Operadores Relacionais e Lógicos, Comandos For e While e String

Operadores Relacionais

>	maior que
>=	maior ou igual a
<	menor que
<=	menor ou igual a
==	igual a
<> ou ~=	diferente de

Operadores e valores lógicos

Operador	Notação Scilab
NOT	~
AND	&
OR	|

Valores lógicos:

- Verdadeiro: constante lógica %t
- Falso: constante lógica %f

Exemplos de operações lógicas

```
-->a = %t; b = %f;
-->~a
 ans  =
  F
-->a & b
 ans  =
  F
-->a | b
 ans  =
  T
-->x = 10; y = 15;
-->a = x > y
 a  =
  F
```

O ";" permite colocar dois comandos em uma mesma linha

Comando de repetição **while**

```
while <condição>
   <bloco de repetição>
end
```

Programa: equação de segundo grau – 4ª versão

```
// Cálculo das raízes de uma equação
// de 2o grau

// Entrada e validação do coeficiente a,
// forçando-o a ter um valor válido
a = input ("Entre com o valor de a: ");
while (a == 0)
printf ("O coeficiente a deve ser diferente
de 0.\n");
     a = input ("Entre com o valor de a:
");
end

// Entrada dos coeficientes b e c
b = input ("Entre com o valor de b: ");
c = input ("Entre com o valor de c: ");
// resto do programa entra aqui
```

Comando de repetição **while**

Quando este *loop* vai parar?

```
x = 5
while (x < 10)
  printf("\nx = %g", x);
  x = x - 1;
end
```

Cuidado com os *loops* infinitos!

Comando de repetição **while**

- Exercício: O que faz o programa a seguir?

```
x = input("Forneça um número natural: ");
y = input("Forneça um número natural: ");
while (x < 0 | y < 0 )
  x = input("Forneça um número natural: ");
  y = input("Forneça um número natural: ");
end
while (y > 0)
   x = x + 1;
   x = y - 1;
end
printf("\n %g", x);
```

Comando de repetição **for**

```
for <variável> = <inicial>:<final>
  <bloco de repetição>
end
```

```
for <variável> = <inicial>:<passo>:<final>
  <bloco de repetição>
end
```

Comando **for** com passo 1

```
for i = 1:5
  printf("\ni = %g", i);
end
```

A variável de controle "i" é incrementada de 1 a cada interação

Saída

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

Comando **for** com passo diferente de 1

```
for i = 1:2:10
  printf('\ni = %g',i);
end
```

i varia de 2 em 2

```
i = 1
i = 3
i = 5
i = 7
i = 9
```

Saída

Repare que i não assumiu o limite superior do loop

Comando **for** com passo negativo

```
for i = 20:-2:16
  printf('\ni = %g',i);
end
```

```
i = 20
i = 18
i = 16
```

Saída

Comando **for** com controle fracionário

A variável de controle pode assumir valores não inteiros

```
for x = 0:0.3:0.7
  printf('\nx = %g',x);
end
```

```
x = 0
x = 0.3
x = 0.6
```

Saída

Equivalência – comandos **while** e **for**

```
for x = 0:2:10
   <bloco de comandos>
end
```

```
x = 0;
while (x <= 10)
   <bloco de comandos>
   x = x + 2;
end
```

Programa: fatorial de n

```
// Leitura e validação de n
n = input("Entre com o valor de n = ");
while (n < 0)
printf (" O valor de n deve ser maior ou igual
a 0!");
      n = input("Entre com o valor de n = ");
end

// Cálculo do fatorial de n
fat = 1;
if (n > 1) then
      for i = 2:n
            fat = fat * i;
      end
end

// Impressão do resultado
printf("O fatorial de %g é %g", n, fat);
```

Programa: Tabela de senos

x	seno(x)
0.0	0.0000
0.2	0.1987
0.4	0.3894
0.6	0.5646
0.8	0.7174

Parada: x = 2Π

Programa: Tabela de senos – 1ª versão

```
// Tabela da função Seno
for x = 0:0.2:2*%pi
  printf("%g %g", x, sin(x));
end
```

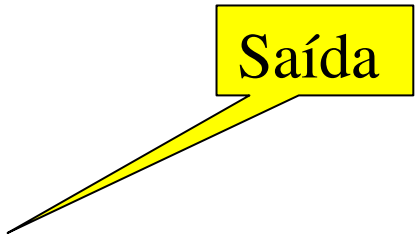
Saída

```
-->
 0 00.2 0.1986690.4 0.3894180.6 0.5646420.8 0.7173561 ...
```

