BIS0005 - Bases Computacionais da Ciência

Aula 06 - Lógica de programação: Estruturas condicionais

Saul Leite
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

Q2 2018

Introdução

Estruturas de controle permitem o controle do fluxo de execução dos comandos. Existem três estruturas básicas de controle:

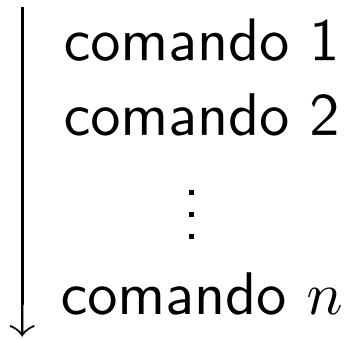
1. Sequencial

3. Condicional ou Desvio (if)

4. Repetição (próxima aula)

Estrutura Sequencial

É padrão em toda a forma de algoritmo. Uma estrutura sequencial é um conjunto de comandos que serão executados em uma sequência linear, de cima para baixo:



Os comandos são executados na **mesma ordem** em que foram escritos.

Estrutura Condicional

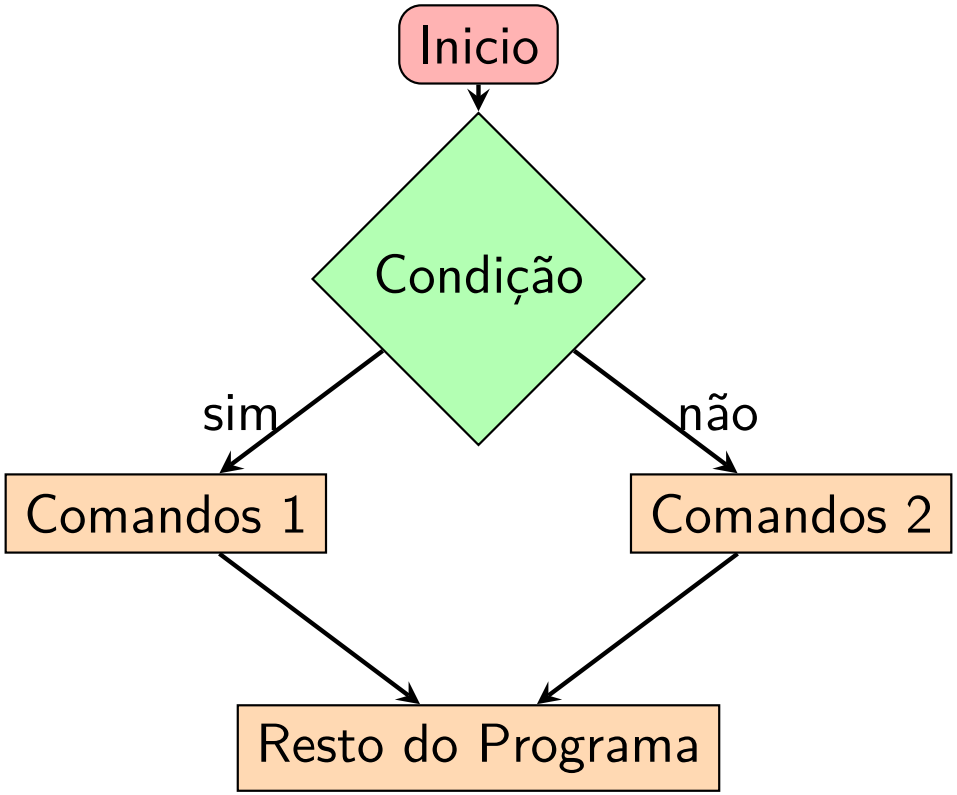
É também conhecida como estrutura de decisão ou seleção.

Um desvio condicional é usado para escolher entre cursos alternativos de ação em um programa. Exemplo:

```
se Condição então
    #Caso condição VERDADEIRA
    Comando a
    Comando b
    ...
senão
    #Caso condição FALSA
    Comando c
    Comando d
    ...
fim
Resto do programa
```

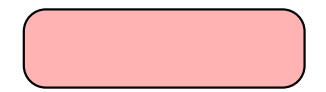
Estrutura Condicional: Fluxogramas

Representando estruturas condicionais com **Fluxogramas**:

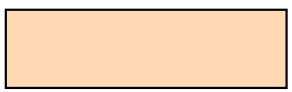


Obs.: O "sim" representa que a condição é verdadeira, e o "não" representa que é falsa.

