Introduction to MATLAB

Programming and Logic Operations

Logic and Relational Operations

- The ability to test conditional statements and make decisions is a fundamental programming concept
  - For example: Is A greater than B? Is T equal to Z OR is T less than 5?
- Often used to control program flow
  - For example:
    - If C equals D and D does not equal 5 do something.
    - Did the user enter the right type of data?
    - While T is less than 10 add this number together.
- To do this, MATLAB compares two arguments (e.g. “A” and “B”) and returns a “true” or false” value.

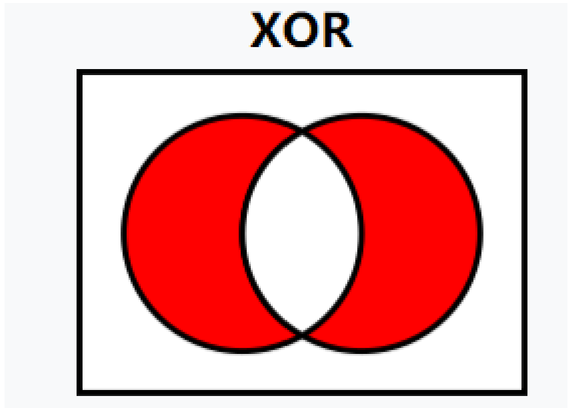
Operators

- Arithmetic operators

`+`, `-`, `*`, `/`, `\`, `^`, etc.

- Logical operators

`&`, `|`, `~`, `xor( )`, etc.



- Relational operators

`==`, `~=`, `<`, `>`, `<=`, `>=`, etc.

Relational Operations

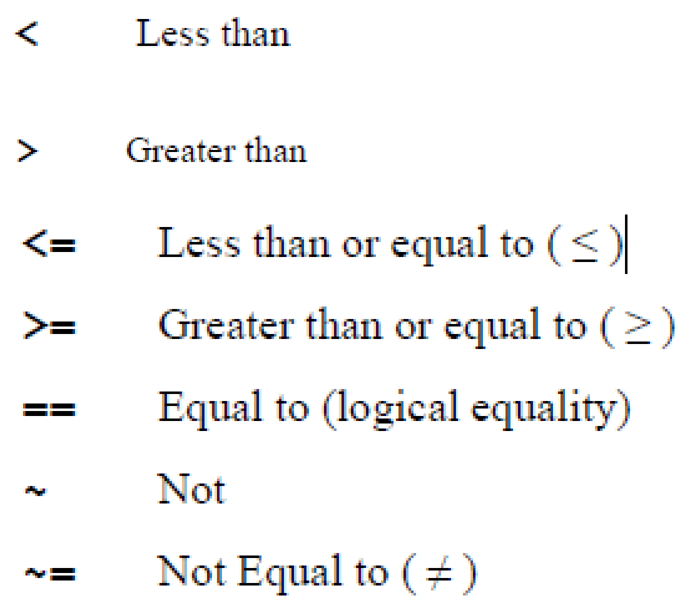
- Relational operators perform element-by element comparisons between two scalars or two arrays (must be same size).

- They return a logical array of the same size

- Elements are set to logical 1 (true) where the relation is true

- Elements set to logical 0 (false) where it is not

Relational Operators

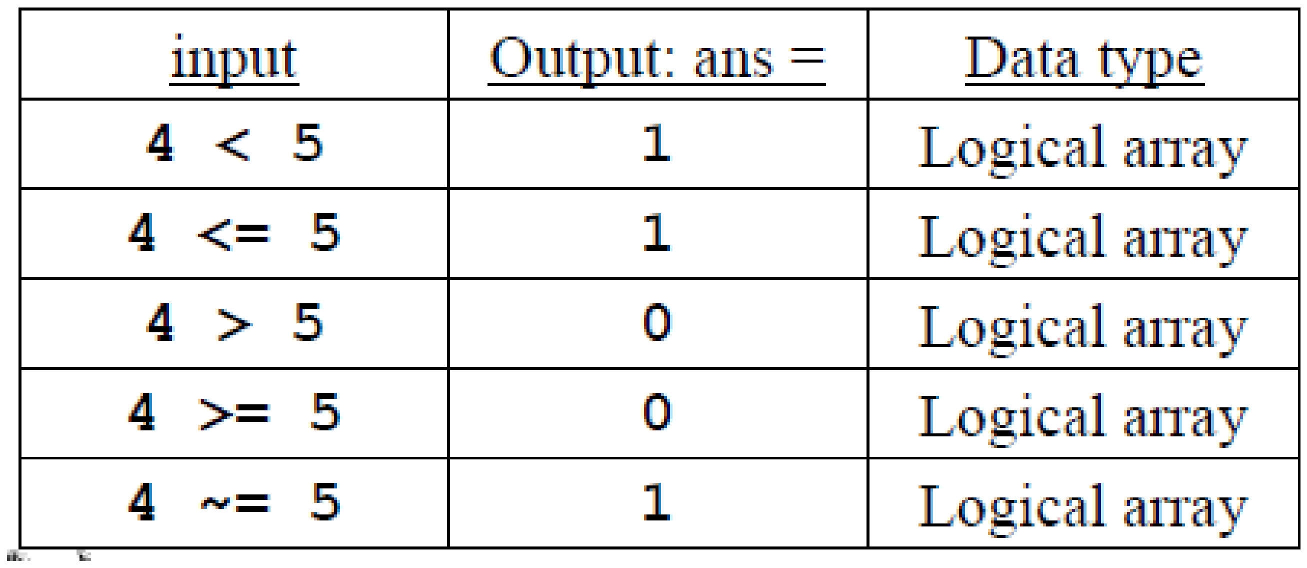


<	Less than
>	Greater than
<=	Less than or equal to ($\leq$)
>=	Greater than or equal to ($\geq$)
==	Equal to (logical equality)
~	Not
~=	Not Equal to ($\neq$)

MATLAB is a bit picky about spaces around relational operators
No spaces, a space after the operator, or a space on both sides work
A space before the operator and none after does not work

Logical Data Type

- The ***logical data type*** is a another MATLAB data type; it is either equal to 1 (true) or equal to 0 (false).

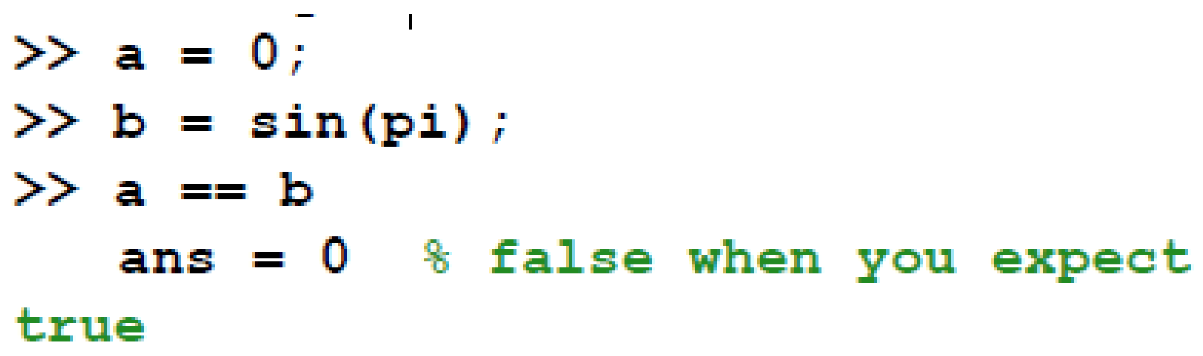


input	Output: ans =	Data type
`4 < 5`	`1`	Logical array
`4 <= 5`	`1`	Logical array
`4 > 5`	`0`	Logical array
`4 >= 5`	`0`	Logical array
`4 ~= 5`	`1`	Logical array

Relational Operations and Round-off Errors

- The **==** and **~=** commands can produce puzzling results due to round-off error.

- For example:



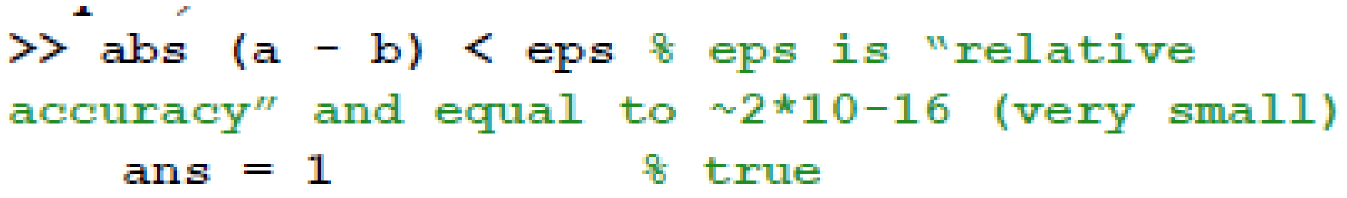
```
>> a = 0;
>> b = sin(pi);
>> a == b
    ans = 0   % false when you expect
true
```

What happened?

- MATLAB computes sin(pi) to within eps, i.e, it computes a number approximately zero

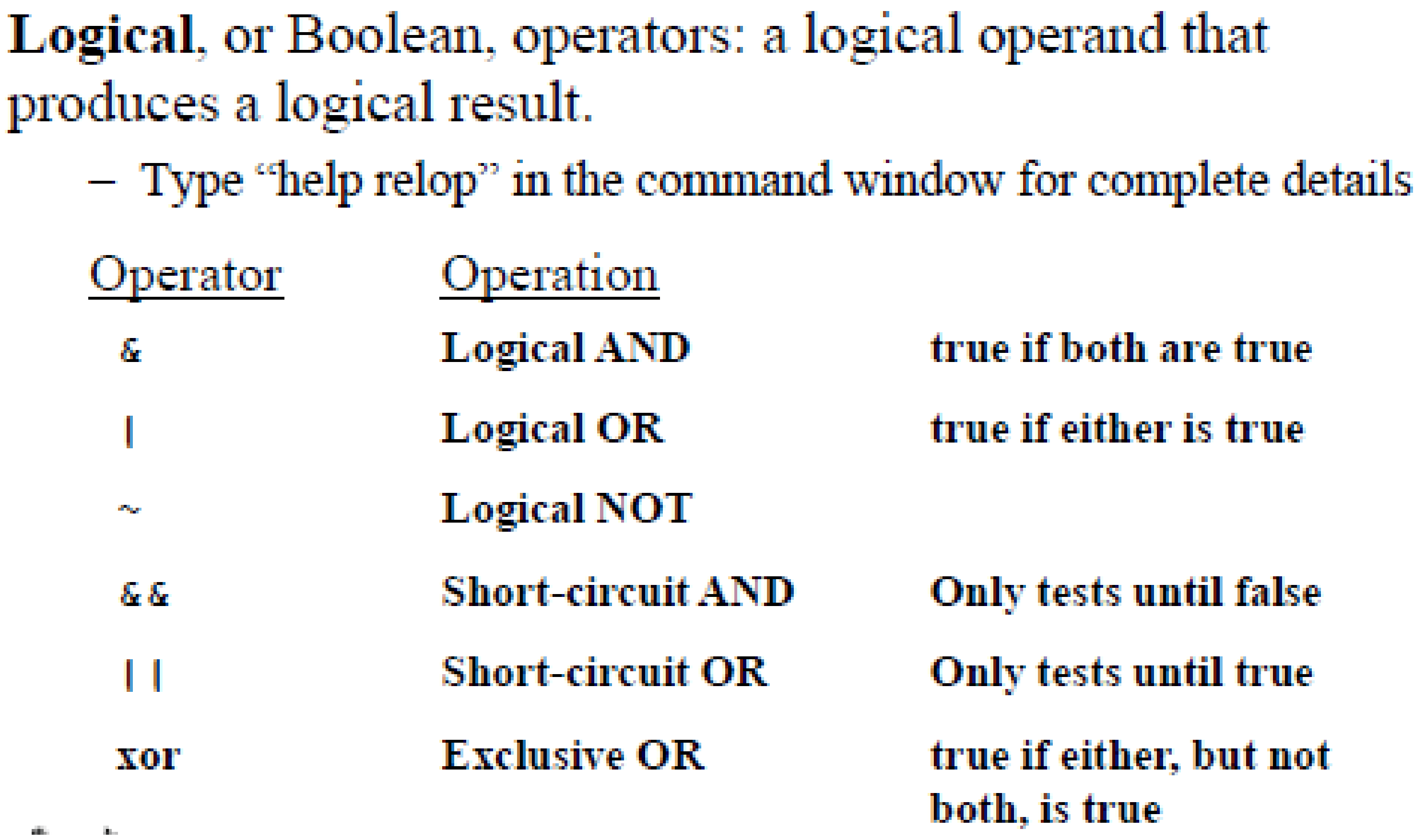
- The == operator properly concludes that 0 and the computed value of sin(p) are different

- The best way to fix this is to check if they are approximately equal (within round off error “eps”)



```
>> abs (a - b) < eps % eps is "relative
accuracy" and equal to ~2*10-16 (very small)
    ans = 1           % true
```

Logical Operators



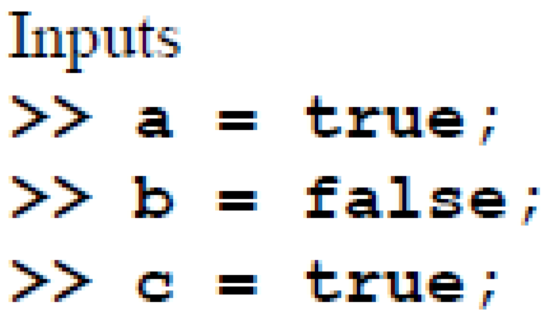
Logical, or Boolean, operators: a logical operand that produces a logical result.

