`if` Constructs in MATLAB

Gerald W. Recktenwald
Department of Mechanical Engineering
Portland State University
gerry@pdx.edu

Relational Operators (1)

Relational operators are used in comparing two values.

Operator	Meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
~=	not equal to

The result of applying a relational operator is a logical value, i.e. either *true* or *false*.

In MATLAB any nonzero value, including a non-empty string, is equivalent to *true*. Only zero is equivalent to *false*.

Note: The <=, >=, and ~= operators have "=" as the *second* character.
=<, => and =~ are *not* valid operators.

Relational Operators (2)

The result of a relational operation is a true or false value.

Examples:

```
>> a = 2;  b = 4;
>> aIsSmaller = a < b
aIsSmaller =
     1

>> bIsSmaller = b < a
bIsSmaller =
     0
```

Relational operations can also be performed on matrices of the same shape, e.g.,

```
>> x = 1:5;   y = 5:-1:1;
>> z = x>y
z =
     0     0     0     1     1
```

Logical Operators

Logical operators are used to combine logical expressions with logical "and" or logical "or", or to reverse a logical value with "not"

Operator	Meaning
&&	and
||	or
~	not

Examples:

```
>> a = 2;  b = 4;
>> aIsSmaller = a < b;
>> bIsSmaller = b < a;
>> bothTrue = aIsSmaller && bIsSmaller
bothTrue =
     0

>> eitherTrue = aIsSmaller || bIsSmaller
eitherTrue =
     1

>> ~eitherTrue
ans =
     0
```

Logical and Relational Operators

Summary:

- Relational operators involve comparison of two values.
- The result of a relational operation is a logical (True/False) value.
- Logical operators combine (or negate) logical values to produce another logical value.
- There is always more than one way to express the same comparison

Free Advice:

- To get started, focus on simple comparison. Do not be afraid to spread the logic over multiple lines (multiple comparisons) if necessary.
- Try reading the test out loud.

Conditional Execution

Conditional Execution or Branching:

As the result of a comparison, or another logical (true/false) test, selected blocks of program code are executed or skipped.

Conditional execution is implemented with `if`, `if...else`, and `if...elseif` constructs, or with a `switch` construct.

There are three types of `if` constructs

1. Plain `if`
2. `if...else`
3. `if...elseif`

`if` Constructs

Syntax:

```
if expression
    block of statements
end
```

The *`block of statements`* is executed only if the *`expression`* is true.

Example:

```
if a < 0
   disp('a is negative');
end
```

The *one line* format uses comma after `if` *`expression`*

```
if a < 0,  disp('a is negative'); end
```

if...else

Multiple choices are allowed with `if...else` and `if...elseif` constructs

```
if x < 0
   error('x is negative; sqrt(x) is imaginary');
else
   r = sqrt(x);
end
```

`if...elseif`

It's a good idea to include a default `else` to catch cases that don't match preceding `if` and `elseif` blocks

```
if x > 0
   disp('x is positive');
elseif x < 0
   disp('x is negative');
else
   disp('x is exactly zero');
end
```

Using `if` and `nargin` to supply default values

```
function drawCircle(r,x0,y0,line_style)
% drawCircle  Draw a circle in the (x,y) plane
%
% Synopsis:  drawCircle(r)
%            drawCircle(r,x0)
%            drawCircle(r,x0,y0)
%            drawCircle(r,x0,y0,line_style)
%
% Input:  r = radius of the circle
%         x0,y0 = (optional) x and y coordinates of center of the circle
%                 Default:  x0 = 0,  y0 = 0;
%         line_style = (optional) string used to specify the line style
%                      of the circle.  Default: line_style = '-' draw
%                      the circle with a solid line
if nargin<2, x0 = 0;  end
if nargin<3, y0 = 0;  end
if nargin<4, line_style = '-';  end

t = linspace(0,2*pi);
x = x0 + r*cos(t);
y = y0 + r*sin(t);
plot(x,y,line_style)
end
```

The `switch` Construct

A switch construct is useful when a test value can take on discrete values that are either integers or strings.

Syntax:

```
switch  expression
  case value1,
    block of statements
  case value2,
    block of statements
    ⋮
  otherwise,
    block of statements
end
```

Example:

```
color = '...';    %  color is a string
switch color
    case 'red'
        disp('Color is red');
    case 'blue'
        disp('Color is blue');
    case 'green'
        disp('Color is green');
    otherwise
        disp('Color is not red, blue, or green');
end
```