Cap. 5 - Informação Simbólica

INF05008 - Fundamentos de Algoritmos

Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre, Brasil
http://www.inf.ufrgs.br

Nomeando Valores

- Quando um **valor ocorre muitas vezes** em um programa, é interessante **atribuir-lhe um nome**
- Exemplo: o valor de π é nomeado da seguinte forma:

```
(define PI 3.14)

(define (área-do-disco r)
    (* PI (* r r)))
```

- Para usar uma aproximação melhor de π, não precisamos buscar todas as ocorrências, **basta mudarmos o valor associado a ele na definição**

Informação Simbólica

- Computadores processam informações **simbólicas** tais como nomes, palavras, direções, imagens, etc.
- **Linguagens de programação** têm suporte para pelo menos uma **forma de representar informação simbólica**
- Racket tem suporte para
  - Símbolos
  - Strings
  - Caracteres (do teclado)
  - Imagens

Símbolos

- Um **símbolo** é uma **sequência de caracteres** do teclado precedida por um apóstrofe
- Exemplos:

  `'Ana  'a  'gato!  'dois^3  'entre%outros?`
- **Interpretação** do símbolo depende do **usuário** e do **contexto**
- Assim como os números, símbolos são dados **atômicos**

Símbolos (cont.)

- **Operação básica** (de comparação) sobre símbolos: **`symbol=?`**
- Exemplos de expressões com `symbol=?`:
  - `(symbol=? 'Hello 'Hello)` é **verdadeiro**
  - `(symbol=? 'Hello 'Howdy)` é **falso**
  - `(symbol=? 'Hello x)` é **verdadeiro** se x for `'Hello` e **falso**, caso x seja qualquer **outro símbolo**
- Exercício: qual é o **contrato** de `symbol=?` ?

Usando Símbolos

- *Função 'resposta' que responde com um comentário às seguintes saudações: "Bom dia", "Tudo bem?", "Boa tarde" e "Boa noite".*
- *Cada uma dessas saudações pode ser representada como um Símbolo:* `'BomDia`, `'TudoBem?`, `'BoaTarde` e `'BoaNoite`.
- Contrato, objetivo e cabeçalho:

```
;; resposta : Símbolo  ->  Símbolo
;; Dada uma saudação (um símbolo),
;; determina resposta (outro símbolo) para ela

(define (resposta s) ...)
```

Usando Símbolos

- Contrato, objetivo e cabeçalho:

```
;; resposta : Símbolo  ->  Símbolo
;; Dada uma saudação (um símbolo),
;; determina resposta (outro símbolo) para ela

(define (resposta s) ...)
```

- Exemplos:

```
;; (resposta 'BomDia) retorna 'Oi
;; (resposta  'TudoBem?)  retorna  'Tranquilo
;; (resposta  'BoaTarde) retorna 'PrecisoDormir
;; (resposta 'BoaNoite) retorna 'EstouMuitoCansado
```

Usando Símbolos (cont.)

- A função deve distinguir entre quatro situações. De acordo com a estrutura de projeto, temos uma expressão condicional com quatro cláusulas:

```
(define (resposta s)
 (cond
   [(symbol=? s 'BomDia)    ...]
   [(symbol=? s 'TudoBem?)     ...]
   [(symbol=? s 'BoaTarde) ...]
   [(symbol=? s 'BoaNoite)    ...]))
```

Usando Símbolos (cont.)

- A partir do *template*, a função final é facilmente definida. Eis uma possibilidade:

```
(define (resposta s)
 (cond
   [(symbol=? s 'BomDia)    'Oi]
   [(symbol=? s 'TudoBem?) 'Tranquilo]
   [(symbol=? s 'BoaTarde) 'PrecisoDormir]
   [(symbol=? s 'BoaNoite) 'EstouMuitoCansado]))
```

Strings

- **Strings** são sequências de caracteres do teclado entre aspas duplas
- Exemplos:

```
"o cão"       "pois é"      "talvez"
"chocolate"    "dois^3"      "e etc."
```

- **Operação básica**: **`string=?`** compara duas *strings*
- *Strings* **não são atômicas** (mais sobre isso em listas...)

