**Aula prática 3**

# Comandos de Desvio 1

**Resumo**

Nesta aula você irá resolver problemas que requerem uma decisão com base em um teste, ou condição. Para implementar a solução de problemas desse tipo, são utlizados comandos de desvio do fluxo de execução do programa (**if-then-else**).

## Sumário



## 1 Comando de desvio

Suponha que você quer escrever um programa Scilab para calcular o valor de $f(x)$, onde $f$ é a função definida a seguir, e $x$ é um valor especificado pelo usuário.

$$f(x) = \ln \frac{1}{1-x}$$

Note que essa função não é definida se $x = 1$. Portanto, seu programa deverá testar se o valor especificado pelo usuário é igual a 1 e, em caso afirmativo, informar ao usuário que esse não é um valor válido. Caso contrário, o programa deve calcular e imprimir o valor de $f(x)$.

O programa poderia ser então escrito do seguinte modo:

```
x = input("Informe o valor de x (deve ser diferente de 1): ")
if x <> 1 then
    printf("f(%g) = %g\n", x, 1/log(1-x))
else
    printf("Valor inválido: deve ser diferente de 1\n")
end
```

Um comando **if-then-else** tem a seguinte sintaxe:

**if** *condição* **then**
    *bloco de comandos 1*
**else**
    *bloco de comandos 2*
**end**

A *condição* deve ser uma expressão *lógica* (*booleana*), isto é, uma expressão cujo valor é verdadeiro (`%t`) ou é falso (`%f`). Cada bloco de comandos é uma sequência de comandos, incluindo, possivelmente, outros comandos de desvio.

A execução de um comando **if-then-else** é feita do seguinte modo: primeiro, a condição é avaliada; se o valor resultante for `%t`, o bloco de comandos 1 é executado; caso contrário (se o valor for `%f`), o bloco de comandos 2 é executado.

*Observação*: a parte **else** do comando pode ser omitida, caso não se deseje executar nenhum comando particular no caso em que a condição seja falsa.

## 2 Operações relacionais

Expressões lógicas podem ser construídas usando-se *operadores relacionais* (tal como < ou <=), e podem ser combinadas por meio de *operadores lógicos*, tais como & e |. Os operadores relacionais e os operadores lógicos disponíveis em Scilab são mostrados nas tabelas a seguir.

| Operadores Relacionais | |
|---|---|
| **operação** | **descrição** |
| $x$ < $y$ | $x$ menor que $y$ |
| $x$ <= $y$ | $x$ menor ou igual a $y$ |
| $x$ > $y$ | $x$ maior que $y$ |
| $x$ >= $y$ | $x$ maior ou igual a $y$ |
| $x$ == $y$ | $x$ igual a $y$ |
| $x$ ~= $y$ | $x$ diferente de $y$ |
| $x$ <> $y$ | |

| Operadores Lógicos | | |
|---|---|---|
| **operação** | **descrição** | **semântica** |
| ~$b$ | negação (não) | verdadeira se e somente se $b$ é falsa |
| $b1$ & $b2$ | conjunção (e) | verdadeira se e somente se $b1$ e $b2$ são ambas verdadeiras |
| $b1$ \| $b2$ | disjunção (ou) | falsa se e somente se $b1$ e $b2$ são ambas falsas |

## 3 Validação de dados

Muitas vezes os dados informados pelo usuário em uma aplicação precisam ser validados antes de serem usados. Isto pode ser feito usando um comando condicional, como é ilustrado no exemplo a seguir.

Para verificar se um número é inteiro, converta-o para o tipo inteiro usando a função `int` e verifique se o resultado é igual ao número original.

Para verificar se um número é natural, verifique se ele é inteiro e não negativo.

Por exemplo, o programa seguinte obtem um número natural e faz a validação do mesmo.

```
num = input("digite um número natural: ");
if int(num) <> num | num < 0 then
    printf("número inválido\n");
else
    printf("número válido: %g\n", num);
end
```

Veja uma outra maneira de fazer a mesma validação:

```
num = input("digite um número natural: ");
if int(num) == num & num >= 0 then
    printf("número válido: %g\n", num);
else
    printf("número inválido\n");
end
```

**Exemplo de execução da aplicação**

```
digite um número natural: 45
número válido: 45
```

**Exemplo de execução da aplicação**

```
digite um número natural: 2.34
número inválido
```

**Exemplo de execução da aplicação**

```
digite um número natural: -67
número inválido
```

**Exemplo de execução da aplicação**

```
digite um número natural: -0.25
número inválido
```

## 4 Problemas

**Tarefa 1: Fechamento de notas**

Escreva um programa que leia quatro notas de um aluno, calcule e mostre a média aritmética das notas e a mensagem de *aprovado* ou *reprovado*, considerando que para aprovação é necessária a média mínima de sete.

Não é necessário fazer a validação dos dados de entrada.

