Logical indexing

- you can use a logical array to perform indexing on another array
  - MATLAB extracts the array elements corresponding to the nonzero values in the logical array
    - the output is always in the form of a column vector unless the array is a vector

```
>> x = 1:5
x =
     1     2     3     4     5

>> I = x > 3
I =
     0     0     0     1     1

>> x(I)
ans =
     4     5
```

```
>> X = pascal(5)
X =
     1     1     1     1     1
     1     2     3     4     5
     1     3     6    10    15
     1     4    10    20    35
     1     5    15    35    70

>> I =

     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     1
     0     0     0     1     1
     0     0     1     1     1
```

```
>>  X(I)
ans =
    15
    20
    35
    15
    35
    70
```

```
>> % rectify a sine wave
>> t = 0:0.05:1;
>> y = sin(t);
>> plot(t, y, 'b'); hold on;
>> I = y < 0;
>> y(I) = -y(I);
>> plot(t, y, 'r');

>> % replace all spaces with –
>> s = 'a string with some spaces in it';
>> s(isspace(s)) = '-'
s =
a-string-with-some-spaces-in-it
```

Selection statements

Selection statements

- so far, all of our scripts and functions have been made up of statements that are executed in sequence
- a selection statement uses a logical value or values to select which statements are executed

if statement

- the if statement allows you to conditionally execute a *single* block of MATLAB statements

if statement

```
if   logical condition
     sequence of MATLAB statements
end
more MATLAB statements
```

if the logical condition is true then

do the sequence of MATLAB statements and then

do more MATLAB statements

if statement

```
if   logical condition
     sequence of MATLAB statements
end
more MATLAB statements
```

if the logical condition is false then

do more MATLAB statements

if-statement

- find the absolute value of a number **`x`**

```
if x < 0
  x = -x
end
```

- don't do this, use **`abs`** instead

if-statement

- print out an error message

```
if degKelvin < 0
  error('impossible temperature!');
end
```

if-statement

- change a column vector to a row vector

```
if size(x, 1) > 1
  x = x';
end
```

- change a row vector to a column vector

```
if size(x, 2) > 1
  x = x';
end
```

if-statement

- create a function that has an optional last parameter

```
function [x, v] = dhmotion(A, m, k, b, phi, t)
%DHMOTION Damped harmonic motion from Lab 4
%   x = dhmotion(A, m, k, b, phi) computes
%   the position of a damped harmonic oscillator
%   at time t = linspace(0, 1)

if nargin == 5
  t = linspace(0,1);
end

% the rest of your function goes here...
```

nargin is the number of input arguments supplied by the caller

if-else statement

- the if-else statement allows you to conditionally execute one of *two* blocks of MATLAB statements

if-else statement

```
if    logical condition
      sequence of MATLAB statements
else
      sequence of MATLAB statements
end
more MATLAB statements
```

if the logical condition is true then

do the first sequence of MATLAB statements and then

do more MATLAB statements

if-else statement

```
if    logical condition
      sequence of MATLAB statements
else
      sequence of MATLAB statements
end
more MATLAB statements
```

if the logical condition is false then

do the second sequence of MATLAB statements and then

do more MATLAB statements

if-else statement

- write a function that returns **`true`** if an integer number is even, and **`false`** if it is even

```
function [tf] = iseven(x)
%ISEVEN True for an even number
%   TF = ISEVEN(X) returns 1 if X is an even integer number
%   and 0 otherwise.

if rem(x, 2) == 0
    tf = 1;
else
    tf = 0;
end
```

`rem(x, 2)` is the remainder of **`x / 2`**

if-else statement

- write a function that returns a sorted vector of two numbers

```
function [sorted] = sort2(x, y)
%SORT2 Sort two numbers in ascending order
%   SORTED = SORT2(X, Y) returns a row vector containing
%   the numbers X and Y in ascending order

if x <= y
    sorted = [x y];
else
    sorted = [y x];
end
```

don't do this; use **`sort`** instead

if-elseif statement

- the if-elseif statement allows you to conditionally execute one of *multiple* blocks of MATLAB statements

if-else statement

```
if    logical condition 1
      sequence of MATLAB statements
elseif logical condition 2
      sequence of MATLAB statements
else
      sequence of MATLAB statements
end
more MATLAB statements
```

if the logical condition 1 is true then

do the first sequence of MATLAB statements and then

do more MATLAB statements

if-elseif statement

```
if      logical condition 1
        sequence of MATLAB statements
elseif  logical condition 2
        sequence of MATLAB statements
else
        sequence of MATLAB statements
end
more MATLAB statements
```

if logical condition 1 is false then

if logical condition 2 is true then

do the second sequence of MATLAB statements and then

do more MATLAB statements

if-elseif statement

```
if    logical condition 1
      sequence of MATLAB statements
elseif logical condition 2
      sequence of MATLAB statements
else
      sequence of MATLAB statements
end
more MATLAB statements
```

if logical condition 1 is false then

if logical condition 2 is false then

do the third sequence of MATLAB statements and then

do more MATLAB statements

if-elseif statement

- write a function that returns a sorted vector of three numbers

```
function [s] = smallest(x, y, z)
%SMALLEST Find the smallest of 3 numbers
%   S = SMALLEST(X, Y, Z) returns the minimum of X, Y, and Z

if x <= y & x <= z
    s = x;
elseif y <= x & y <= z
    s = y;
else
    s = z;
end
```

don't do this; use **`min`** instead