Fluxogramas

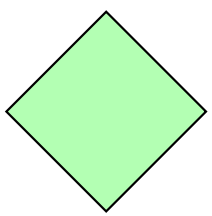


Representando o inicio ou fim.



Representa um processo (ou comando).

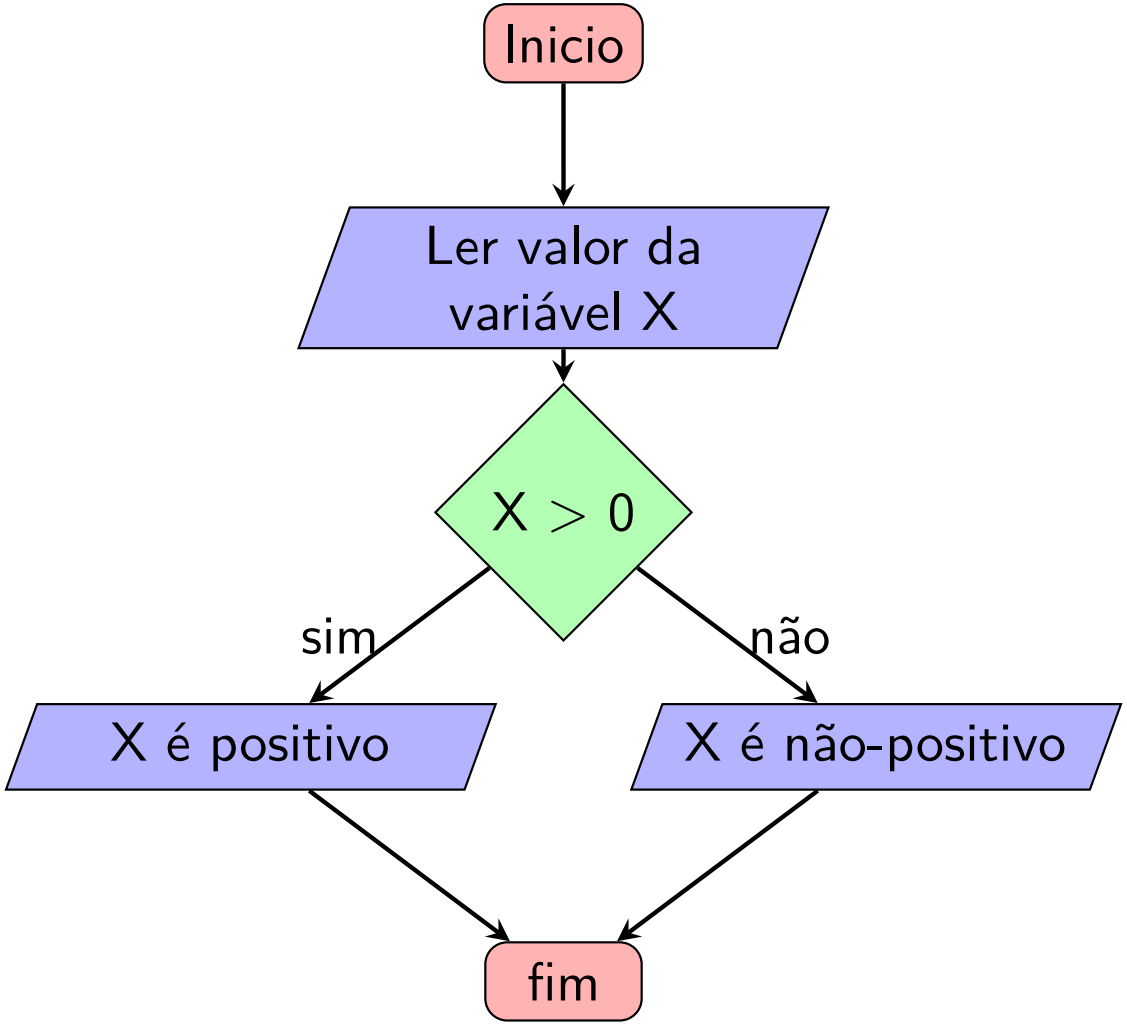
Representa entrada ou saída de dados.