Caracteres

- **Caracteres** também são representados em Racket

```
#\a, #\b ... #\z

#\A, #\B ... #\Z

#\0, #\1 ... #\9

#\+, #\-, #\$, #\!, #\space, ...
```

- **Operação básica**: **`char=?`** compara caracteres

Imagens

- Racket também aceita **imagens** como informação simbólica
  - Imagens são **valores**, assim como números e booleanos
  - Podem ser usadas **em qualquer expressão**
  - Para facilitar a referência, é usual **nomeá-las**
  - Pode-se usar a **operação `image=?`** para **comparar imagens**

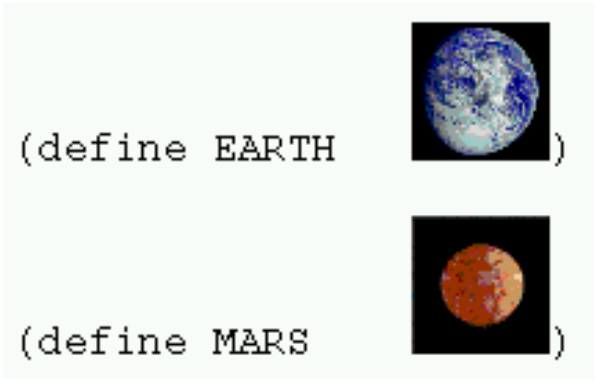

figura 7, página 47 de HTDP e exercício 5.5.1

Exercícios

- Qual é o resultado de cada expressão?

```
(define a 2)
(define b 4)

(define Castelo [Castelo-img])
(define Palácio [Castelo-img])

(+ 'a a)

(+ 'a 'b)

(+ a (sqr b))
```

Exercícios

- Qual é o resultado de cada expressão?

```
(define a 2)
(define b 4)

(define Castelo [Castelo-img])
(define Palácio [Castelo-img])

(+ 'a a) => ERRO! Não é possível somar um
símbolo a um número

(+ 'a 'b)

(+ a (sqr b))
```

Exercícios

- Qual é o resultado de cada expressão?

```
(define a 2)
(define b 4)

(define Castelo [Castelo-img])
(define Palácio [Castelo-img])

(+ 'a a) => ERRO! Não é possível somar um
símbolo a um número

(+ 'a 'b) => ERRO! Operação aplicável apenas
a números

(+ a (sqr b))
```

Exercícios

- Qual é o resultado de cada expressão?

```
(define a 2)
(define b 4)

(define Castelo [Castelo-img])
(define Palácio [Castelo-img])

(+ 'a a) => ERRO! Não é possível somar um
símbolo a um número

(+ 'a 'b) => ERRO! Operação aplicável apenas
a números

(+ a (sqr b)) => 18
```

Exercícios (cont.)

```
(char=? #\a #\b)

(char=? #\aa #\a)

(symbol=? #\a 'a)

(symbol=? '#\a 'a)

(symbol=? '\a 'a)

(string=? "aa" "a a")

(string=? "#\a" "#\a")

(string=? "Hi" "hi")
```

Exercícios (cont.)

```
(char=? #\a #\b) => false

(char=? #\aa #\a)

(symbol=? #\a 'a)

(symbol=? '#\a 'a)

(symbol=? '\a 'a)

(string=? "aa" "a a")

(string=? "#\a" "#\a")

(string=? "Hi" "hi")
```

Exercícios (cont.)

```
(char=? #\a #\b) => false

(char=? #\aa #\a) => ERRO! Caracter incorreto: #\aa

(symbol=? #\a 'a)

(symbol=? '#\a 'a)

(symbol=? '\a 'a)

(string=? "aa" "a a")

(string=? "#\a" "#\a")

(string=? "Hi" "hi")
```