Programa: Tabela de senos – 2ª versão

```
// Tabela da função Seno
for x = 0:0.2:2*%pi
  printf("\n %g %g", x, sin(x));
end
```

```
0 0
0.2 0.198669
0.4 0.389418
0.6 0.564642
0.8 0.717356
1 0.841471
1.2 0.932039
```



Programa: Tabela de senos – 3ª versão

```
// Tabela da função Seno

// Impressão do cabeçalho
printf("\n x   seno(x)");

// Impressão das linhas da tabela
for x = 0:0.2:2*%pi
  printf("\n %3.1f %7.4f", x, sin(x));
End
```

Saída do programa anterior

```
  x    seno(x)
0.0   0.0000
0.2   0.1987
0.4   0.3894
0.6   0.5646
0.8   0.7174
1.0   0.8415
1.2   0.9320
```

"Indentação"

```
if delta < 0 then
  printf('Raízes complexas!');
else
  r1 = (-b + sqrt(delta))/(2*a);
  r2 = (-b - sqrt(delta))/(2*a);
  printf('r1=%g e r2=%g.',r1,r2);
end
```

Mais legível

```
if delta < 0 then
printf('Raízes complexas!');
else
r1 = (-b + sqrt(delta))/(2*a);
r2 = (-b - sqrt(delta))/(2*a);
printf('r1=%g e r2=%g.',r1,r2);
end
```

Menos legível

"Indentação"

- Para o Scilab, os dois programas são absolutamente equivalentes.
- Para nós, a disposição do texto do programa afeta muito a legibilidade .
- Qualquer bloco de comando é mais facilmente identificado com "indentação".
  - Assim, os possíveis fluxos de execução ficam mais claros.

Strings

- Até o momento, as variáveis definidas armazenam apenas valores numéricos ou lógicos.
- Variáveis podem armazenar também valores alfanuméricos (cadeias de caracteres) denominados *strings*.

```
-->a = "Programação";
 a  =
 Programação
-->b = " de ';
 b  =
  de
-->c = "Computadores";
 c  =
 Computadores
```

Aspas simples (') e duplas (") são equivalentes

Concatenação de *strings*

- *Strings* podem ser concatenados (justapostos).

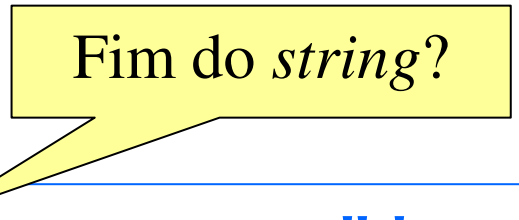
Para *strings*, + significa concatenação

```
-->a = "Programação";
-->b = " de ";
-->c = "Computadores";

-->Disciplina = a + b + c;
 Disciplina  =
 Programação de Computadores
```