Representa decisão.

Representa a direção do fluxo de processo.

Fluxogramas: Exemplo



Condição: Operadores relacionais

A **Condição** é um **predicado lógico** (expressões que devolvem um valor VERDADEIRO OU FALSO).

Geralmente são construídos com **expressões relacionais**:

(valor 1) **(Operador Relacional)** **(valor 2)**

Condição: Operadores relacionais

A **Condição** é um **predicado lógico** (expressões que devolvem um valor VERDADEIRO OU FALSO).

Geralmente são construídos com **expressões relacionais**:

(valor 1) **(Operador Relacional)** **(valor 2)**

Exemplos: (X e Y são variáveis)

X >= **0**

Condição: Operadores relacionais

A **Condição** é um **predicado lógico** (expressões que devolvem um valor VERDADEIRO OU FALSO).

Geralmente são construídos com **expressões relacionais**:

(valor 1) **(Operador Relacional)** **(valor 2)**

Exemplos: (X e Y são variáveis)

X	>=	**0**
Y	==	**"Olá"**

Condição: Operadores relacionais

Condições são construídas com **expressões relacionais**:

(valor 1) **(Operador Relacional)** **(valor 2)**

Os **Operadores relacionais** são dados por:

Operador	Descrição	Em R
$<$	menor	$<$
$\leq$	menor ou igual	$<=$
$>$	maior	$>$
$\geq$	maior ou igual	$>=$
$=$	igual	$==$
$\neq$	diferente	$!=$

Condição: Operadores relacionais

Suponha que X e Y são duas variáveis numéricas (*double*). Temos os seguintes exemplos de **expressões relacionais**:

```
X > 0
Y <= X + 2
X == Y
Z != "Olá"
```

Condição: Operadores relacionais

Suponha que X e Y são duas variáveis numéricas (*double*). Temos os seguintes exemplos de **expressões relacionais**:

```
X > 0
Y <= X + 2
X == Y
Z != "Olá"
```

Se o valor das variáveis são os seguintes,

```
X = 3
Y = 4
Z = "olá"
```

quais são os resultados das expressões acima?

Condição: Operadores relacionais

Suponha que X e Y são duas variáveis numéricas (*double*). Temos os seguintes exemplos de **expressões relacionais**:

```
X > 0              #----> TRUE
Y <= X + 2         #----> TRUE
X == Y             #----> FALSE
Z != "Olá"         #----> TRUE
```

Se o valor das variáveis são os seguintes,

```
X = 3
Y = 4
Z = "olá"
```

quais são os resultados das expressões acima?

Condição: Operadores Lógicos (ou Booleanos)

Operadores lógicos podem ser usados para gerar condições mais elaboradas. Os operadores abaixo são os mais frequentemente empregados:

Operador	Descrição	Em R
e	conjunção	&
ou	disjunção	|
não	negação	!

Condição: Operadores Lógicos (ou Booleanos)

Tabela Verdade:

a	b	a & b	a | b	!a
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

Condição: Operadores Lógicos (ou Booleanos)

Exemplos: (Entre esses valores no R)

```
X <- 3
Y <- 4


(X > Y) & (X > 0)
(X > Y) | (X > 0)
!(X > 0)
(X < Y) & (X > 0) & (Y > 0)
```

Condição: Operadores Lógicos (ou Booleanos)

Exemplos: (Entre esses valores no R)

```
X <- 3
Y <- 4

(X > Y) & (X > 0)                  #-----> FALSE
(X > Y) | (X > 0)                  #-----> TRUE
!(X > 0)                           #-----> FALSE
(X < Y) & (X > 0) & (Y > 0)        #-----> TRUE
```

Estrutura Condicional: Simples

O desvio condicional na linguagem R é feito com os comandos **if** e **else**. Os blocos de comandos são agrupados com **chaves {}**, como no exemplo abaixo:

```
if( X > 0 ){

  #comando executado se X for positivo
  cat("Numero é positivo\n")

} else {

  #comando executado se X for negativo ou zero
  cat("Numero é negativo ou zero\n")
}
```

Obs.: O else deve **sempre** estar na mesma linha do chave que fecha o comando **if**.