- Type "help relop" in the command window for complete details

Operator	Operation	
&	**Logical AND**	**true if both are true**
|	**Logical OR**	**true if either is true**
~	**Logical NOT**	
&&	**Short-circuit AND**	**Only tests until false**
||	**Short-circuit OR**	**Only tests until true**
xor	**Exclusive OR**	**true if either, but not both, is true**

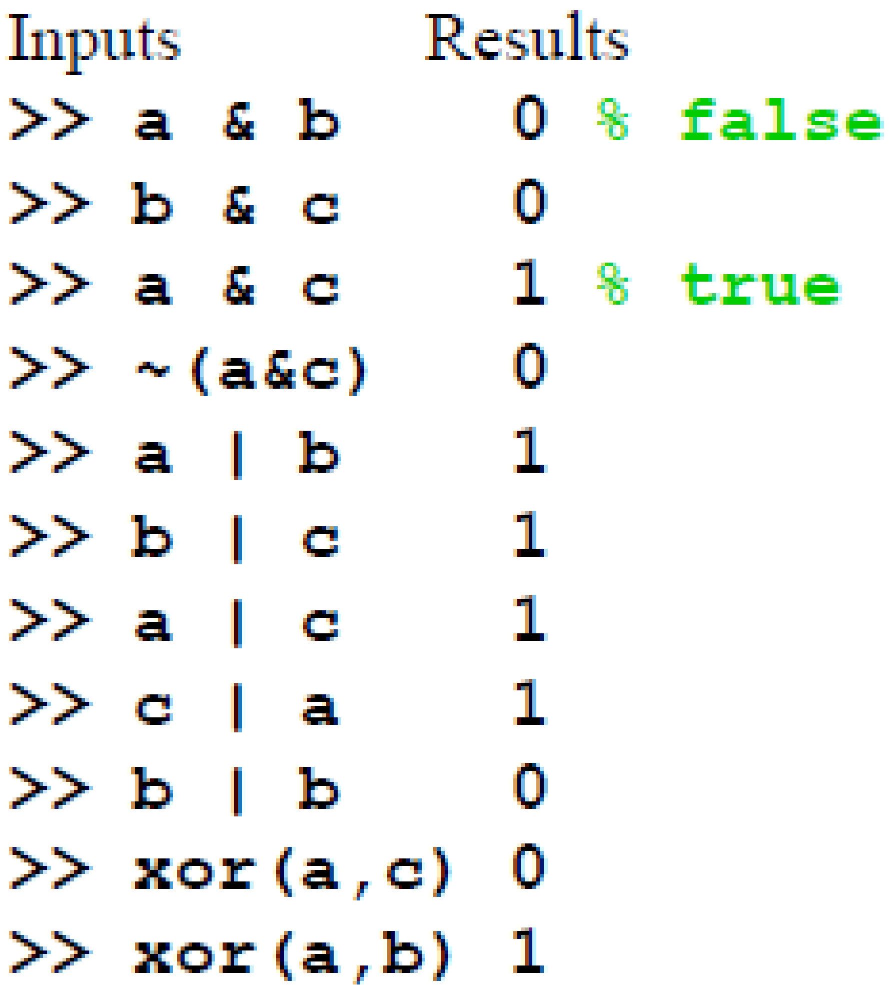
expr1 && expr2
expr2 is not evaluated if expr1 is false
expr1 || expr2
expr2 is not evaluated if expr1 is true

Logical Operator Examples



Inputs

```
>> a = true;
>> b = false;
>> c = true;
```



Inputs   Results

```
>> a & b      0 % false
>> b & c      0
>> a & c      1 % true
>> ~(a&c)     0
>> a | b      1
>> b | c      1
>> a | c      1
>> c | a      1
>> b | b      0
>> xor(a,c)  0
>> xor(a,b)  1
```

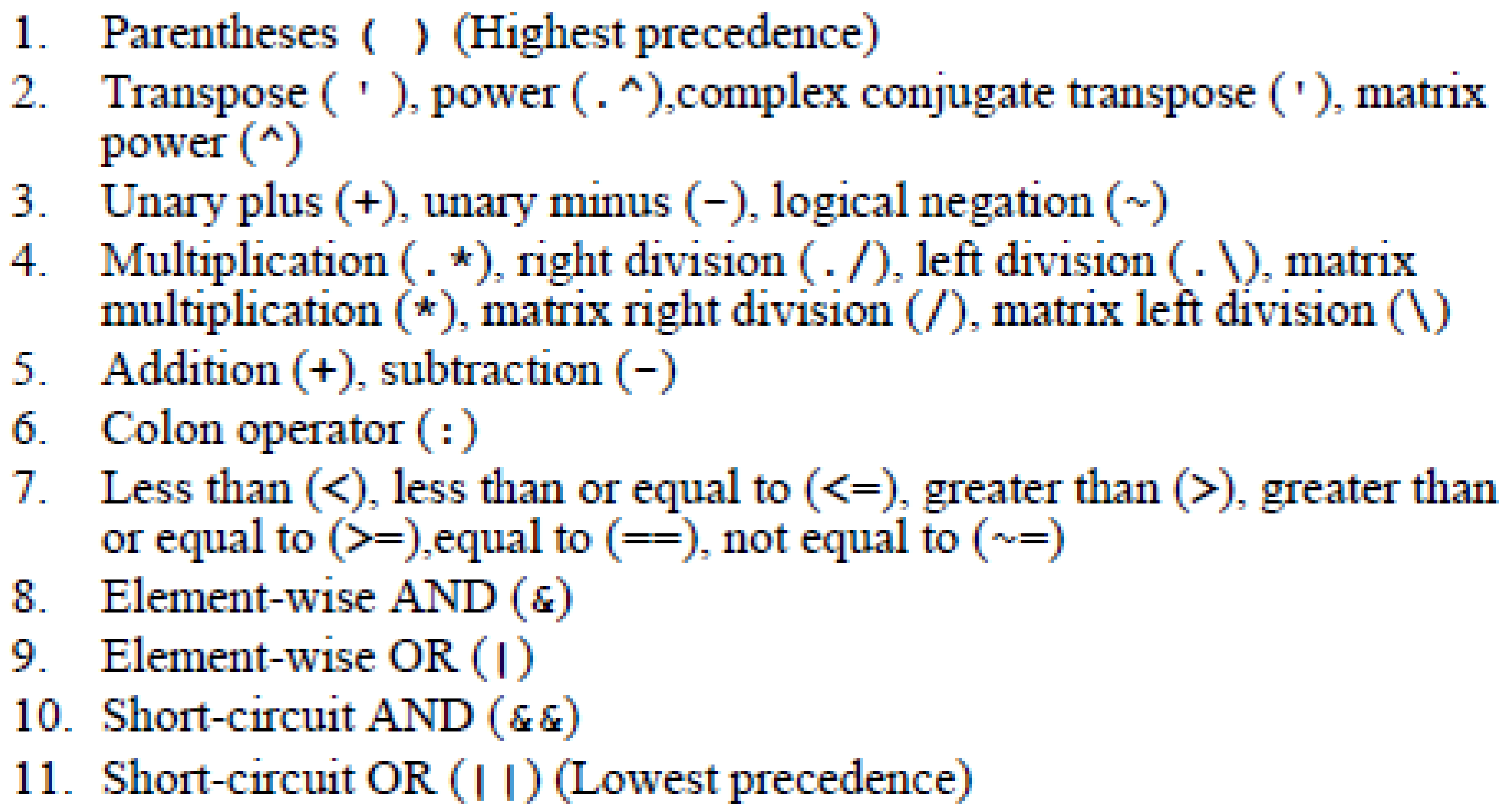
Hierarchy Of Operations

- Logical Operators are evaluated **after** arithmetic and relational operations.

Order of Operations

1. Arithmetic
2. Relational Operators
3. All ~ (Not) operators
4. All & (And) operators evaluated from left to right
5. All | (Or) operators evaluated from left to right

Operator Precedence



1. Parentheses ( ) (Highest precedence)
2. Transpose ('), power (. ^),complex conjugate transpose ('), matrix power (^)
3. Unary plus (+), unary minus (−), logical negation (~)
4. Multiplication (. *), right division (. /), left division (. \), matrix multiplication (*), matrix right division (/), matrix left division (\)
5. Addition (+), subtraction (−)
6. Colon operator (:)
7. Less than (<), less than or equal to (<=), greater than (>), greater than or equal to (>=),equal to (==), not equal to (~=)
8. Element-wise AND (&)
9. Element-wise OR (|)
10. Short-circuit AND (&&)
11. Short-circuit OR (||) (Lowest precedence)

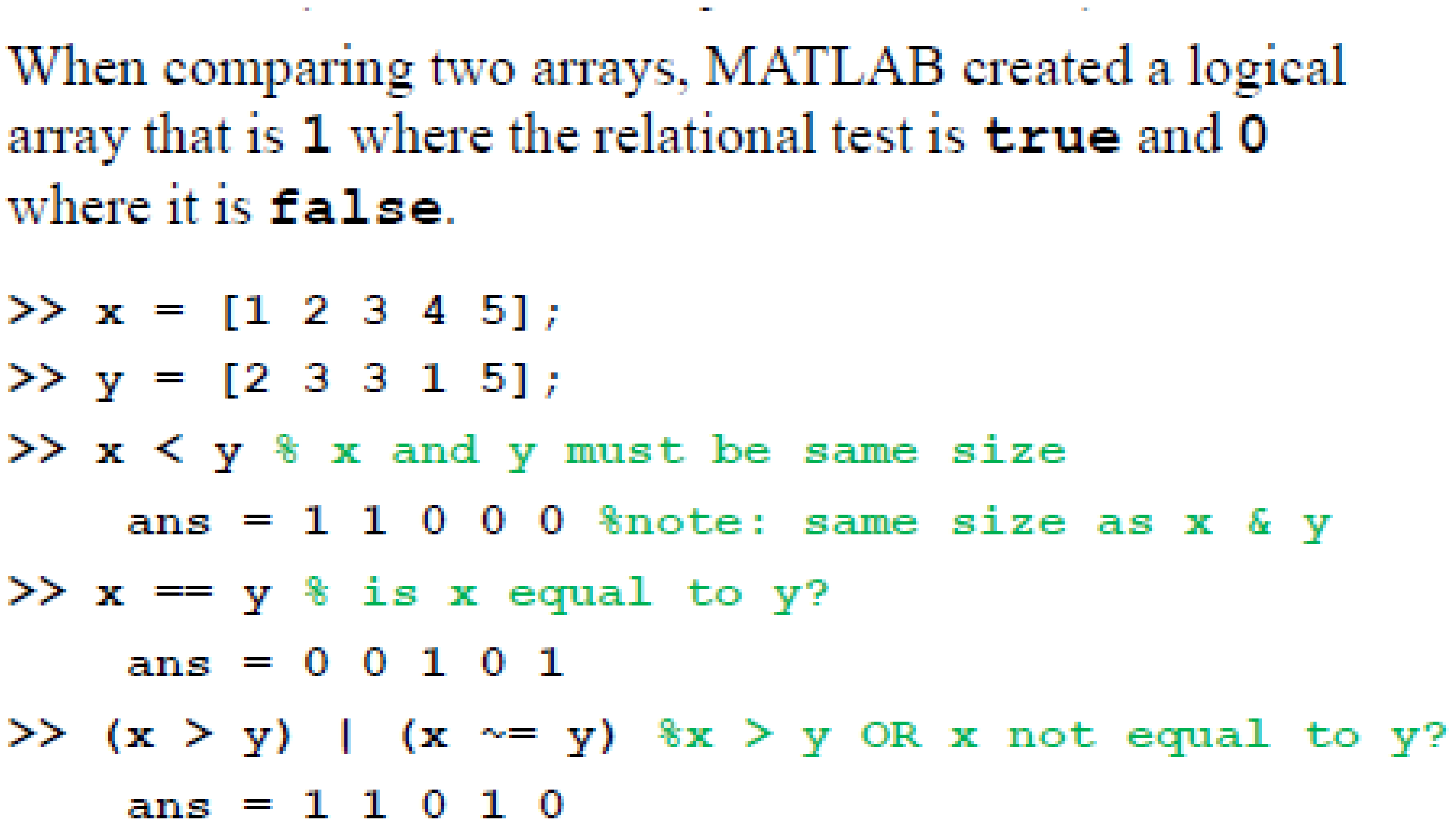
Combination of Operators

- Operators are often combined
- For example:

```
>> a = 1; b = 2; c = 3; d = 2; e = -3;
>> (a < b) & (b < c)    % 1 (true)
>> (a == b) | (b == d)  % 1 (true)
>> (a > c) | (b < d)    % 0 (false)
>> -5 <= e <= -2        % 0 (false)
***WARNING** doesn't work like you think.
Use (-5 <= e) & (e <= -2) instead since (-5 <= e)
can only be 1 or 0 which is always than -2.
```

- For complex arrangements it is often a good idea to break into a series of simpler tests (or use " ( ) ") and then combine so they are easier to debug

Logical Operator and Arrays (element by element)



When comparing two arrays, MATLAB created a logical array that is **1** where the relational test is **true** and **0** where it is **false**.

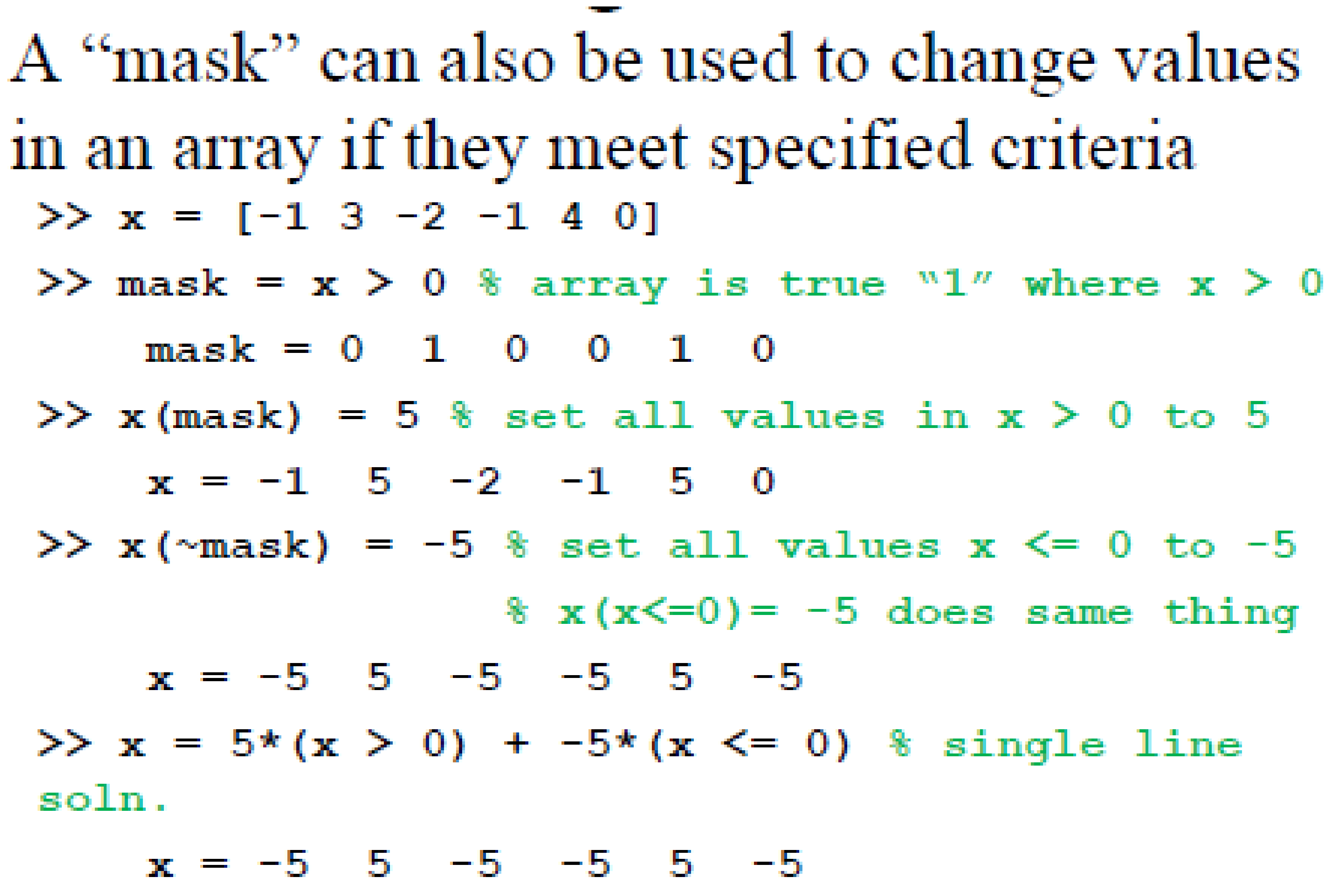
```
>> x = [1 2 3 4 5];
>> y = [2 3 3 1 5];
>> x < y % x and y must be same size
    ans = 1 1 0 0 0 %note: same size as x & y
>> x == y % is x equal to y?
    ans = 0 0 1 0 1
>> (x > y) | (x ~= y) %x > y OR x not equal to y?
    ans = 1 1 0 1 0
```