Exercícios (cont.)

```
(char=? #\a #\b) => false

(char=? #\aa #\a) => ERRO! Caracter incorreto: #\aa

(symbol=? #\a 'a) => ERRO! Somente para símbolos

(symbol=? '#\a 'a)

(symbol=? '\a 'a)

(string=? "aa" "a a")

(string=? "#\a" "#\a")

(string=? "Hi" "hi")
```

Exercícios (cont.)

```
(char=? #\a #\b) => false

(char=? #\aa #\a) => ERRO! Caracter incorreto: #\aa

(symbol=? #\a 'a) => ERRO! Somente para símbolos

(symbol=? '#\a 'a) => ERRO! Símbolo incorreto: '#\a

(symbol=? '\a 'a)

(string=? "aa" "a a")

(string=? "#\a" "#\a")

(string=? "Hi" "hi")
```

Exercícios (cont.)

```
(char=? #\a #\b) => false

(char=? #\aa #\a) => ERRO! Caracter incorreto: #\aa

(symbol=? #\a 'a) => ERRO! Somente para símbolos

(symbol=? '#\a 'a) => ERRO! Símbolo incorreto: '#\a

(symbol=? '\a 'a) => true

(string=? "aa" "a a")

(string=? "#\a" "#\a")

(string=? "Hi" "hi")
```

Exercícios (cont.)

```
(char=? #\a #\b) => false

(char=? #\aa #\a) => ERRO! Caracter incorreto: #\aa

(symbol=? #\a 'a) => ERRO! Somente para símbolos

(symbol=? '#\a 'a) => ERRO! Símbolo incorreto: '#\a

(symbol=? '\a 'a) => true

(string=? "aa" "a a") => false

(string=? "#\a" "#\a")

(string=? "Hi" "hi")
```

Exercícios (cont.)

```
(char=? #\a #\b) => false

(char=? #\aa #\a) => ERRO! Caracter incorreto: #\aa

(symbol=? #\a 'a) => ERRO! Somente para símbolos

(symbol=? '#\a 'a) => ERRO! Símbolo incorreto: '#\a

(symbol=? '\a 'a) => true

(string=? "aa" "a a") => false

(string=? "#\a" "#\a") => true

(string=? "Hi" "hi")
```

Exercícios (cont.)

```
(char=? #\a #\b) => false

(char=? #\aa #\a) => ERRO! Caracter incorreto: #\aa

(symbol=? #\a 'a) => ERRO! Somente para símbolos

(symbol=? '#\a 'a) => ERRO! Símbolo incorreto: '#\a

(symbol=? '\a 'a) => true

(string=? "aa" "a a") => false

(string=? "#\a" "#\a") => true

(string=? "Hi" "hi") => false
```

Exercícios (cont.)

```
(image=? Castelo "Castelo")

(image=? Castelo Palácio)

(+ Castelo a)

(or (symbol=? 'a 'b) (= a 2))
```

Exercícios (cont.)

```
(image=? Castelo "Castelo") => ERRO! Operação
para imagens

(image=? Castelo Palácio)

(+ Castelo a)

(or (symbol=? 'a 'b) (= a 2))
```

Exercícios (cont.)

```
(image=? Castelo "Castelo") => ERRO! Operação
para imagens

(image=? Castelo Palácio) => true

(+ Castelo a)

(or (symbol=? 'a 'b) (= a 2))
```

Exercícios (cont.)

```
(image=? Castelo "Castelo") => ERRO! Operação
para imagens

(image=? Castelo Palácio) => true

(+ Castelo a) => ERRO! Operação para números

(or (symbol=? 'a 'b) (= a 2))
```

Exercícios (cont.)

```
(image=? Castelo "Castelo") => ERRO! Operação
para imagens

(image=? Castelo Palácio) => true

(+ Castelo a) => ERRO! Operação válida apenas
para números

(or (symbol=? 'a 'b) (= a 2)) => true
```