Estrutura Condicional: Simples

Qual é o resultado do programa abaixo?

```
x <- 13

if( (X %% 2 == 0) & (X > 0) ){

  cat("O Número ",x," é par e positivo\n")

}
```

Obs.: A utilização do **else** é opcional.

Estrutura Condicional: Simples

Exemplo: Faça um programa que calcula as raízes reais de um polinômio de segundo grau:

$$ax^2 + bx + c.$$

O programa deve pedir os valores de a, b, e c e determinar as raízes usando a fórmula de Bhaskara:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

O programa deve verificar se as raízes serão reais! Se não forem reais, o programa deve exibir uma mensagem de erro.

Estrutura Condicional: Simples

Exemplo:

```
A <- as.numeric(readline("Digite o coeficiente a: "))
B <- as.numeric(readline("Digite o coeficiente b: "))
C <- as.numeric(readline("Digite o coeficiente c: "))

delta <- B^2 - 4*A*C

if( delta >= 0){

  xp <- (-B + delta)/(2*A)
  xm <- (-B - delta)/(2*A)

  cat("As raízes são: ",xp," e ",xm,"\n")

} else {

  cat("As raízes não são reais\n")
}
```

Estrutura Condicional: Composta

Podemos compor várias estruturas condicionais para tratar casos em que temos diversas condições. Isso é feito da seguinte forma:

```
if( condição 1 )
{
  #Comandos se a condição 1 for verdadeira

} else if (condição 2) {

  #Comandos se a condição 2 for verdadeira

} else if (condição 3) {

  #Comandos se a condição 3 for verdadeira

} else {

  #Comandos se nenhuma condição for verdadeira
}
```

Estrutura Condicional: Composta

Exemplo: Crie um programa que solicita um valor em decibéis do usuário. Se o valor entrado for igual a um valor da tabela abaixo, você deve exibir o nome do ruído listado na tabela. Se o usuário digitar um número de decibéis entre os ruídos listados, seu programa deverá exibir uma mensagem indicando os ruídos que estão acima e abaixo do valor entrado.

Ruídos	Decibéis (dB)
Britadeira	130
Cortador de grama a gasolina	106
Rádio-relógio (despertador)	70
Sala silenciosa	40

Exercício do livro: "The Python Workbook" por Ben Stephenson.

Estrutura Condicional: Composta

```
db <- as.numeric(readline("Digite um valor em decibel: "))
if( db == 130 ){
  cat("Ruído igual a uma britadeira\n")
} else if ( 106 < db & db < 130 ){
  cat("Ruído entre uma britadeira e um cortador de grama\n")
} else if ( db == 106 ){
  cat("Ruído igual a um cortador de grama\n")
} else if ( 70 < db & db < 106 ){
  cat("Ruído entre um cortador de grama e rádio-relógio\n")
} else if ( db == 70 ){
  cat("Ruído igual a um rádio-relógio\n")
} else if ( 40 < db & db < 70 ){
  cat("Ruído entre um rádio-relógio e uma sala silenciosa\n")
} else if ( db == 40 ){
  cat("Ruído igual a uma sala silenciosa\n")
} else {
  cat("Valor em decibel fora do intervalo\n")
}
```

Funções revisitadas: argumentos e retorno

Já vimos que podemos criar **funções** no R usando o comando **function**, como no exemplo abaixo:

```
f <- function(x) x^2 + 2
```

Neste exemplo, criamos a função $f(x) = x^2 + 2$.

Funções revisitadas: argumentos e retorno

Já vimos que podemos criar **funções** no R usando o comando **function**, como no exemplo abaixo:

```
f <- function(x) x^2 + 2
```

Neste exemplo, criamos a função $f(x) = x^2 + 2$.

Usamos a seguinte nomenclatura:

1. **argumento ou parâmetro**: nome dado as variáveis passadas para a função (neste exemplo a variável x);
2. **valor de retorno**: o valor resultante da função.

Funções revisitadas: argumentos e retorno

Por exemplo:

```
f <- function(x) x^2 + 2
f(1)
```

```
## [1] 3
```

No exemplo acima, passamos 1 para o valor do **argumento** x, e o valor de **retorno** da função é igual a 3.

