# Expressões Aritméticas

Fundamentos de Algoritmos

INF05008

# Números e Aritmética

- Espécies mais populares
  - Inteiros: `-5, 0, 1, 5`
  - Frações (ou racionais): `1/2, 0.5, 0.`$\overline{3}$
  - Reais inexatos: `#i1.4142135623731`

- Tamanho dos números **não é fixo**

```
123457643215678745243424234234234
```

- Toda operação em **Scheme** possui o seguinte formato:

```
(operador a b c ...)
```

  - **Operador** sempre à frente
  - **Operandos** separados por espaços

- Operações aritméticas básicas
  - Somas: `(+ -5 5)`
  - Subtrações: `(- 5 5)`
  - Multiplicações: `(* 3 4)`
  - Divisões: `(/ 8 12)`

- A **avaliação** é tal como nós fazemos, **reduzindo termos**

```
   (* (+ 2 2) (/ (* (+ 3 5) (/ 30 10)) 2))
= (* 4 (/ (* 8 3) 2))
= (* 4 (/ 24 2))
= (* 4 12)
= 48
```

- Avaliação dos argumentos **da esquerda para a direita**
- Muitas operações binárias são também **n-árias** em Scheme

```
(+ 1 2 3 4)
```

- Por causa da forma, **nunca há dúvidas** sobre ordem da avaliação
- Compare com $3 + 4 * 5$
- Com o tempo e a prática, aprendemos que a multiplicação é feita primeiro e, depois, a soma

Outras operações aritméticas:

- `(sqrt A)` computa $\sqrt{A}$
- `(expt A B)` computa $A^B$
- `(quotient A B)` computa o quociente da divisão inteira $A/B$
- `(remainder A B)` computa o resto da divisão inteira $A/B$
- `(log A)` computa logaritmo natural de A
- `(sin A)` computa o seno de $A$ (em radianos)

- Scheme computa com inteiros e racionais **exatos** se usarmos operações primitivas que produzem resultados exatos
- Assim, o resultado de `(/ 44 14)` é exibido como `22/7`
- Scheme (e outras linguagens) produz valores **inexatos** quando lida com números reais
- Raiz quadrada de 2 não é racional e, sim, um **real**

A notação `#i` avisa ao programador de que o resultado é uma **aproximação** do valor verdadeiro

```
   (- #i1.0 #i0.9)
 = #i0.09999999999999998
```

mas

```
   (- #i1000.0 #i999.9)
 = #i0.10000000000002274
```

mesmo sabendo, da Matemática, que ambas diferenças deveriam ser $0,1$

**Na disciplina: números exatos**

# Variáveis e Programas

- Pela Matemática, aprendemos a formular dependências entre quantidades usando **variáveis** para representar quantidades desconhecidas
- Exemplo: um disco de raio $r$ tem área aproximada de $3,14 * r^2$
- Para um disco de raio $5$, fazemos:

$$3,14 * 5^2 = 3,14 * 25 = 78,5$$

- Uma expressão com variáveis é uma **regra** que descreve como computar um número quando temos valores para as variáveis
- Um **programa** é uma regra que diz como **produzir um dado a partir de um outro dado**
- Deve haver uma maneira de **nomear regras**
- Regra para computar área de um disco

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

- Depois de definida, uma regra pode ser usada **fornecendo um valor para cada variável** que segue o nome da regra

```
(área-do-disco 5)
```

- Dizemos que **aplicamos** `área-do-disco` para 5.

```
  (área-do-disco 5)
= (* 3.14 (* 5 5))
= (* 3.14 25)
= 78.5
```

- Muitos programas consomem **mais do que uma entrada**
- Exemplo: Programa para computar a **área de um anel**, ou seja, de um disco com um buraco no meio
- Área do anel é a área do disco externo menos a área do disco interno
- Há, portanto, **dois valores desconhecidos**: o raio do disco interno e o raio do disco externo

```
(define (área-do-anel externo interno)
 (- (área-do-disco externo)
    (área-do-disco interno)))
```

- `área-do-anel` é um programa que aceita duas entradas, chamadas `externo` e `interno`
- O resultado será a diferença entre *(área-do-disco externo)* e *(área-do-disco interno)*
- Usamos **operações básicas** de Scheme e também **programas definidos**

