab1 3 6 's 18)      visualiza seta para Sul a vermelho, na célula 3,6;
> (celula tab1 5 4 'e 18)      visualiza seta para Este na célula 5,4;
> (celula tab1 5 5 'o 18)      visualiza seta para Oeste na célula 5,5;
> (celula tab1 8 5 '+ 2)       visualiza caracteres sobrepostos
                               na célula 8,5;

> (celula tab1 8 6 '+ 2)       visualiza + a azul, na célula 8,6
                               para limpar (pintar a célula a branco) em seguida;
                               vai visualizar uma sequência de 4 pontos azuis,
                               perto do canto superior-direito, ao ritmo de 1 por segundo
> (celulas tab1 (list (cons 7 0) (cons 8 0) (cons 9 0) (cons 9 1)) 'p 1000 2)
```

Esta abstracção baseia-se na abstracção janela gráfica ([Anexo B](#)) e desta disponibiliza os procedimentos janela e limpa, com os quais é possível abrir e limpar janelas gráficas.