Strings contendo aspas

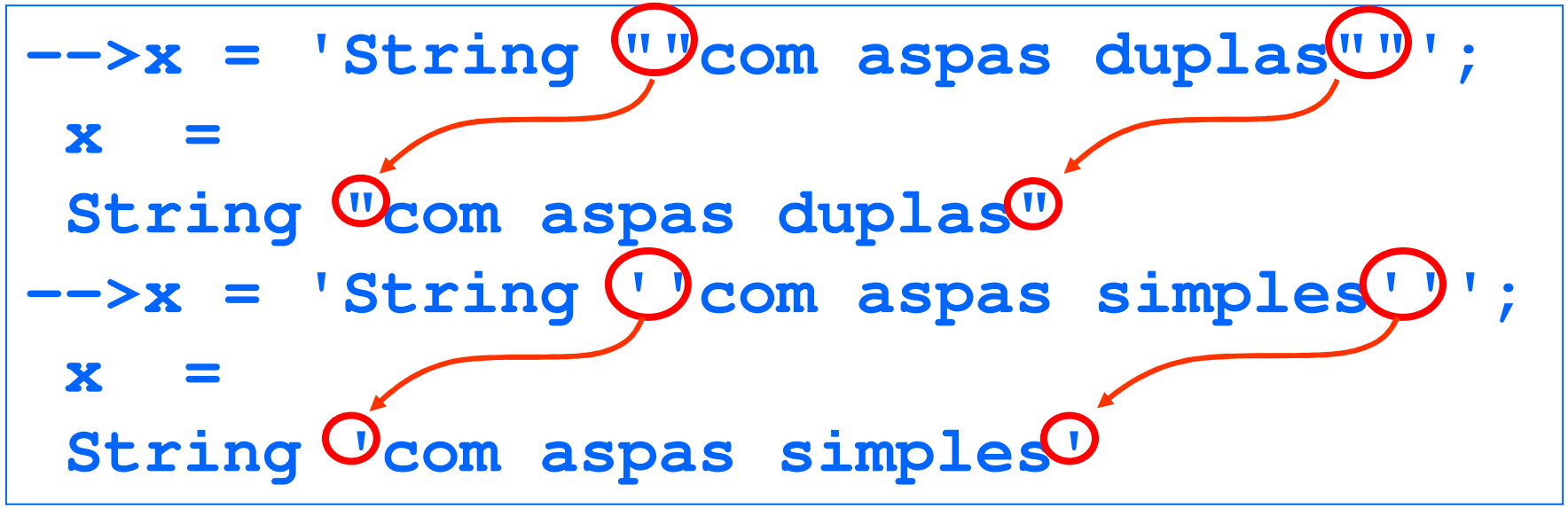
- Como já visto, o Scilab usa aspas para reconhecer o começo e o fim de um *string*.
- Como, então, representar *strings* que contêm aspas?




```
-->x = 'String "com aspas"';
                    !--error 276
Missing operator, comma, or semicolon
```

Strings contendo aspas

- Para representar *strings* com aspas, deve-se colocar duas aspas consecutivas na posição desejada.



Strings de dígitos

- *Strings* formados por dígitos não são valores numéricos.

```
-->format(16)
-->%pi
 %pi  =
    3.1415926535898
-->StringPi = "3.1415926535898"
 StringPi  =
 3.1415926535898
-->2*%pi
 ans  =
    6.2831853071796
-->2*StringPi
    !--error 144
Undefined operation for the given operands
```

Números passam a ser exibidos com 16 posições

Programa: passou - não passou

- Faça um programa em Scilab que:
  - leia o nome de um aluno;
  - leia o total de pontos feitos em uma disciplina pelo aluno;
  - retorne, conforme o caso, uma frase do tipo

    "<aluno>, com <tantos pontos>, você passou!"

    ou

    "<aluno>, com <tantos pontos>, você não passou!".

Programa: passou - não passou

```
//Leitura do nome
printf("Escreva o seu nome ""entre aspas"".\n");
nomealuno = input("Nome: ");
//Leitura dos pontos obtidos
printf ("\n%s, quantos pontos você teve?\n", ...
            nomealuno);
nota = input("Pontos: ");
//Impressão de mensagem com o resultado
if (nota >= 60) then
  printf("Parabéns, %s." + ...
  "\nTendo feito %g pontos, você foi aprovado.\n\n", ...
  nomealuno, nota);
else
  printf("%s, ainda não foi desta vez." + ...
  "\nCom %g pontos, você não foi aprovado.\n\n ", ...
  nomealuno, nota);
end
```

Programa: passou - não passou

Comandos

```
printf("Escreva o seu nome ""entre aspas"".\n");
nomealuno = input("Nome: ");
```

Mudança de linha

Para obter aspas

Efeito

```
Escreva o seu nome "entre aspas".
Nome: Fulano
```

***Bug* do Scilab 5.1.1:**
O *string* não pode conter acentos ou cedilhas.

Programa: passou - não passou

Para imprimir uma variável *string*

"..." indicam ao Scilab que o comando continua na linha seguinte

Comandos

```
printf ("\n%s, quantos pontos você teve?\n", ...
              nomealuno);
nota = input("Pontos: ");
```

Efeito

```
Fulano, quantos pontos você teve?
Pontos: 47
```

Programa: passou - não passou

Comandos

```
if (nota >= 60) then
  printf("Parabéns, %s." + ...
  "\nTendo feito %g pontos, você foi aprovado.\n\n", ...
  nomealuno, nota);
else
  printf("%s, ainda não foi desta vez." + ...
  "\nCom %g pontos, você não foi aprovado.\n\n ", ...
  nomealuno, nota);
end
```

Efeito

```
Fulano, ainda não foi desta vez.
Com 47 pontos, você não foi aprovado.
```