Exercícios - Parte 2

1. Uma loja dá descontos nas suas mercadorias dependendo do tempo em que elas estão no estoque. Se um item fica no estoque por até 2 semanas (exclusive), vale o preço original. A partir de 2 semanas em estoque (inclusive), o seu preço cai 25% (em relação ao preço original). Após iniciada a terceira semana, o preço cai 50%, e após iniciada a quarta semana, em 75%. A partir da quinta semana, não são dados descontos adicionais. Faça uma função que, dados o preço original de uma mercadoria e o número de semanas em que ela está no estoque, calcula o seu novo valor de venda.

Exercícios - Parte 2

2. Desenvolva a função `testa-chute` que consome dois números, chamados `chute` e `valor` e, dependendo da relação entre eles, devolve uma das seguintes opções: `'MuitoBaixo`, `'Exato!`, `'MuitoAlto`.

Exercícios - Parte 2

3. Desenvolva a função `testa-cores` que implementa um jogo de adivinhação de cores. Esta função recebe **duas cores** atribuídas pelo jogador 1 a dois objetos `p1` e `p2` (nesta ordem), bem como **duas outras cores que são os chutes do jogador 2** `c1` e `c2`. As respostas possíveis são: `'Exato!`, se as cores chutadas corresponderem às cores atribuídas aos dois objetos na ordem correta; `'UmChuteCorreto`, se apenas um chute corresponder à cor e objeto corretos; `'UmaCorCorreta`, se pelo menos um dos chutes corresponder a uma das cores atribuídas a um dos objetos (mas na ordem incorreta); e `'TudoErrado!`, caso nenhuma das outras respostas se aplique.

Exercícios - 1

```
;; valor-de-venda : Número Número -> Número
;; Calcula o valor de venda de um produto
;; dados o preço original do produto e o número
;; de semanas em que está no estoque

;; Exemplos
(valor-de-venda 100 1) ;; produz 100
(valor-de-venda 100 2) ;; produz 75
(valor-de-venda 100 3) ;; produz 50
(valor-de-venda 100 4) ;; produz 25
(valor-de-venda 100 5) ;; produz 25

(define (valor-de-venda preço-produto nro-semanas)
(...))
```

Exercícios - 1

```
;; valor-de-venda : Número Número -> Número
;; Calcula o valor de venda de um produto
;; dados o preço original do produto e o número
;; de semanas em que este está no estoque
```

```
(define (valor-de-venda preço nro-semanas)
(cond
 ;; itens a menos de 2 semanas no estoque: não há redução
[...]
 ;; itens entre 2 e 3 semanas em estoque, o preço cai 25%
[...]
 ;; itens entre 3 e 4 semanas em estoque, o preço cai 50%
[...]
 ;; na quinta semana o desconto é 75% e o preço se estabiliza
[...] ))
```

Exercícios - 1

```
;; valor-de-venda : Número Número -> Número
;; Calcula o valor de venda de um produto
;; dados o preço original do produto e o número
;; de semanas em que este está no estoque
;; Exemplos
(valor-de-venda 100 1) ;; produz 100
(valor-de-venda 100 2) ;; produz 75
(valor-de-venda 100 3) ;; produz 50
(valor-de-venda 100 4) ;; produz 25
(valor-de-venda 100 5) ;; produz 25

(define (valor-de-venda preço nro-semanas)
(cond
[(< nro-semanas 2) preço]
[(< nro-semanas 3) (- preço (* 0.25 preço))]
[(< nro-semanas 4) (- preço (* 0.50 preço))]
[else (- preço (* 0.75 preço))] ))
```

Exercícios - 2

```
;; testa-chute : Número Número -> Símbolo
;; Dado um  número denominado 'chute' e
;; um número 'valor', verifica a relação entre
;; eles e retorna o símbolo correspondente
;;'MuitoBaixo, 'Exato ou 'MuitoAlto

;; Exemplos
(testa-chute 2 5) ;; produz 'MuitoBaixo
(testa-chute 7 5) ;; produz 'MuitoAlto
(testa-chute 5 5) ;; produz 'Exato!

(define (testa-chute chute valor)
(...))
```