Para usar *área-do-anel*, fornecemos duas entradas

```
(área-do-anel 5 3)
```

Essa expressão é avaliada da seguinte forma:

```
  (área-do-anel 5 3)

= (- (área-do-disco 5)
     (área-do-disco 3))

= (- (* 3.14 (* 5 5))
     (* 3.14 (* 3 3)))
= ...
```

# Enunciado de Problemas

- Raramente o problema é representar expressões aritméticas em programas
- Em geral, descrições de problemas são textos **informais**, com **ambiguidades** e **detalhes sem importância**
- **Primeira tarefa**: obter a **informação relevante** e, só então, formular a **expressão apropriada**

Eis um exemplo:

*"A empresa XYX & Cia. paga aos seus empregados $R\$12$ por hora. Um empregado típico trabalha de 20 a 65 horas por semana. Desenvolva um programa que calcule o salário de um empregado a partir do seu número de horas trabalhadas."*

Eis um exemplo:

*"A empresa XYX & Cia. paga aos seus empregados $R\$12$* ***por hora****. Um empregado típico trabalha de 20 a 65 horas por semana. Desenvolva um programa que* ***calcule o salário*** *de um empregado a partir do seu* ***número de horas trabalhadas****."*

Eis um exemplo:

*"A empresa XYX & Cia. paga aos seus empregados $R\$12$* ***por hora****. Um empregado típico trabalha de 20 a 65 horas por semana. Desenvolva um programa que* ***calcule o salário*** *de um empregado a partir do seu* ***número de horas trabalhadas****."*

Se um empregado trabalha $h$ horas, o seu salário será $12 * h$. Logo, o programa será:

```
(define (salário h)
   (* 12 h))
```

# Erros de Sintaxe

- **Expressões** ou são **atômicas** (números e variáveis)
- Ou são expressões **compostas**, que começam com “(”, seguido de operação, mais expressões e terminam por “)”
- **Definições** têm a seguinte forma:

```
(define (f x ... y)
  <expressão>)
```

# Erros de Sintaxe (cont.)

As seguintes definições possuem erros de **sintaxe**:

```
(define (P x)
  (+ (x) 10))

(define (Q x)
  x 10)
```

# Erros de Execução

- Nem toda expressão **legal** possui um **resultado**
- Exemplo óbvio é `(/ 1 0)`
- Da mesma forma, se definirmos

```
(define (f n)
   (+ (/ n 3) 2))
```

  DrScheme não poderá avaliar `(f 5 8)`

- **RUN-TIME ERROR**

# Erros Lógicos

- Programador pode também cometer erros **lógicos**
- **Não são detectados** por DrScheme e nenhuma mensagem é produzida
- Tais erros somente podem ser evitados através do projeto **cuidadoso e metódico** de programas
- Exemplo de programa com erro lógico:

```
(define (salário h)
 (+ 12 h))
```

# Compondo Funções

- Em geral, programa consiste de **várias definições**
- Programa para **cálculo da área de um anel** possui 2 definições: *área-do-anel* e *área-do-disco*
- *área-do-anel* é a **função principal**
- *área-do-disco* é uma **função auxiliar**

# Compondo Funções (cont.)

- Eis duas versões para o cálculo da área de um anel
- Qual das duas é mais legível?

```
(define (área-do-anel externo interno)
  (- (área-do-disco externo)
     (área-do-disco interno)))
```

ou

```
 (define (área-do-anel externo interno)
  (- (* 3.14 (* externo externo))
     (* 3.14 (* interno interno))))
```

# Compondo Funções (cont.)

- Para **programas pequenos**, a diferença entre os dois estilos é pequena
- Para **programas maiores**, o uso de funções auxiliares é uma **necessidade**
- Mas, mesmo para programas menores, devemos considerar **dividir o problema** com o uso de auxiliares

# Exercícios

1. Leia os capítulos 1, 2 e 3 do livro `www.htdp.org`

2. Instale o DrScheme em seu computador e/ou use o DrScheme já instalado nos laboratórios

3. Experimente o DrScheme

   - Experimente usar algumas funções predefinidas
   - Use a avaliação passo-à-passo (*stepper*)
   - Force erros de sintaxe e de execução e observe as mensagens de erro produzidas pelo DrScheme