Masking

- A “mask” can be used to extract values from an array that meet specified criteria

```
>> x = [-1 3 -2 -1 4 0]
>> mask1 = x > 0 % array is true "1" where x > 0
    mask = 0   1   0   0   1   0
>> pos_x = x(mask1) % extract only values > 0
    pos_x = 3   4
>> neg_x = x(~mask1) % extract only values <= 0
                     % x(x<=0) does the same thing
    neg_x = -1   -2   -1   0
```

Masking Continued



A "mask" can also be used to change values in an array if they meet specified criteria

```
>> x = [-1 3 -2 -1 4 0]
>> mask = x > 0 % array is true "1" where x > 0
    mask = 0   1   0   0   1   0
>> x(mask) = 5 % set all values in x > 0 to 5
    x = -1   5   -2   -1   5   0
>> x(~mask) = -5 % set all values x <= 0 to -5
                 % x(x<=0)= -5 does same thing
    x = -5   5   -5   -5   5   -5
>> x = 5*(x > 0) + -5*(x <= 0) % single line
soln.
    x = -5   5   -5   -5   5   -5
```

Logical Functions

- Logical functions are MATLAB functions that implement logical operations.
- **`find( )`** returns the index number of the array that meet the given logical statement

```
>> v = [1, 3, 5, 6; 3, 4, 5, 7; 4, 2, 3, 9];
>> [row, col] = find(v == 4)
row = 3 2, col = 1 2 % v(3,1) and v(2,2) are 4
>> element = find(v == 4)
element = 3 5 % 4 is the 3rd and 5th element in v
```

Other Logical Functions

• **`ischar( )`**	returns **1** if **( )** contains character data
• **`isinf( )`**	returns **1** if **( )** contains infinity (**`Inf`**)
• **`isnan( )`**	returns **1** if **( )** contains not a number (**`NaN`**)
• **`isnumeric( )`**	returns **1** if **( )** contains numeric data
• **`isempty( )`**	returns **1** if **( )** contains empty (**`x = [ ]`**)

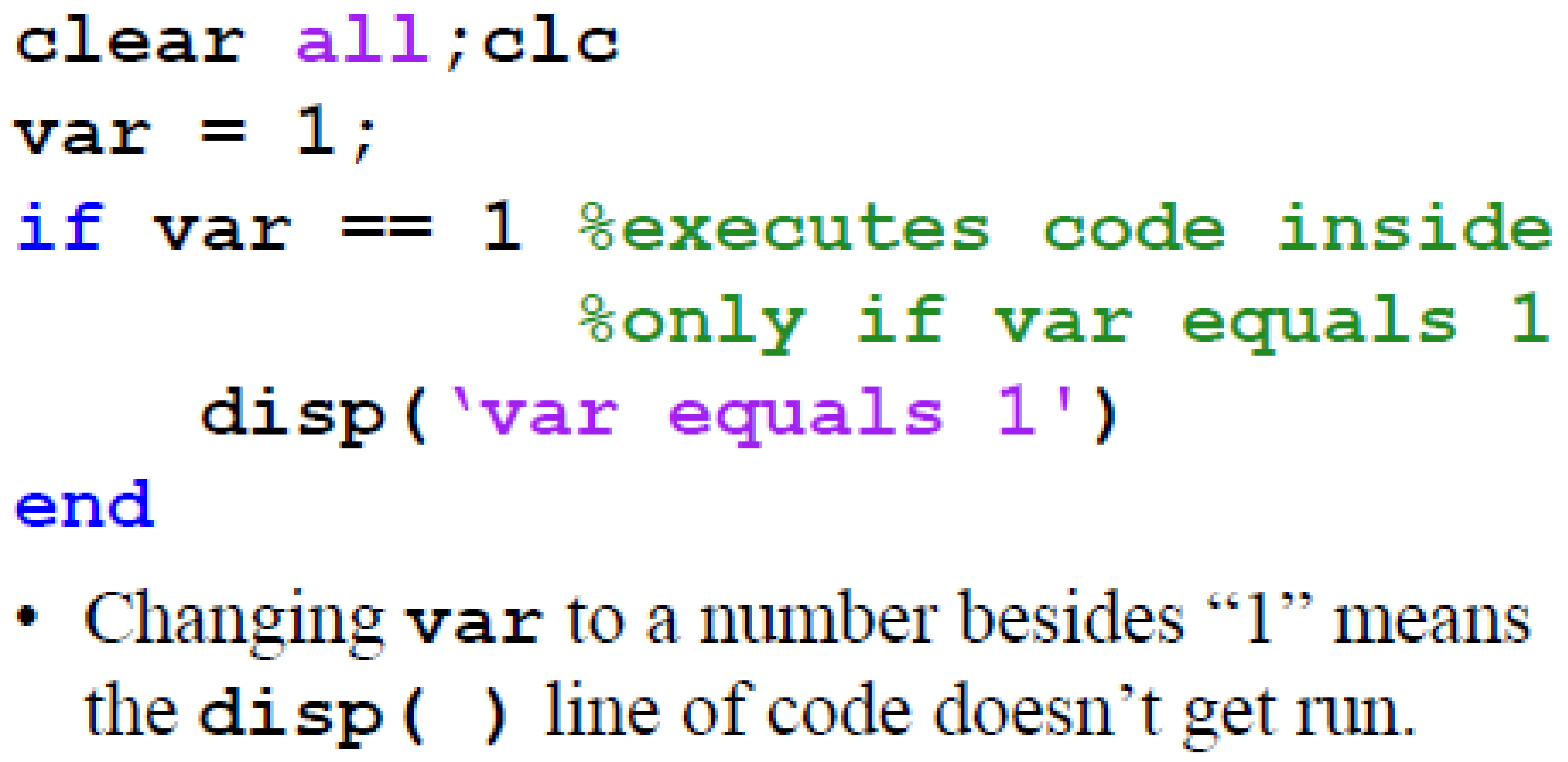
Logic Control and Looping

- Decisions about what to do next
  - Using an if statement
    - E.g., `if-end`, `if-else-end`, `if-elseif-else-end`, etc
  - Using `switch` statement
    - E.g., `switch-case-otherwise-end`
- Looping – repeating a series of commands
  - Using `for` statement
    - E.g., `for-end`
  - Using `while` statement
    - E.g., `while-end`

Logic Control & Looping Comments

- This is a very powerful set of tools
- Power == Often hard to fully control
- It often takes much longer get the logic right than it takes to get the math & display right
  - Use **`disp( )`** and **`fprintf( )`** to provide info about how the program is actually running
- Infinite loops are fairly common
  - Use ctrl-c to stop the program

Example if Statement



```
clear all;clc
var = 1;
if var == 1 %executes code inside
            %only if var equals 1
    disp('var equals 1')
end
```

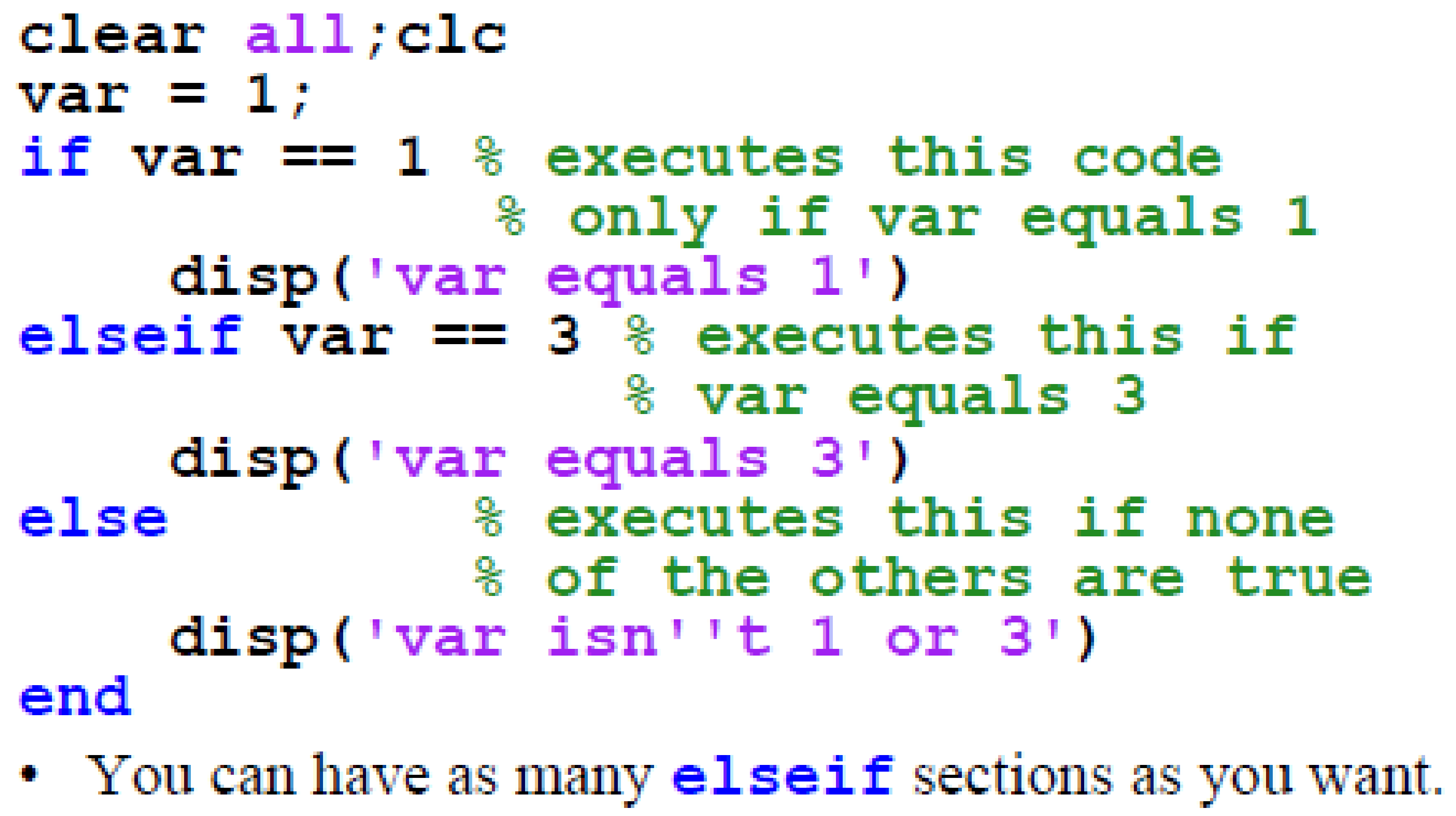
- Changing **`var`** to a number besides “1” means the **`disp( )`** line of code doesn’t get run.

Example if-else Statement

```
clear all;clc;
var = 1;
if var == 1 %executes code inside
          %only if var equals 1
    disp(‘var equals 1')
else % executes this if
     % var doesn't equal 1
    disp(‘var isn''t 1')
end
```

- The `else` portion ensures at least something is always done no matter what the value of `var` is. Change `var` to another value to change the output.

Example if-elseif-else Statement



```
clear all;clc
var = 1;
if var == 1 % executes this code
             % only if var equals 1
    disp('var equals 1')
elseif var == 3 % executes this if
                 % var equals 3
    disp('var equals 3')
else            % executes this if none
                % of the others are true
    disp('var isn''t 1 or 3')
end
```

- You can have as many **elseif** sections as you want.

Example switch Statement

```
clear all;clc;
var = 1
switch var
    case 1 %does this if var == 1
        disp('var equals 1')
    case 3 %does this if var == 3
        disp('var equals 3'
    otherwise %does this otherwise
        disp(‘No match, var isn''t 1 or 3')
end
```

- The switch input (**`var`** in this instance) must be either a scalar or a character string
- Can use an **`if-elseif-else`** statement to do the same thing

Example for Statement

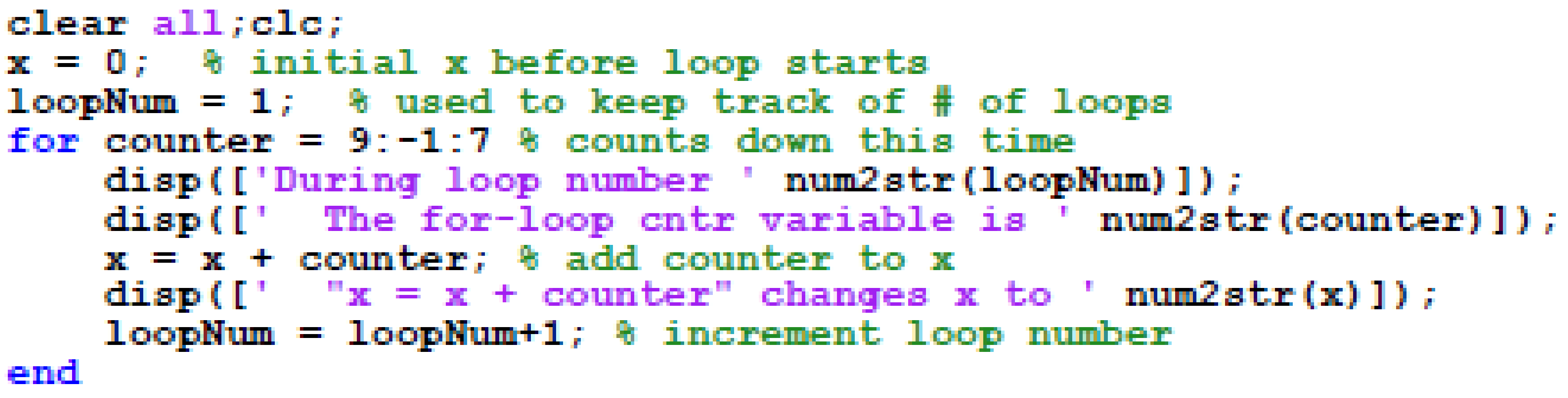
```
for k = 1:5
    disp(['k is now ' num2str(k)])
end
```

The `for` loop create a counter variable, **k**, that changes each time the loop executes. The code loops 5 times and the output is:

```
k is now 1
k is now 2
k is now 3
k is now 4
k is now 5
```

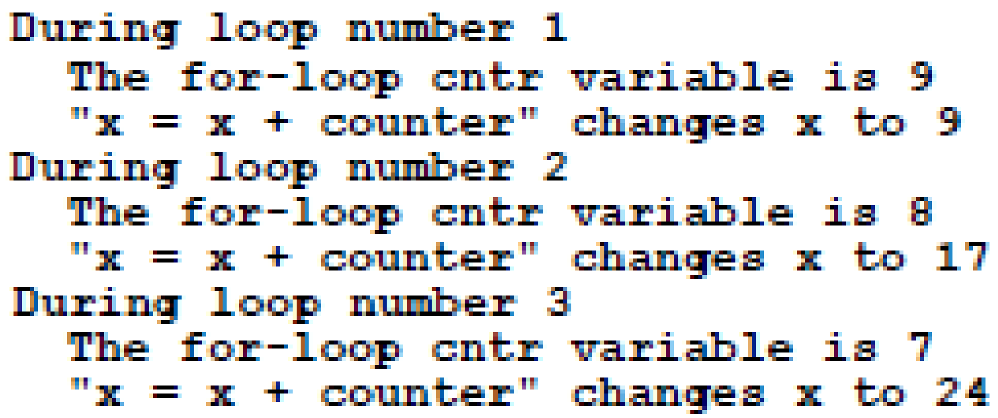
Use when you know exactly how many times you want to loop.