Funções revisitadas: vários argumentos

Poremos ter funções que dependem de vários **argumentos**. Por exemplo, a função abaixo depende de x e de y:

```
#uma função de duas variáveis
g <- function(x,y) x*y + 3
```

Funções revisitadas: vários argumentos

Poremos ter funções que dependem de vários **argumentos**. Por exemplo, a função abaixo depende de x e de y:

```
#uma função de duas variáveis
g <- function(x,y) x*y + 3
```

Qual será o valor de **retorno** se $x = 2$ e $y = 3$?

Funções revisitadas: vários argumentos

Poremos ter funções que dependem de vários **argumentos**. Por exemplo, a função abaixo depende de x e de y:

```
#uma função de duas variáveis
g <- function(x,y) x*y + 3
```

Qual será o valor de **retorno** se $x = 2$ e $y = 3$?

```
g(2,3)
```

```
## [1] 9
```

Funções revisitadas: vários comandos

Podemos criar funções compostas por várias linhas no R. Todos os comandos usados para fazer a função devem vir entre **chaves { }**.

Por exemplo, vamos criar uma função para calcular a área de um triângulo usando a fórmula de Heron:

```
area.triangulo <- function(a,b,c){

  p <- (a+b+c)/2
  area <- sqrt(p*(p-a)*(p-b)*(p-c))
  return(area)

}
```

Indicamos o valor de **retorno** da função usando o comando **return**.

Funções revisitadas: vários comandos

Função para calcular a área do triangulo com a fórmula de Heron:

```
area.triangulo <- function(a,b,c){

  p <- (a+b+c)/2
  area <- sqrt(p*(p-a)*(p-b)*(p-c))
  return(area)

}
```

Exemplo de execução:

```
area.triangulo(3,4,5)
```

```
## [1] 6
```

Funções revisitadas: argumentos

Note que no exemplo abaixo, criamos as variáveis **p** e **área**.

```
area.triangulo <- function(a,b,c){

  p <- (a+b+c)/2
  area <- sqrt(p*(p-a)*(p-b)*(p-c))
  return(area)

}
```

Estas variáveis somente existem **dentro** da função e **não fazem parte** do **ambiente global**. O mesmo acontece para as variáveis de argumento, **a**, **b**, e **c**.

ATIVIDADE EM SALA

Exercicio 1

Faça um programa na linguagem R que lê do teclado dois números x e y e atribui o menor desses valores na variável *menor* e o maior na variável *maior*. No final, exiba o resultado na tela.

```
Exemplo1:
valores entrados: 3, 4
saída: O maior é 4 e o menor é 3

Exemplo2:
valores entrados: 40, 5
saída: O maior é 40 e o menor é 5
```

Exercicio 2

Faça um programa que leia três lados a, b, e c. Verifique se estes valores formam um triângulo e exiba o resultado na tela.

obs.: Os lados a, b e c formam um triângulo se todos os lados são maiores do que zero e a soma de dois lados é sempre maior do que o valor do terceiro.

```
Exemplo1:
valores entrados: 3, 4, 5
saída: Forma um triângulo.

Exemplo2:
valores entrados: 3, 10, 40
saída: Não forma um triângulo.
```

Exercicio 3

Defina uma função que recebe por argumento três lados a, b e c e calcula a área do triângulo com a fórmula de Heron. Desta vez, verifique se os valores passados por argumento formam de fato um triângulo. A função deve imprimir uma mensagem de erro caso os valores passados por argumento não formam um triângulo.

Obs.: Crie um novo *script* que contenha somente a definição desta função. Chame sua função na linha de comando para testá-la.

Exercicio 4

Defina uma função que recebe por parâmetroos coeficientes a, b e c da equação abaixo:

$$ax^2 + bx + c = 0$$

e retorna o valor de suas raízes. A função deve imprimir uma mensagem de erro se as raízes da equação não são valores reais.

Obs.: Crie um novo *script* que contenha somente a definição desta função. Chame sua função na linha de comando para testá-la.

Exercicio 5

Crie um programa para determinar a quantidade de dias para um determinado mês do ano. (Assuma que o ano não é bissexto.)