Exercícios - 2

```
;; testa-chute : Número Número -> Símbolo
;; Dado um  número denominado 'chute' e
;; um número 'valor', verifica a relação entre
;; eles e retorna o símbolo correspondente
;;'MuitoBaixo, 'Exato ou 'MuitoAlto

;; Exemplos
(testa-chute 2 5) ;; produz 'MuitoBaixo
(testa-chute 7 5) ;; produz 'MuitoAlto
(testa-chute 5 5) ;; produz 'Exato!
```

```
(define (testa-chute chute valor)
(cond
[(< chute valor) 'MuitoBaixo]
[(> chute valor) 'MuitoAlto]
[(= chute valor) 'Exato!]))
```

Exercícios - 3

```
;; testa-cores : Símbolo Símbolo Símbolo Símbolo -> Símbolo
;; Verifica se as cores atribuídas aos objetos 'p1' e 'p2' (2 símbolos)
;; correspondem aos chutes 'c1' e 'c2' (2 símbolos), na mesma ordem

;; Exemplos
(testa-cores 'vermelho 'azul 'vermelho 'azul) ;; produz 'Exato!
(testa-cores 'vermelho 'azul 'vermelho 'amarelo) ;; produz
'UmChuteCorreto
(testa-cores 'vermelho 'azul 'rosa 'azul) ;; produz 'UmChuteCorreto
(testa-cores 'vermelho 'azul 'azul 'preto) ;; produz 'UmaCorCorreta
(testa-cores 'vermelho 'azul 'marrom 'vermelho) ;; produz
'UmaCorCorreta
(testa-cores 'vermelho 'azul 'azul 'vermelho) ;; produz 'UmaCorCorreta
(testa-cores 'vermelho 'azul 'preto 'amarelo) ;; produz 'TudoErrado!
```

Exercícios - 3 (cont.)

```
;; testa-cores : Símbolo Símbolo Símbolo Símbolo -> Símbolo
;; Verifica se as cores atribuídas aos objetos 'p1' e 'p2' (2 símbolos)
;; correspondem aos chutes 'c1' e 'c2' (2 símbolos), na mesma ordem

;; exemplos: ver slide anterior

(define (testa-cores p1 p2 c1 c2)
(cond
 ;; chute 1 e chute 2 na ordem correta
[...]
 ;; chute 1 OU chute 2 na ordem correta
[...]
 ;; chute 1 OU chute 2 corretos mas ordem errada
[...]
 ;; nenhum chute correto
[...]))
```

Exercícios - 3 (cont.)

```
;; testa-cores : Símbolo Símbolo Símbolo Símbolo -> Símbolo
;; Verifica se as cores atribuídas aos objetos 'p1' e 'p2' (2 símbolos)
;; correspondem aos chutes 'c1' e 'c2' (2 símbolos), na mesma ordem

;; exemplos: ver slide anterior

(define (testa-cores p1 p2 c1 c2)
(cond
 ;; chute 1 e chute 2 na ordem correta
[(and (symbol=? p1 c1) (symbol=? p2 c2)) 'Exato!]
 ;; chute 1 OU chute 2 na ordem correta
[(or (symbol=? p1 c1) (symbol=? p2 c2)) 'UmaPosiçãoCorreta]
 ;; chute 1 OU chute 2 corretos mas ordem errada
[(or (symbol=? p1 c2) (symbol=? p2 c1)) 'UmaCorCorreta]
 ;; nenhum chute correto
[else 'TudoErrado!]))
```

Lembrete

- Ler Cap. 5 e 6
- Instale e use o Racket (próxima aula é laboratório 1 e conhecimento de Racket acelera seu andamento)
- Resolver exercícios da lista (sobre informação simbólica, cap. 5)