Another for Example



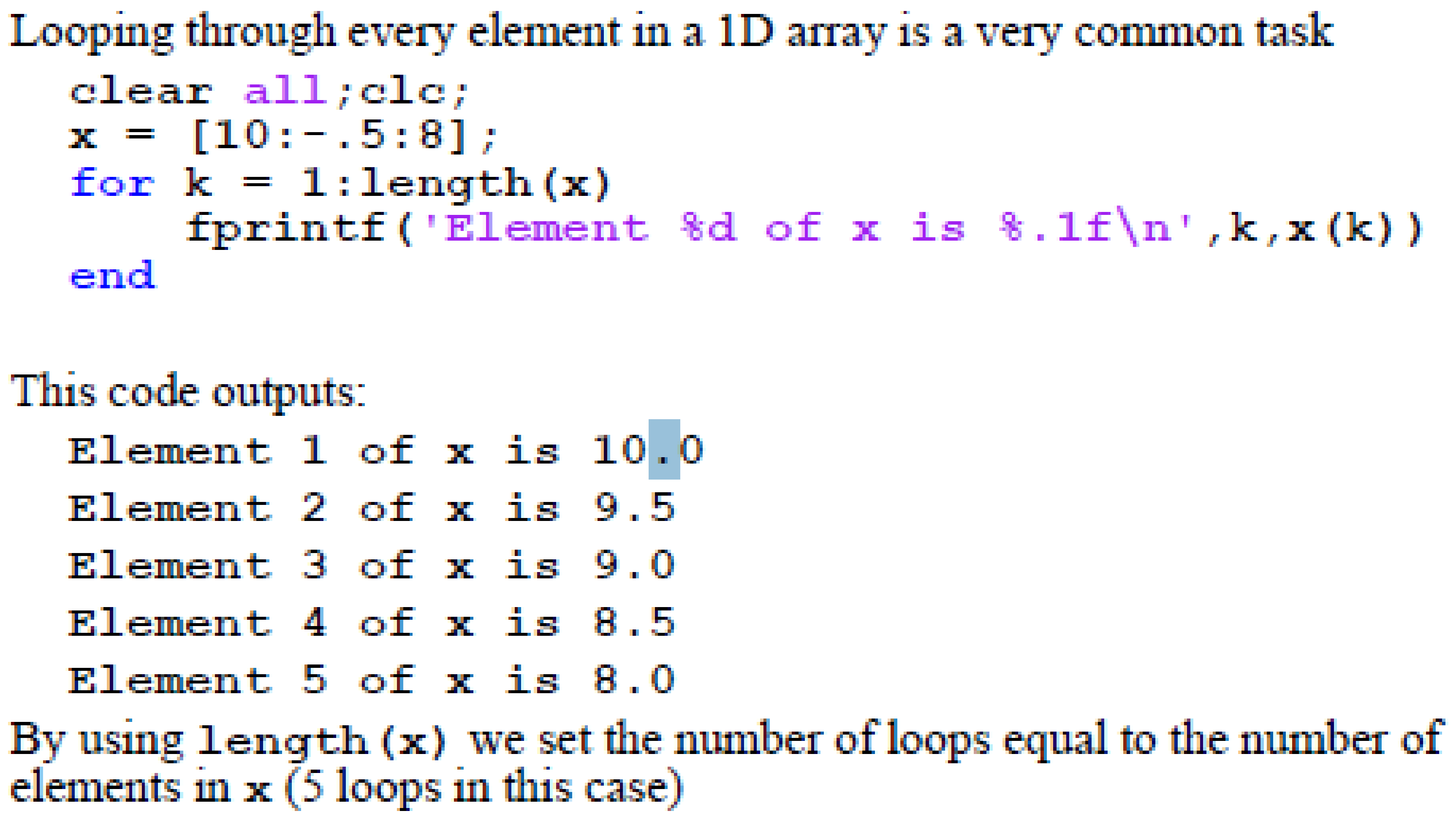
```
clear all;clc;
x = 0;  % initial x before loop starts
loopNum = 1;  % used to keep track of # of loops
for counter = 9:-1:7 % counts down this time
    disp(['During loop number ' num2str(loopNum)]);
    disp(['  The for-loop cntr variable is ' num2str(counter)]);
    x = x + counter; % add counter to x
    disp(['  "x = x + counter" changes x to ' num2str(x)]);
    loopNum = loopNum+1; % increment loop number
end
```

Running the code generates the following:



```
During loop number 1
  The for-loop cntr variable is 9
  "x = x + counter" changes x to 9
During loop number 2
  The for-loop cntr variable is 8
  "x = x + counter" changes x to 17
During loop number 3
  The for-loop cntr variable is 7
  "x = x + counter" changes x to 24
```

for loops and 1D arrays



Looping through every element in a 1D array is a very common task

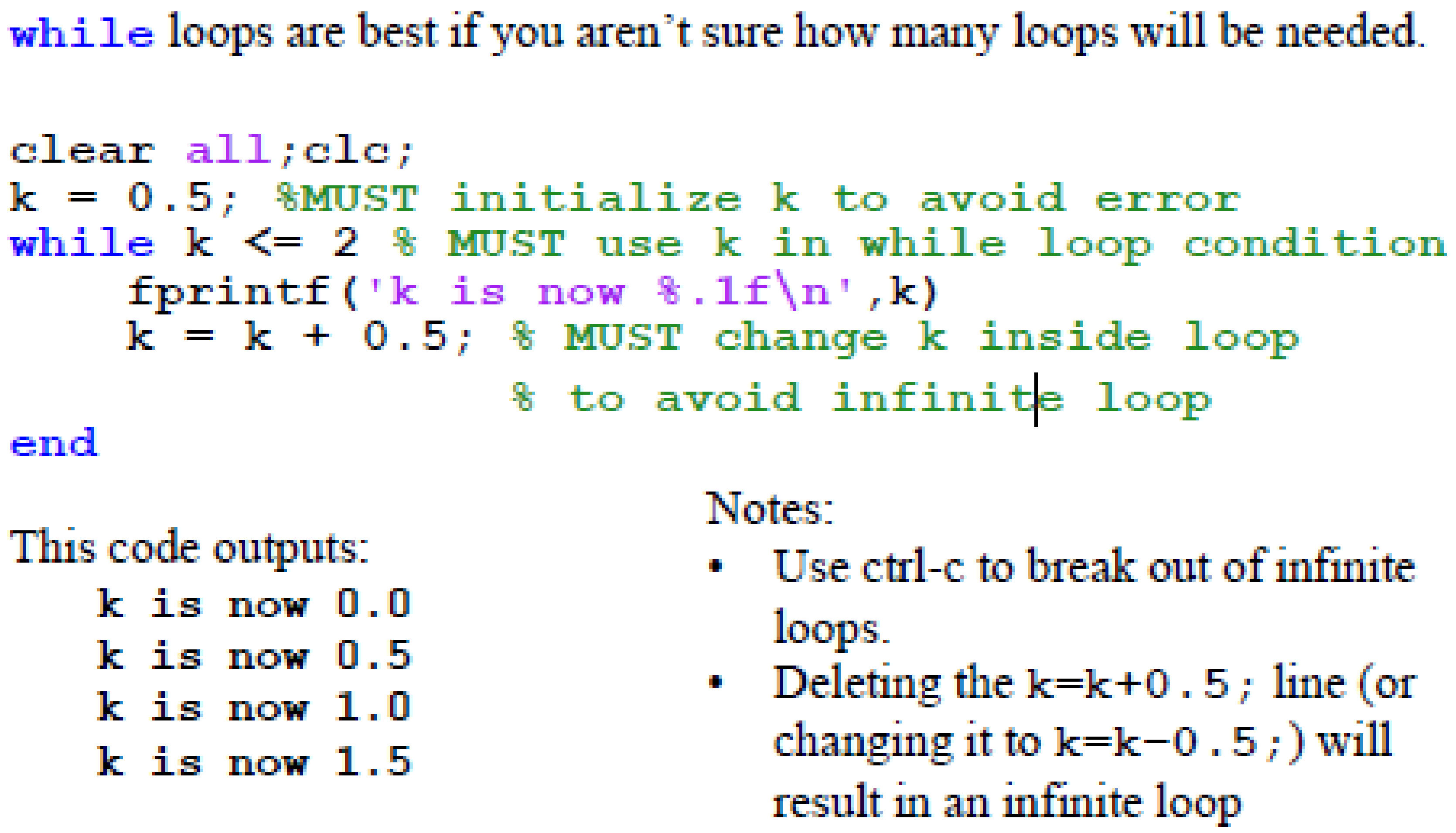
```
clear all;clc;
x = [10:-.5:8];
for k = 1:length(x)
    fprintf('Element %d of x is %.1f\n',k,x(k))
end
```

This code outputs:

```
Element 1 of x is 10.0
Element 2 of x is 9.5
Element 3 of x is 9.0
Element 4 of x is 8.5
Element 5 of x is 8.0
```

By using `length(x)` we set the number of loops equal to the number of elements in **x** (5 loops in this case)

Example while Statement



while loops are best if you aren't sure how many loops will be needed.

```
clear all;clc;
k = 0.5; %MUST initialize k to avoid error
while k <= 2 % MUST use k in while loop condition
    fprintf('k is now %.1f\n',k)
    k = k + 0.5; % MUST change k inside loop
                 % to avoid infinite loop
end
```

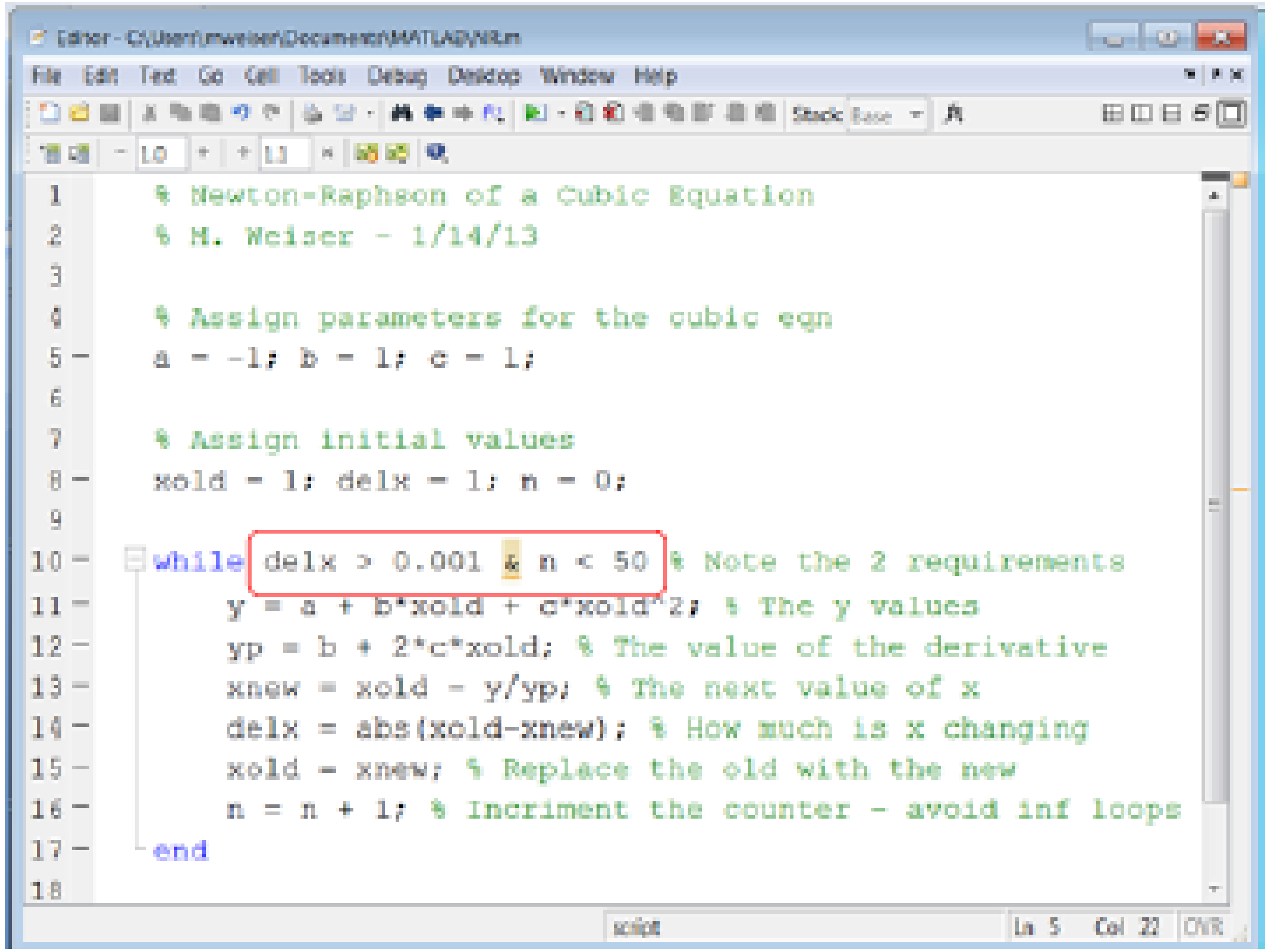
This code outputs:

```
   k is now 0.0
   k is now 0.5
   k is now 1.0
   k is now 1.5
```

Notes:

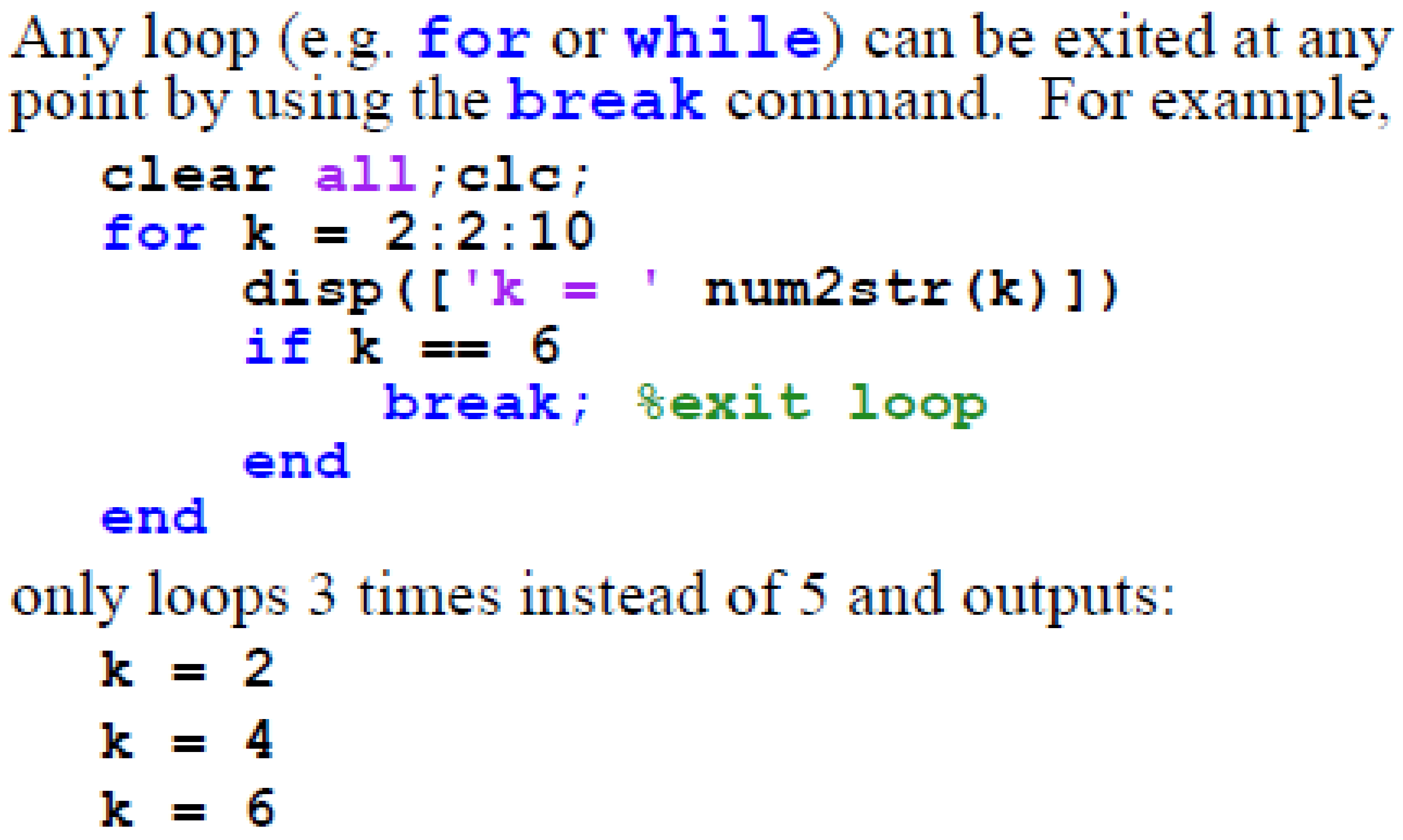
- Use ctrl-c to break out of infinite loops.
- Deleting the `k=k+0.5;` line (or changing it to `k=k−0.5;`) will result in an infinite loop

Can Use Logical Operators in a **while** Statement


Editor - C:\Users\mweiser\Documents\MATLAB\NR.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack Base
1.0
1.1
script
Ln 5 Col 22 OVR

```
% Newton-Raphson of a Cubic Equation
% M. Weiser - 1/14/13

% Assign parameters for the cubic eqn
a = -1; b = 1; c = 1;

% Assign initial values
xold = 1; delx = 1; n = 0;

while delx > 0.001 & n < 50 % Note the 2 requirements
    y = a + b*xold + c*xold^2; % The y values
    yp = b + 2*c*xold; % The value of the derivative
    xnew = xold - y/yp; % The next value of x
    delx = abs(xold-xnew); % How much is x changing
    xold = xnew; % Replace the old with the new
    n = n + 1; % Incriment the counter - avoid inf loops
end
```

Exiting Loops Early



Any loop (e.g. `for` or `while`) can be exited at any point by using the `break` command. For example,

```
clear all;clc;
for k = 2:2:10
    disp(['k = ' num2str(k)])
    if k == 6
        break; %exit loop
    end
end
```

only loops 3 times instead of 5 and outputs:

```
k = 2
k = 4
k = 6
```

MATLAB Errors and Debugging

- The process of finding and correcting errors is called debugging. It is a primary task in code design.
- Debugging can be very difficult and frustrating
- Three common types of errors
  - Syntax – incorrect spelling or use of arguments
  - Run-time – illegal operations
  - Logic – mistakes in math, branching, or looping
    - Can be very tough to find
- Two good debugging approaches for MATLAB are:
  - Breaking script into sections and running each section separately
  - Using MATLAB's debugging tools

Syntax Errors

- Errors in the MATLAB statement itself, such as spelling or punctuation errors. If found MATLAB won't even try to run your script.
- Examples:
  - `2=x` or `for=10` - Where's the error(s)?
    - **Error: The expression to the left of the equals sign is not a valid target for an assignment.**
  - `a=(4+3)/2)` - Where's the error(s)?
    - **Error: Unbalanced or unexpected parenthesis or bracket.**

Run-time Errors

- Illegal operations that cause a script/program to “crash” when trying to run
- Examples:
  - `x = [1 2 3;4]` - Where’s the error(s)?
    - **Error: Dimensions of matrices being concatenated are not consistent.** – How can you fix it?
    - Fix: `x = [1 2 3;4 0 0]` or similar
  - `sni(pi)` - Where’s the error(s)?
    - **Error: Undefined function 'sni' for input arguments of type 'double'.** – How can you fix it?
    - Fix: `sin(pi)`

Example Run-time Error

- Running the following generates a run-time error. Can you spot it?

```
x = 4:10;
for k = 0:length(x)
    v = v + x(k);
end
```

  - Error 1: `Undefined function or variable 'v'.` Fix?
    - Fix by initializing v before `for` loop: `v = 0`
  - Error 2: `Attempted to access x(0); index must be a positive integer or logical.` Fix?
    - Fix by changing 0 to 1: `k = 1:length(x)`

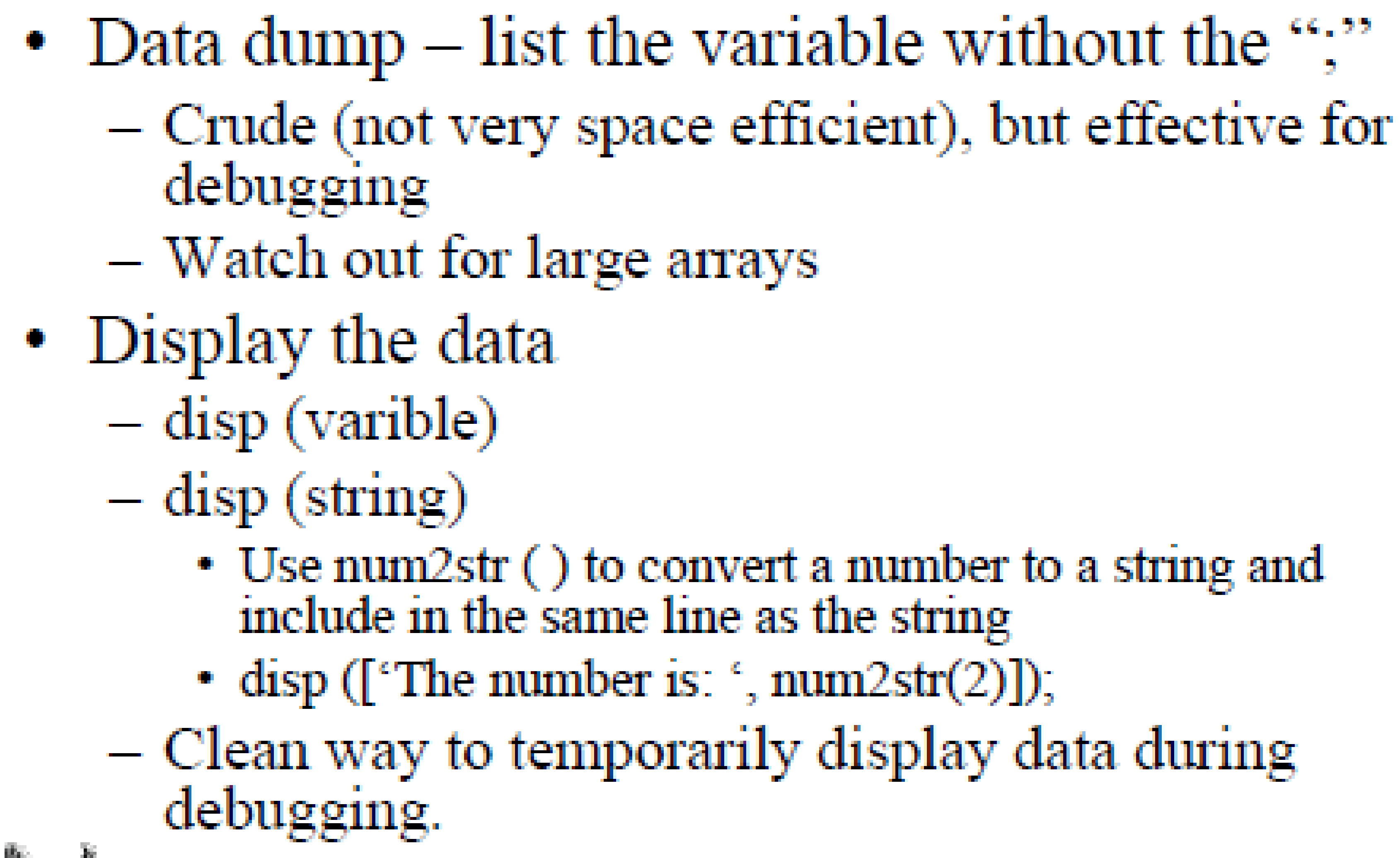
Logical Errors

- Occur when the program runs without displaying an error, but produces an unexpected result.
- Example:

```
side = input('What is the side of your
square?  ')
area = side + side;
fprintf('The area is %.2f.\n', area)
```

  - Can you find the error?
  - What to you expect the result to be if the user enters 5?
  - What will be the result if the user enters 5?
  - How do you fix the error?
    - `area = side*side;`
  - Note: If you enter 2 instead, you will see that it could be difficult to find logical errors.

Debugging using the Command Window

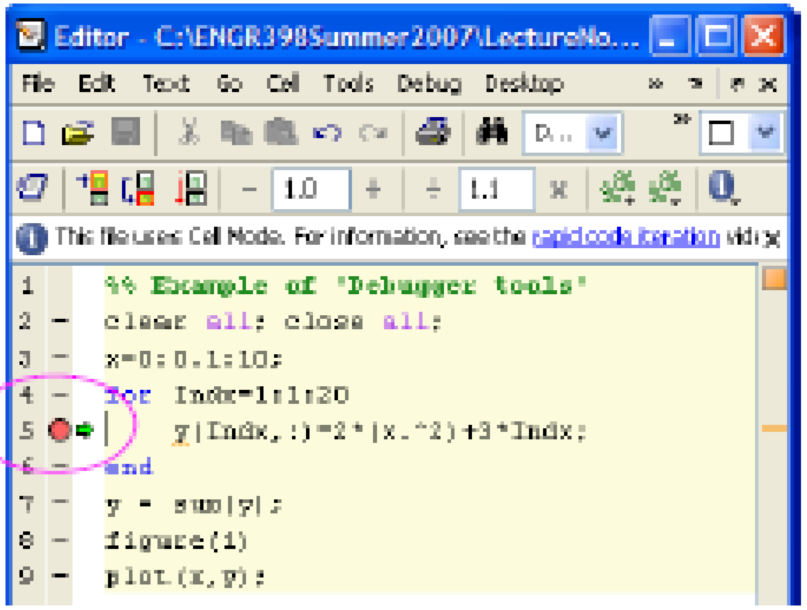


- Data dump – list the variable without the “;”
  - Crude (not very space efficient), but effective for debugging
  - Watch out for large arrays
- Display the data
  - disp (varible)
  - disp (string)
    - Use num2str ( ) to convert a number to a string and include in the same line as the string
    - disp ([‘The number is: ‘, num2str(2)]);
  - Clean way to temporarily display data during debugging.

MATLAB Debug Tools

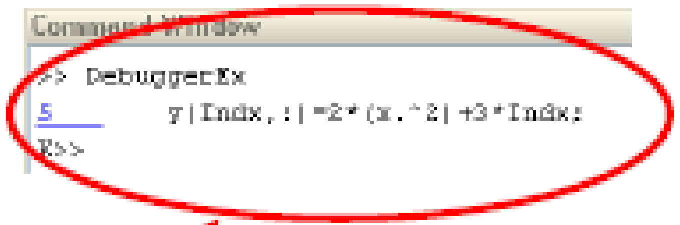
- There is a debugger built into the MATLAB editor
- To enter debug mode, set one or more "break points" by clicking the "–" marks beside a line numbers in the MATLAB editor.
- When the script is run, it will pause at the first "break point".
- When the script is paused, you can use the command window to investigate current variables.
- You can then use "step" to continue executing the script one line at a time or select other options as detailed on the next slides.

Example of Debugging Tools


Editor - C:\ENGR398Summer2007\LectureNo...
File Edit Text Go Cell Tools Debug Desktop
1.0
1.1
This file uses Cell Mode. For information, see the rapid code iteration video

```
%% Example of 'Debugger tools'
clear all; close all;
x=0:0.1:10;
for Indx=1:1:20
    y(Indx,:)=2*(x.^2)+3*Indx;
end
y = sum(y);
figure(1)
plot(x,y);
```

A break point is set. You can set more than one if you want.