Mês	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
Dias	31	28	31	30	31	30	31	31	30	31	30	31

O Programa deve solicitar um valor numérico para o mês e imprimir o número de dias, de acordo com a tabela acima. Se o valor entrado não corresponder a um mês, imprimir uma mensagem de erro.

Exercicio 6

Exemplo: Crie um programa que solicita um valor de peso (em kg) e altura (em metros) e determina o índice de massa corporal (IMC) e sua classificação. O cálculo do IMC é dado por:

$$IMC = \frac{\text{peso}}{\text{altura}^2},$$

com a seguinte classificação:

IMC	Classificação
$<$18.5	Peso Baixo
18.5 - 24.9	Peso Normal
25.0 - 29.9	Sobrepeso
30.0 - 34.9	Obesidade (grau I)
35.0 - 39.9	Obesidade (grau II)
$>=$ 40	Obesidade (grau III)

Exercicio 7

Faça um programa que leia três valores. Esses valores representarão o comprimento dos lados de um triângulo. Verifique se esses valores formam um triângulo e classifique-o em:

- equilátero: três lados iguais;
- isósceles: dois lados iguais;
- escaleno: três lados diferentes.

Exercicio 8

Escreva um programa que leia três valores, armazenando-os nas variáveis x, y e z, e ordene esses valores de modo que, ao final, o menor valor esteja armazenado na variável x, o valor intermediário esteja armazenado na variável y e o maior valor esteja armazenado na variável z.

MATERIAL EXTRA

Revisitando funções: Vetorização

Suponha que temos a seguinte função:

```
h <- function(x) x^3 -2*x+11
```

Sabemos que para fazer o gráfico, basta definir um vetor de pontos e chamar a função acima para estes valores:

```
x <- seq(-5,5,0.01)
plot(x,h(x),type="l")
```

Neste caso, passamos para a função $h(x)$ um **vetor** x por argumento. Como as operações (+,-,*,\,^) em vetores na linguagem R são feitas *elemento a elemento*, o resultado é um outro vetor contendo o valor da função para cada elemento do vetor.

Revisitando funções: Vetorização

Podemos visualizar isso com o seguinte exemplo:

```
h <- function(x) x^3 -2*x + 11
x <- c(-1,0,1)
h(x)
```

```
## [1] 12 11 10
```

Note, que $h(x)$ dá o resultado de $h(-1)$, $h(0)$, e $h(1)$ em um vetor.

Revisitando funções: Vetorização

Contudo, quando usamos o comando **if** e **else** para definirmos nossa função, como no exemplo abaixo:

```
w <- function(x){
  if(x <= 2){
    return(x^3 -2*x + 11)
  } else {
    return(3*x-2)
  }
}
```

Nós temos um **erro** ao fazer!!

```
x <- c(-1,0,1,3)
w(x)
```

Isso porque o comando **if** não funciona com **vetores**!

Revisitando funções: Vetorização

Podemos resolver este problema **vetorizando** a função. Isso é feito chamando a função *Vectorize* no R. Esta função recebe por argumento uma função e retorna uma outra função vetorizada.

Exemplo:

```
w <- function(x){
  if(x <= 2){
    return(x^3 -2*x + 11)
  } else {
    return(3*x-2)
  }
}

w.vec <- Vectorize(w)
```

Revisitando funções: Vetorização

```
w <- function(x){
  if(x <= 2){
    return(x^3 -2*x + 11)
  } else {
    return(3*x-2)
  }
}

w.vet <- Vectorize(w)
```

Agora podemos passar vetores para a função sem problemas!

```
x <- c(-1,0,1,3)
w.vet(x)
```

```
## [1] 12 11 10  7
```

Exercício 9

Considere a função abaixo:

$$g(t) = \begin{cases} 0 & t \leq -2 \\ -4 - 2t & -2 < t \leq 0 \\ -4 - 3t & 0 < t \leq 4 \\ 16 - 2t & 4 < t \leq 8 \\ 0 & t > 8 \end{cases}$$

Faça um *scrit* em R que define a função acima e gera seu gráfico no intervalo $[-3, 10]$.

Obs.: Você terá que **vetorizar** a função.

Referências

- Aulas dos Profs. David Correa Martins Jr, Jesús P. Mena-Chalco.
- Livro Bases Computacionais da Ciência.