Processo de repetição

```
continua = %T;
while continua
  // Comandos quaisquer

  // Decisão sobre a continuação do programa
  decisao = ...
     input("Deseja continuar?(s/n)", "string");
  continua = decisao == "s";
end
printf ("Término da repetição.\n");
```

Parâmetro extra do `input` que elimina a necessidade de aspas ao entrar com *string*

Processo de repetição

```
// Cálculo das raízes de diversas equações de 2o grau
continua = %t;
while continua
  a = input("Digite o valor de a:");
  b = input("Digite o valor de b:");
  c = input("Digite o valor de c:");
  delta = b^2 - 4*a*c;
  if delta >= 0 then
    x1 = (-b+sqrt(delta))/(2*a);
    x2 = (-b-sqrt(delta))/(2*a);
    printf ("As raízes são %g e %g", x1, x2);
  else
    printf ("As raízes são complexas");
  end
  // Decisão de continuação pelo usuário
  decisao = input("Outra equação? (s/n)", "string");
  continua = decisao == "s";
end
Printf ("\nTérmino do programa");
```

Comandos aninhados

- Blocos internos a comandos condicionais e comandos de repetição podem conter qualquer tipo de comando, incluindo:
  - comandos de atribuição;
  - comandos de entrada e saída de dados;
  - outros comandos condicionais e de repetição.
- Esta generalidade proporciona uma imensa flexibilidade à programação.

Comandos aninhados

- Por exemplo, blocos “então” ou “senão” de *ifs* podem conter qualquer tipo de comando, inclusive outros *ifs*.

```
if <condição 1> then
    // comandos
    if <condição 2> then
        // comandos
    else
        if <condição 3> then
            // comandos
        end
    end
end
```

Programa: conceitos e notas

- Faça um programa Scilab que:
  - leia o nome e a nota de um aluno em uma determinada disciplina;
  - retorne o conceito correspondente, segundo a tabela:

Resultado	Conceito
90 <= Nota <= 100	A
80 <= Nota < 90	B
70 <= Nota < 80	C
60 <= Nota < 70	D
40 <= Nota < 60	E
0 <= Nota < 40	F

Programa: conceitos e notas

```
// leitura e validação dos dados de entrada
if Nota >= 90 then
  Conceito = "A";
else
  if Nota >= 80 then
    Conceito = "B";
  else
    if Nota >= 70 then
      Conceito = "C";
    else
      if Nota >= 60 then
        Conceito = "D";
      else
        if Nota >= 40 then
          Conceito = "E";
        else
          Conceito = "F";
        end
      end
    end
  end
end
// apresentação do resultado
```

A importância da "indentação"

```
if Nota >= 90 then
  Conceito = 'A';
else
  if Nota >= 80 then
    Conceito = 'B';
  else
    if Nota >= 70 then
      Conceito = 'C';
    else
      if Nota >= 60 then
        Conceito = 'D';
      else
        if Nota >= 40 then
          Conceito = 'E';
        else
          Conceito = 'F';
        end
      end
    end
  end
end
```

Mais legível

```
if Nota >= 90 then
Conceito = 'A';
else
if Nota >= 80 then
Conceito = 'B';
else
if Nota >= 70 then
Conceito = 'C';
else
if Nota >= 60 then
Conceito = 'D';
else
if Nota >= 40 then
Conceito = 'E';
else
Conceito = 'F';
end
end
end
end
end
```

Menos legível

Programa: tabuada

- Faça um programa em Scilab que gere a seguinte tabela de tabuada de multiplicação:

```
1  2  3  4  5  6  7  8  9
2  4  6  8 10 12 14 16 18
3  6  9 12 15 18 21 24 27
4  8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

Programa: tabuada – 1ª versão

```
// Tabuada de multiplicação
for linha = 1:9
   for coluna = 1:9
      printf("%g",linha*coluna);
   end
end
```

Corpo do *loop* externo: imprime uma linha

Corpo do *loop* interno: imprime uma coluna de uma linha

Programa: tabuada

- Ao executar o programa anterior, verifica-se a saída não está legível:

  1234567892468101214161836912151821 2 ...

- É preciso:
  - após a impressão de uma linha, mudar de linha com o `\n`;
  - dentro de cada linha, imprimir cada valor em um número fixo de colunas.

Programa: tabuada – 2ª versão

```
// Tabuada de multiplicação
for linha = 1:9
   for coluna = 1:9
      printf ("%3g",linha*coluna);
   end
   printf("\n");
end
```

Código de formatação

Fora do loop interno!