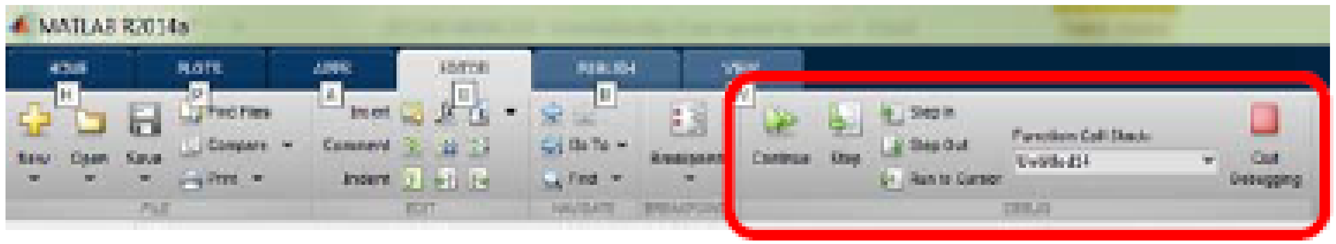


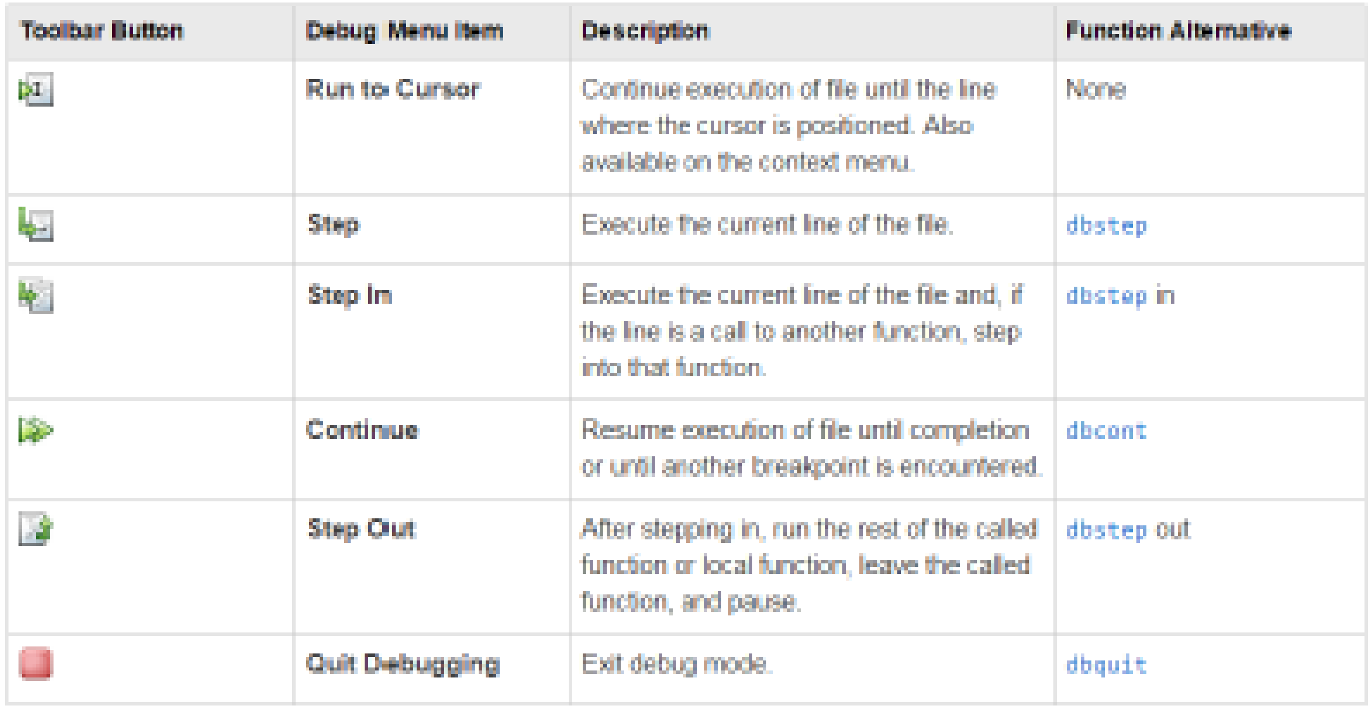
```
>> DebuggerEx
5       y(Indx,:)=2*(x.^2)+3*Indx;
K>>
```

Note:

1. Program stopped at the break point.
2. You can check any variables, plot data, etc. at the point program stopped.

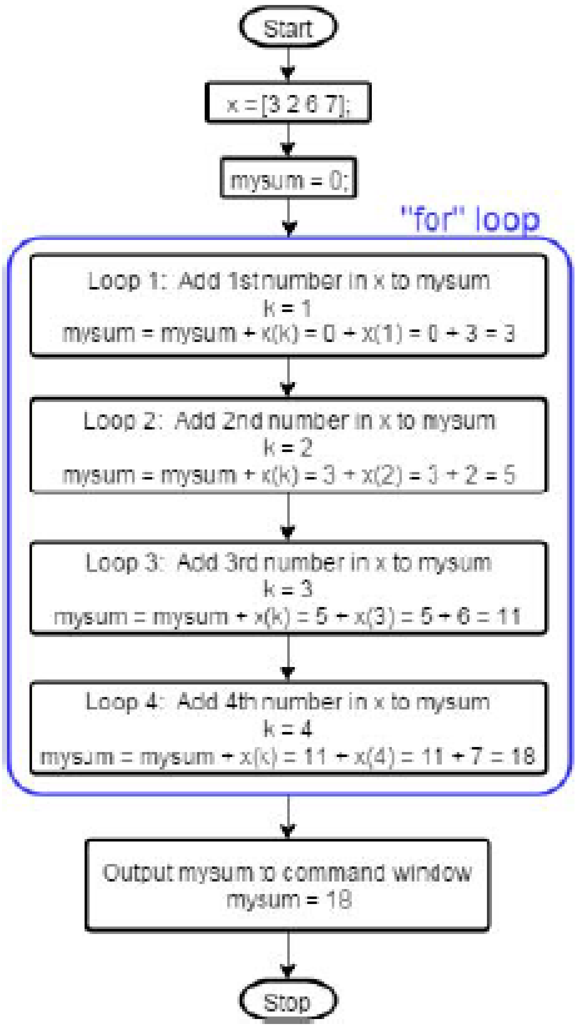
Debugging Mode Menu



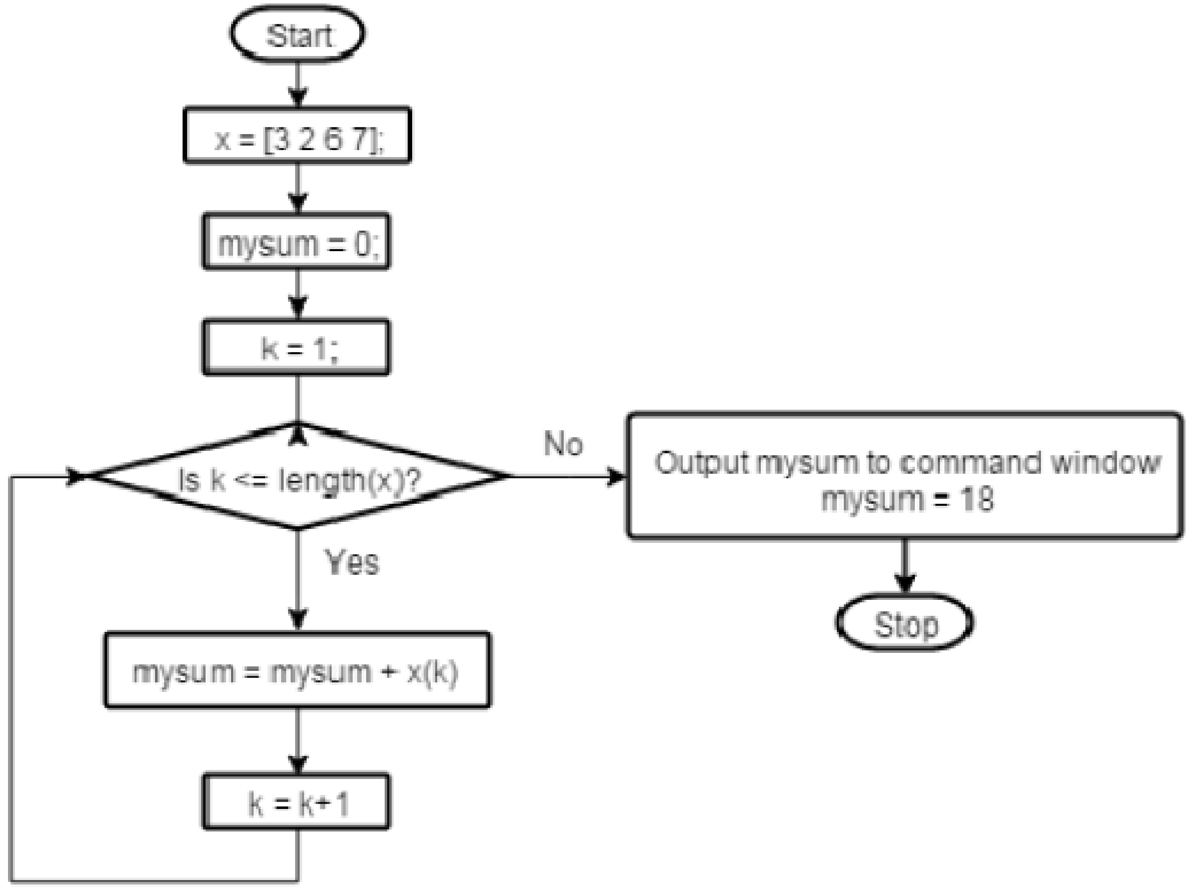


Toolbar Button	Debug Menu Item	Description	Function Alternative
⇥	**Run to Cursor**	Continue execution of file until the line where the cursor is positioned. Also available on the context menu.	None
↓	**Step**	Execute the current line of the file.	`dbstep`
↘	**Step In**	Execute the current line of the file and, if the line is a call to another function, step into that function.	`dbstep` in
⏩	**Continue**	Resume execution of file until completion or until another breakpoint is encountered.	`dbcont`
↗	**Step Out**	After stepping in, run the rest of the called function or local function, leave the called function, and pause.	`dbstep` out
🟥	**Quit Debugging**	Exit debug mode.	`dbquit`

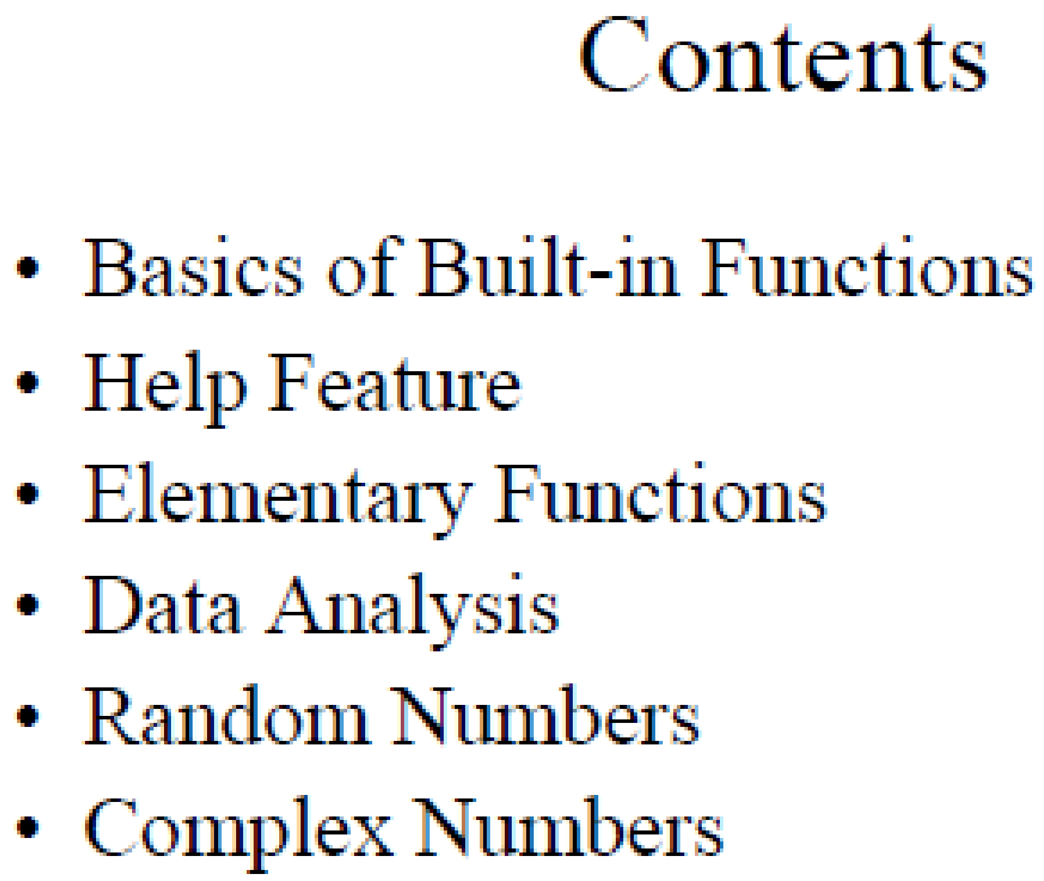
for Loop Sum Flowchart


Start
x = [3 2 6 7];
mysum = 0;
"for" loop
Loop 1: Add 1st number in x to mysum
k = 1
mysum = mysum + x(k) = 0 + x(1) = 0 + 3 = 3
Loop 2: Add 2nd number in x to mysum
k = 2
mysum = mysum + x(k) = 3 + x(2) = 3 + 2 = 5
Loop 3: Add 3rd number in x to mysum
k = 3
mysum = mysum + x(k) = 5 + x(3) = 5 + 6 = 11
Loop 4: Add 4th number in x to mysum
k = 4
mysum = mysum + x(k) = 11 + x(4) = 11 + 7 = 18
Output mysum to command window
mysum = 18
Stop

while Loop Sum Flowcharts



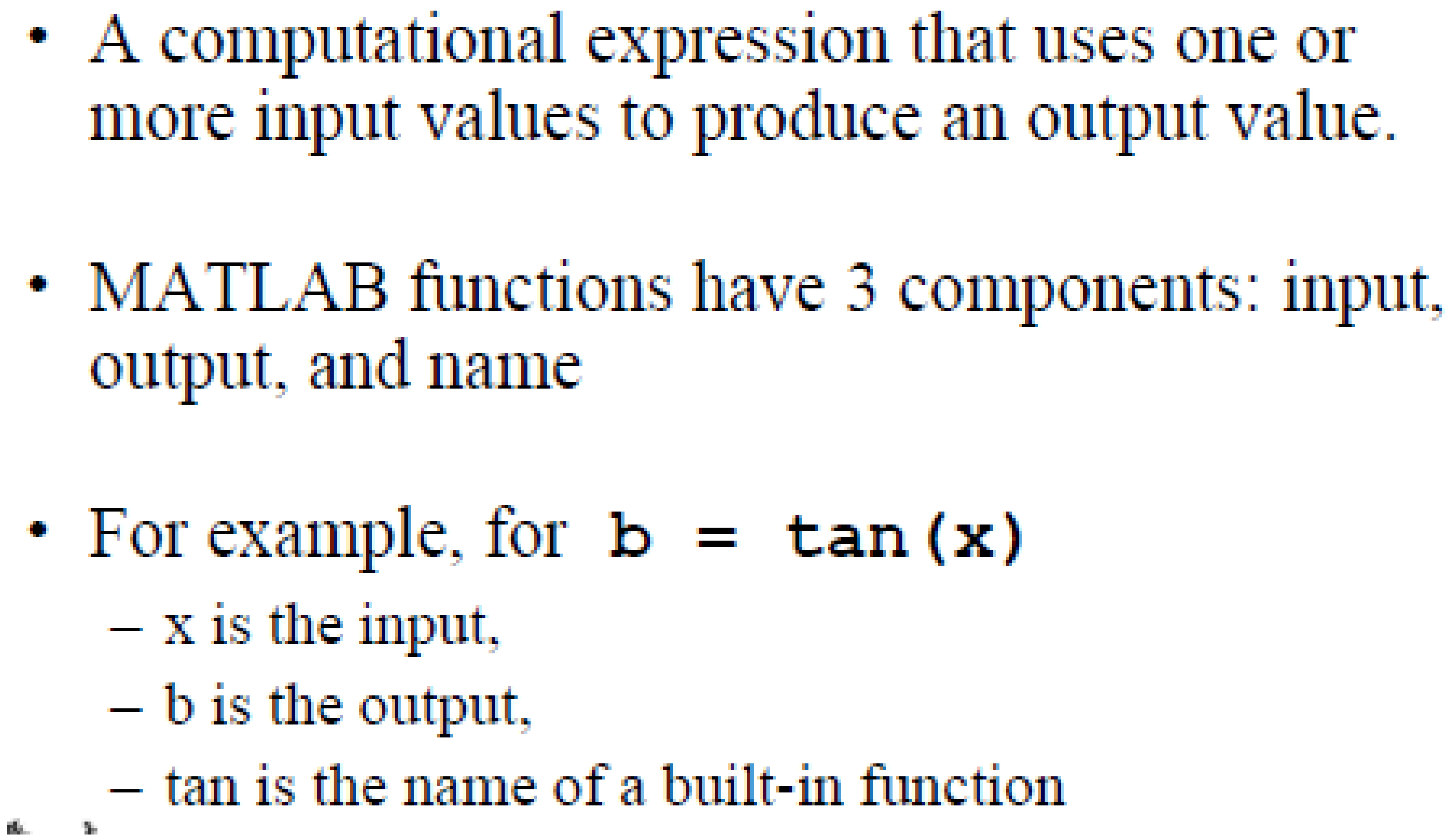
Multiple Useful Functions



Contents

- Basics of Built-in Functions
- Help Feature
- Elementary Functions
- Data Analysis
- Random Numbers
- Complex Numbers

What is a Built-In Function?