**Exemplo de execução da aplicação**

```
RESULTADO DAS NOTAS
-------------------
nota 1: 8.1
nota 2: 9.8
nota 3: 6.9
nota 4: 7.5
a média aritmética das notas é 8.075
aprovado
```

**Exemplo de execução da aplicação**

```
RESULTADO DAS NOTAS
-------------------
nota 1: 6.75
nota 2: 5.2
nota 3: 7
nota 4: 4.5
a média aritmética das notas é 5.8625
reprovado
```

**Solução:**

```
clc; clear;
printf("RESULTADO DAS NOTAS\n");
printf("-------------------\n")

// entrada de dados
nota1 = input("nota 1: ");
nota2 = input("nota 2: ");
nota3 = input("nota 3: ");
nota4 = input("nota 4: ");

// média aritmética das notas
media = (nota1 + nota2 + nota3 + nota4)/4;
printf("a média aritmética das notas é %g\n", media);

// situação do aluno
if media >= 7 then
    printf("aprovado\n");
else
    printf("reprovado\n");
end
```

**Tarefa 2: Ternos pitagóricos**

Em Matemática, nomeadamente em Teoria dos Números, um terno pitagórico (ou trio pitagórico, ou ainda tripla pitagórica) é formado por três números naturais $a$, $b$ e $c$ tais que $a^2+b^2 = c^2$. O nome vem do teorema de Pitágoras, que afirma que se as medidas dos lados de um triângulo retângulo são números inteiros, então são um terno pitagórico.

Codifique um programa que leia três números naturais e verifique se representam um terno pitagórico.

Caso os valores digitados pelo usuário não sejam números naturais, o programa deve apresentar uma mensagem indicando que os valores são inválidos.

**Exemplo de execução da aplicação**

```
Verificação de ternos pitagóricos
-----------------------------------------
Digite o valor de a: 3
Digite o valor de b: 4
Digite o valor de c: -5.6
Valores inválidos!
```

**Exemplo de execução da aplicação**

```
Verificação de ternos pitagóricos
-----------------------------------------
Digite o valor de a: 3
Digite o valor de b: 4
Digite o valor de c: 5
3, 4 e 5 representam um terno pitagórico
```

**Exemplo de execução da aplicação**

```
Verificação de ternos pitagóricos
-----------------------------------------
Digite o valor de a: 2
Digite o valor de b: 2
Digite o valor de c: 10
Os valores não representam um terno pitagórico
```

**Solução:**

```
clc; clear;
printf("Verificação de ternos pitagóricos\n");
printf("-----------------------------------------\n");

// entrada de dados
a = input("Digite o valor de a: ");
b = input("Digite o valor de b: ");
c = input("Digite o valor de c: ");

// validação dos dados de entrada
if int(a) <> a | int(b) <> b | int(c) <> c | a < 0 | b < 0 | c < 0 then
    printf("Valores inválidos!\n");
else
    // verificação de terno pitagórico
    if a^2 + b^2 == c^2 then
        printf("%g, %g e %g representam um terno pitagórico\n", a, b, c);
    else
        printf("Os valores não representam um terno pitagórico\n");
    end
end
```

## Tarefa 3: Peso ideal

Segundo uma tabela médica, o peso ideal de uma pessoa está relacionado com a altura e o sexo da pessoa, como mostra a tabela a seguir.

Fazer um programa que receba como entradas a altura e o sexo; a seguir ele calcula e imprime o peso ideal dessa pessoa, utilizando as seguintes fórmulas:

| sexo | peso ideal |
|---|---|
| masculino | $72.7 \times h - 58$ |
| feminino | $62.1 \times h - 44.7$ |

onde $h$ é a altura da pessoa.

O programa deve verificar se os dados são válidos, ou seja, se a altura não é negativa e o sexo é *m* ou *f*.

Veja a seguir ilustrações de entradas e saídas de execuções do programa.

**Exemplo de execução da aplicação**

```
Cálculo do peso ideal
-----------------------------------------
Qual é a altura (em metros)? -1.4
Qual é o sexo (m/f)? m
Dados inválidos!
```

**Exemplo de execução da aplicação**

```
Cálculo do peso ideal
----------------------------------------
Qual é a altura (em metros)? 1.72
Qual é o sexo (m/f)? h
Dados inválidos!
```

**Exemplo de execução da aplicação**

```
Cálculo do peso ideal
----------------------------------------
Qual é a altura (em metros)? 1.65
Qual é o sexo (m/f)? f
O peso ideal é 57.765 kg
```

**Exemplo de execução da aplicação**

```
Cálculo do peso ideal
----------------------------------------
Qual é a altura (em metros)? 1.8
Qual é o sexo (m/f)? m
O peso ideal é 72.86 kg
```

**Solução:**

```
clc; clear;
printf("Cálculo do peso ideal\n");
printf("----------------------------------------\n");

// entrada de dados
altura = input("Qual é a altura (em metros)? ");
sexo = input("Qual é o sexo (m/f)? ", "string");

// validação dos dados de entrada
if altura < 0 | ~(sexo == "m" | sexo == "f") then
    printf("Dados inválidos!\n");
else
    // cálculo do peso ideal
    if sexo == "m" then
        peso_ideal = 72.7 * altura - 58;
    else
        peso_ideal = 62.1 * altura - 44.7;
    end
    // exibição do resultado
    printf("O peso ideal é %g kg\n", peso_ideal);
end
```

**Tarefa 4: Quantidade de ladrilhos**

Um pedreiro precisa calcular quantos ladrilhos de cerâmica ele deve comprar para cobrir a área de uma sala. Faça um programa que leia a área da sala (em $cm^2$), e o tipo de ladrilho a ser adquirido, e calcule e imprima o número de ladrilhos necessários. As áreas de cada um dos tipos de ladrilhos disponíveis são dadas na tabela abaixo:

| tipo do ladrilho | área de uma peça ($cm^2$) |
|---|---|
| 1 | 80 |
| 2 | 60 |
| 3 | 40 |

**Exemplo de execução da aplicação**

```
Cálculo da quantidade de peças de ladrilho
-----------------------------------------
Área da sala (em cm^2): 820
Tipo do ladrilho (1/2/3): 1

Quantidade de ladrilhos necessários: 11
```

**Exemplo de execução da aplicação**

```
Cálculo da quantidade de peças de ladrilho
-----------------------------------------
Área da sala (em cm^2): 820
Tipo do ladrilho (1/2/3): 2

Quantidade de ladrilhos necessários: 14
```

**Exemplo de execução da aplicação**

```
Cálculo da quantidade de peças de ladrilho
-----------------------------------------
Área da sala (em cm^2): 820
Tipo do ladrilho (1/2/3): 3

Quantidade de ladrilhos necessários: 21
```

**Exemplo de execução da aplicação**

```
Cálculo da quantidade de peças de ladrilho
-----------------------------------------
Área da sala (em cm^2): -500
Tipo do ladrilho (1/2/3): 8

Dados inválidos!
```

**Dica** Os dados serão inválidos quando:

- o área for negativa, OU
- NÃO for verdade que
  - o tipo é 1 OU
  - o tipo é 2 OU
  - o tipo é 3

Codifique esta condição usando os operadores lógicos `|` e `~`.

**Solução:**

```
clc; clear;
printf("Cálculo da quantidade de peças de ladrilho\n");
printf("-----------------------------------------\n");

// entrada de dados
area = input("Área da sala (em cm^2): ");
tipo = input("Tipo do ladrilho (1/2/3): ");

// validação dos dados
if area < 0 | ~ (tipo == 1 | tipo == 2 | tipo == 3) then
    printf("\nDados inválidos!\n");
else
    // cálculo
    if tipo == 1 then
        quantidade = ceil(area/80);
    else
        if tipo == 2 then
            quantidade = ceil(area/60);
        else
            quantidade = ceil(area/40);
        end
    end
    // exibição da quantidade de ladrilhos
    printf("\nQuantidade de ladrilhos necessários: %g\n", quantidade);
end
```

**Tarefa 5: Locação de veículos**

Uma empresa de locação de veículos utiliza os seguintes valores para locação de um veículo:

- R$ 1,20 para os primeiros 100 km rodados,
- R$ 0,80 para os próximos 200 km rodados, e
- R$ 0,70 para a quilometragem acima de 300 km.

Escreva um programa Scilab que tenha como entrada a quilometragem percorrida por um veículo e que calcule e exiba o custo total da locação e o custo médio por quilômetro percorrido por esse veículo.

Veja a seguir exemplos de execução do programa.

**Exemplo de execução da aplicação**

```
CUSTO DA LOCAÇÃO DE UM VEÍCULO
------------------------------------------
distância percorrida (km): -348
dados inválidos!
```

**Exemplo de execução da aplicação**

```
CUSTO DA LOCAÇÃO DE UM VEÍCULO
------------------------------------------
distância percorrida (km): 80
custo total da locação: R$96
custo médio: R$1.2/km
```

**Exemplo de execução da aplicação**

```
CUSTO DA LOCAÇÃO DE UM VEÍCULO
----------------------------------------
distância percorrida (km): 150
custo total da locação: R$160
custo médio: R$1.06667/km
```

**Exemplo de execução da aplicação**

```
CUSTO DA LOCAÇÃO DE UM VEÍCULO
----------------------------------------
distância percorrida (km): 220
custo total da locação: R$216
custo médio: R$0.981818/km
```

**Exemplo de execução da aplicação**

```
CUSTO DA LOCAÇÃO DE UM VEÍCULO
----------------------------------------
distância percorrida (km): 431.6
custo total da locação: R$372.12
custo médio: R$0.862187/km
```

**Solução:**

```
clc; clear;
printf("CUSTO DA LOCAÇÃO DE UM VEÍCULO\n");
printf("----------------------------------------\n");

// entrada de dados
distancia = input("distância percorrida (km): ");

// validação dos dados
if distancia <= 0 then
    printf("dados inválidos!\n");
else
    // cálculos
    if distancia <= 100 then
        custo = 1.20*distancia;
    else
        if distancia <= 300 then
            custo = 1.20*100 + 0.80*(distancia - 100);
        else
            custo = 1.20*100 + 0.80*200 + 0.70*(distancia - 300);
        end
    end
    custoMedio = custo/distancia;
    // exibição dos resultados
    printf("custo total da locação: R$%g\n", custo);
    printf("custo médio: R$%g/km\n", custoMedio);
end
```