- A computational expression that uses one or more input values to produce an output value.
- MATLAB functions have 3 components: input, output, and name
- For example, for **`b = tan(x)`**
  - x is the input,
  - b is the output,
  - tan is the name of a built-in function

MATLAB Functions

- Functions take the form:
  variable = function(number or variable)
- MATLAB has many functions stored in its file system.
- To use one of the built-in functions you need to know its name and what the input values are.
- For example, the square root function: `sqrt( )`.
- Find the square root of 9 using MATLAB

```
 >> a = sqrt(9)
    a = 3
```

HELP Feature

- You may not always recall the name and syntax of a function you want to use since MATLAB has so many built-in functions.
- MATLAB has an indexed library of its built-in functions that you can access through the help feature.
- If you type help in the command window MATLAB provides a list of help topics.

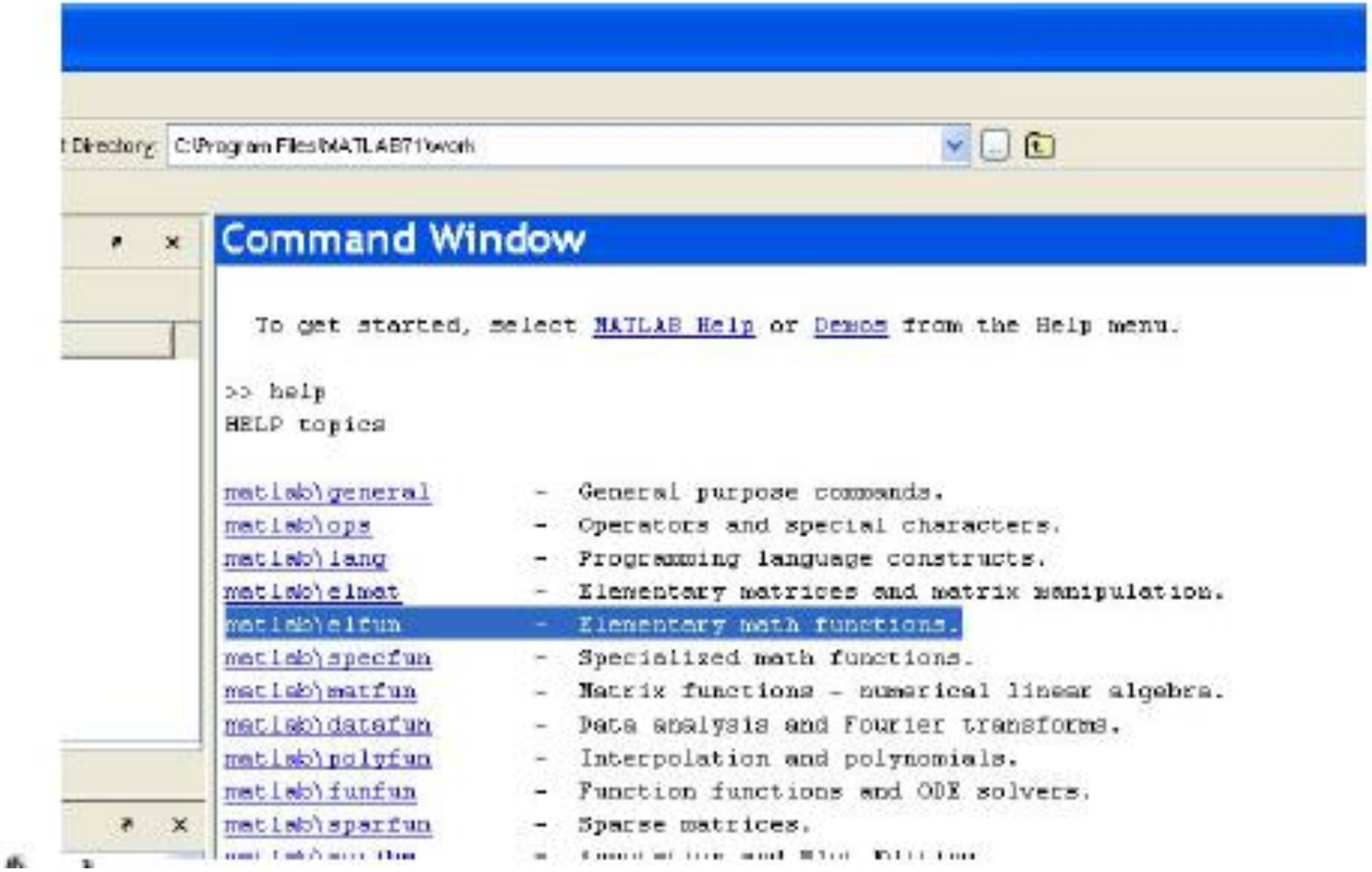
Help

- In MATLAB command window type

```
>> help
```

- If we are interested in the elementary mathematics functions, we find it on the list of help topics (5th down) and click it.
- A list of commands appears with the description of what each one does.
- Try a few!

MATLAB Help



```
 To get started, select MATLAB Help or Demos from the Help menu.

>> help
HELP topics

matlab\general      -  General purpose commands.
matlab\ops          -  Operators and special characters.
matlab\lang         -  Programming language constructs.
matlab\elmat        -  Elementary matrices and matrix manipulation.
matlab\elfun        -  Elementary math functions.
matlab\specfun      -  Specialized math functions.
matlab\matfun       -  Matrix functions - numerical linear algebra.
matlab\datafun      -  Data analysis and Fourier transforms.
matlab\polyfun      -  Interpolation and polynomials.
matlab\funfun       -  Function functions and ODE solvers.
matlab\sparfun      -  Sparse matrices.
```

More Help

- For more specific help: **help topic**
- Check it out:

```
>> help tan
```

- MATLAB describes what the function **tan** does and provides helpful links to similar or related functions to assist you.

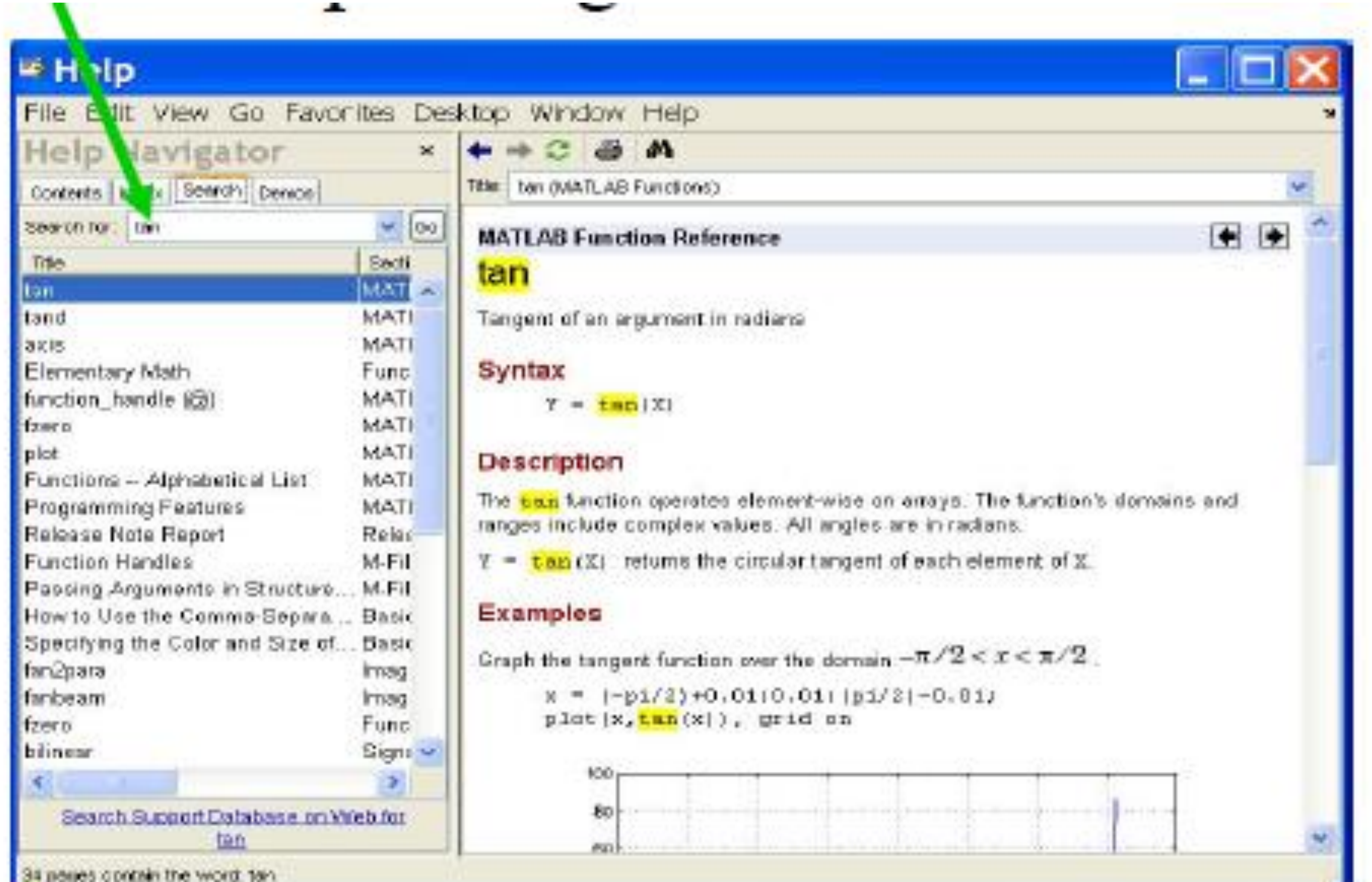
Hands-On

- Use MATLAB help to find the exponential, natural logarithm, and logarithm base 2 functions
- Calculate e^7 `>> exp(7)`
- Calculate $\ln(4)$ `>> log(4)`
- Calculate $\log_2(12)$ `>> log2(12)`

Help Navigator

- Click on Help on the tool bar at the top of MATLAB, and select MATLAB Help.
- A HELP window will pop up.
- Under Help Navigator on the left of screen select the Search tab.
- You can Search for specific help topics by typing your topic in the space after the :.
- You can also press F1 on your keyboard to access the windowed help.

Help Navigator window


Help
File Edit View Go Favorites Desktop Window Help
Help Navigator
Contents
Index
Search
Demos
Search for: tan
Go
Title
Secti
Search Support Database on Web for tan
34 pages contain the word: tan
Title: tan (MATLAB Functions)

Title	Secti
tan	MATI
tand	MATI
axis	MATI
Elementary Math	Func
function_handle (@)	MATI
fzero	MATI
plot	MATI
Functions -- Alphabetical List	MATI
Programming Features	MATI
Release Note Report	Relea
Function Handles	M-Fil
Passing Arguments in Structure...	M-Fil
How to Use the Comma-Separa...	Basic
Specifying the Color and Size of...	Basic
fan2para	Imag
fanbeam	Imag
fzero	Func
bilinear	Signa

MATLAB Function Reference

tan

Tangent of an argument in radians

Syntax

```
Y = tan(X)
```

Description

The tan function operates element-wise on arrays. The function's domains and ranges include complex values. All angles are in radians.

Y = tan(X) returns the circular tangent of each element of X.

Examples

Graph the tangent function over the domain $-\pi/2 < x < \pi/2$.

```
x = (-pi/2)+0.01:0.01:(pi/2)-0.01;
plot(x,tan(x)), grid on
```

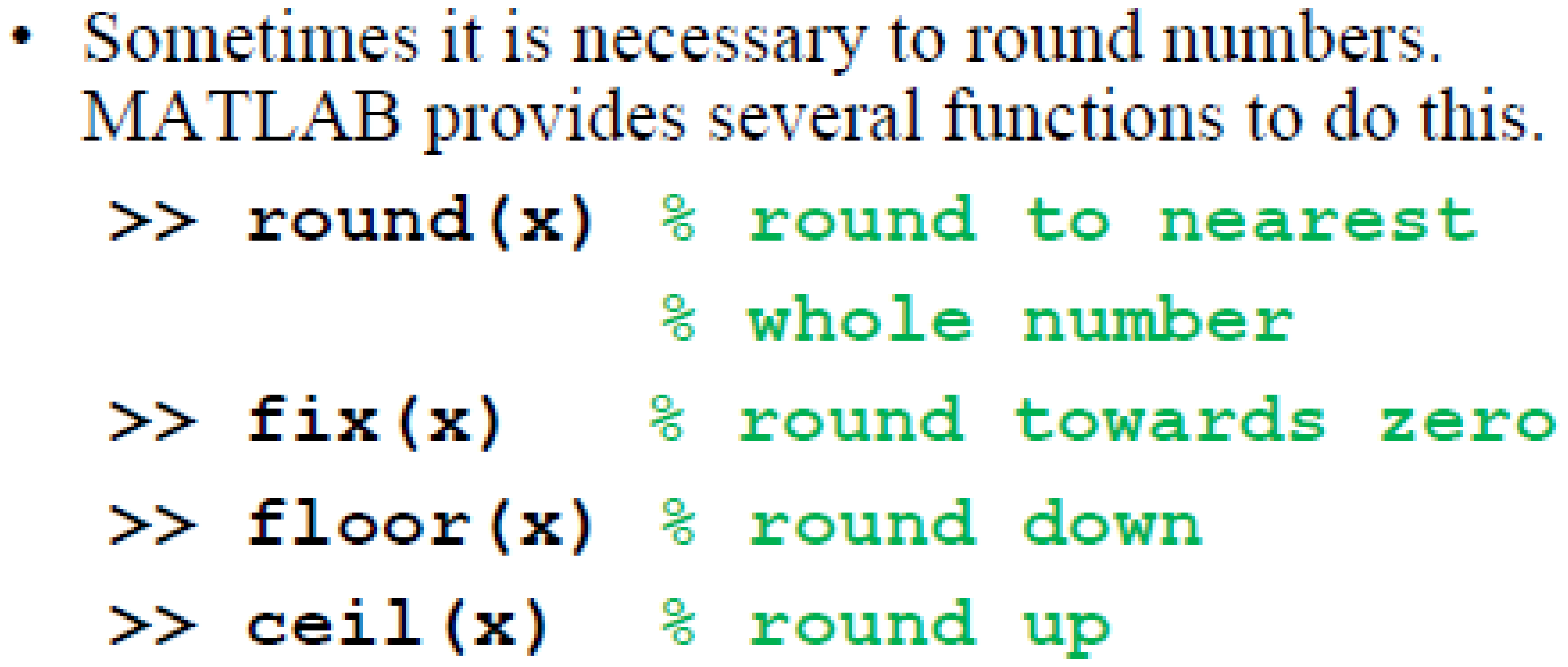
Elementary Mathematical Functions

- MATLAB can perform all of the operations that your calculator can and more.
- Search for the topic Elementary math in the Help Navigator just described.
- Try the following in MATLAB to continue your exploration of MATLAB capabilities

```
    >> sqrt(625)                         ans = 25
    >> log(7)                            ans = 1.9459
```

Of course, try a few others.

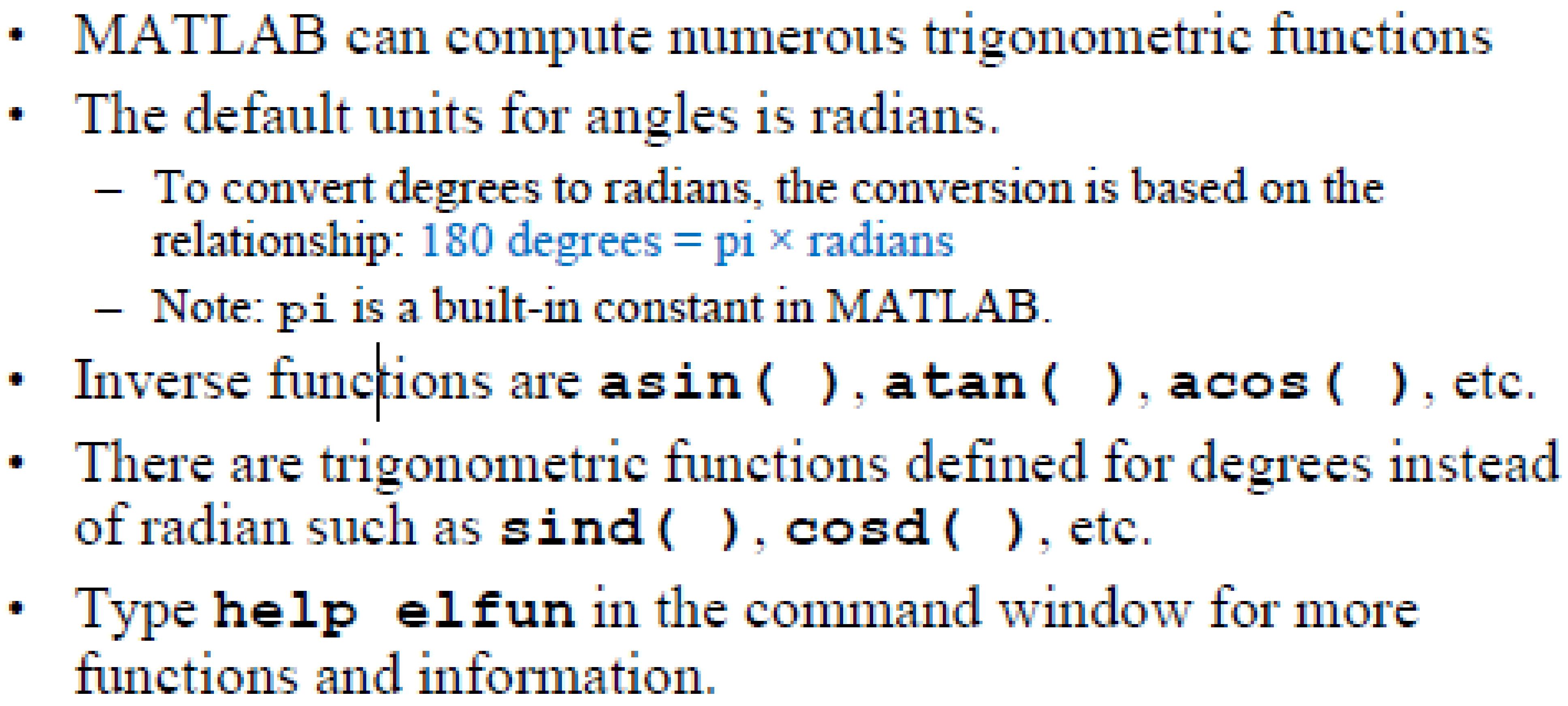
Rounding Functions



- Sometimes it is necessary to round numbers. MATLAB provides several functions to do this.

```
>> round(x)  % round to nearest
             % whole number
>> fix(x)    % round towards zero
>> floor(x)  % round down
>> ceil(x)   % round up
```

Trigonometric Functions



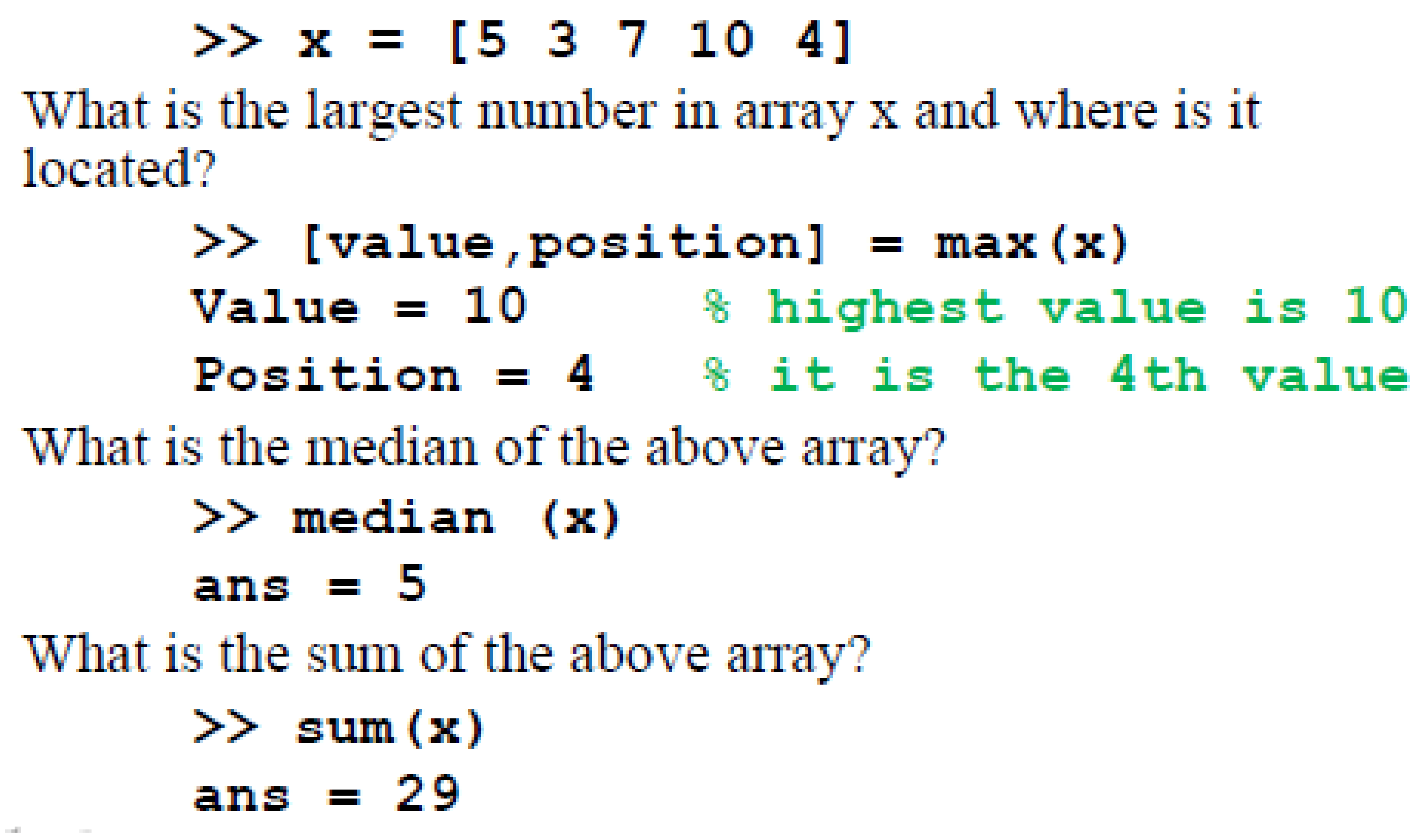
- MATLAB can compute numerous trigonometric functions
- The default units for angles is radians.
  - To convert degrees to radians, the conversion is based on the relationship: 180 degrees = pi × radians
  - Note: `pi` is a built-in constant in MATLAB.
- Inverse functions are **`asin( )`**, **`atan( )`**, **`acos( )`**, etc.
- There are trigonometric functions defined for degrees instead of radian such as **`sind( )`**, **`cosd( )`**, etc.
- Type **`help elfun`** in the command window for more functions and information.

Data Analysis Functions

- It is often necessary to analyze data statistically.
- MATLAB has many statistical functions:

```
max( )        sum( )          size( )
min( )        prod( )         length( )
mean( )       sort( )         std( )
median( )     sortrows( )     var( )
              find( )
```

Data Analysis Practice



```
>> x = [5 3 7 10 4]
```

What is the largest number in array x and where is it located?

```
>> [value,position] = max(x)
Value = 10        % highest value is 10
Position = 4      % it is the 4th value
```

What is the median of the above array?

```
>> median (x)
ans = 5
```

What is the sum of the above array?

```
>> sum(x)
ans = 29
```

Hands-On

```
>> v = [2 24 53 7 84 9]
>> y = [2 4 56; 3 6 88]
```

Sort v in descending order

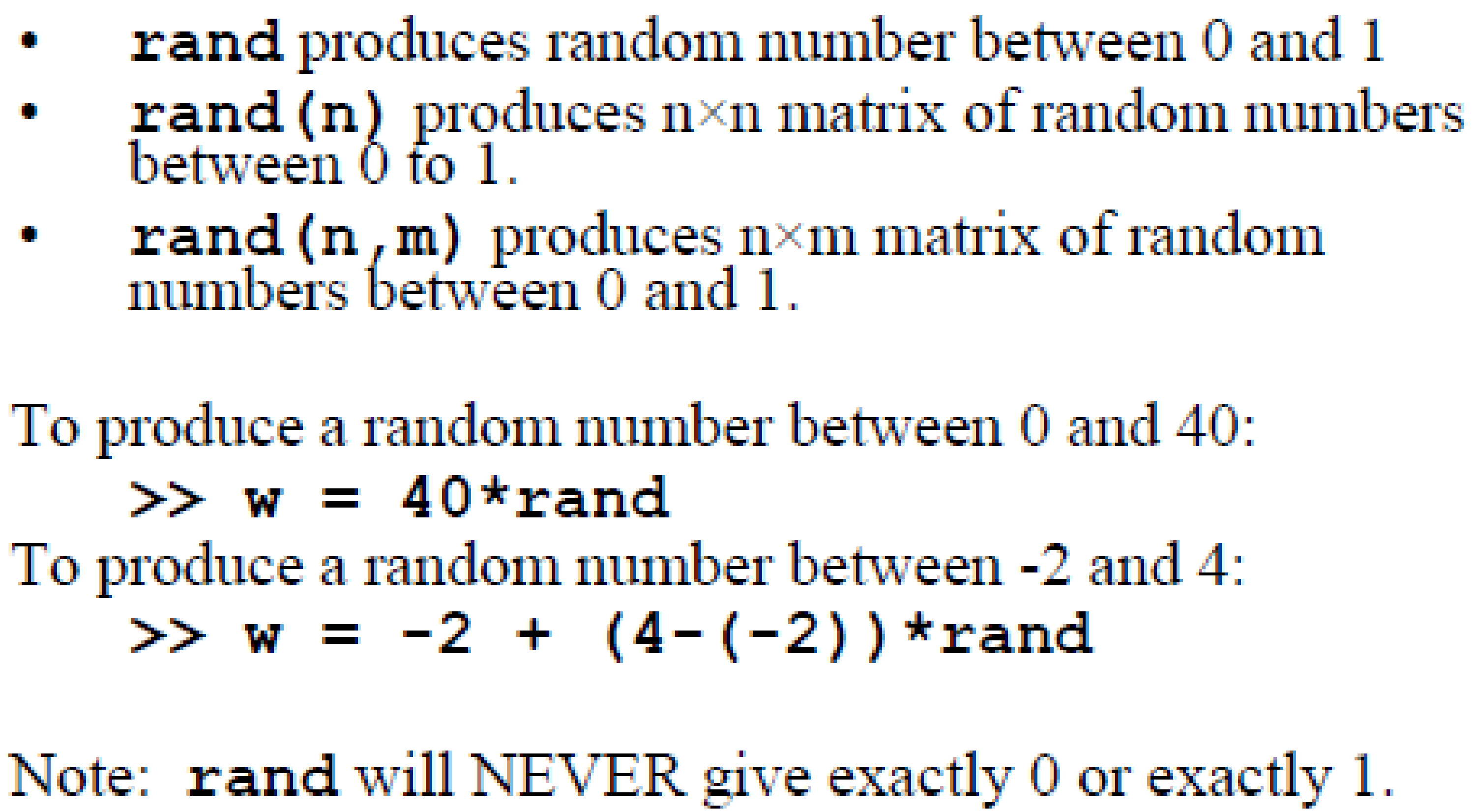
Find the size of y

Find the standard deviation of v

Find the cumulative product of v

Sort the rows of y based on the 3rd column.

Generation of Random Numbers



- **`rand`** produces random number between 0 and 1
- **`rand(n)`** produces n×n matrix of random numbers between 0 to 1.
- **`rand(n,m)`** produces n×m matrix of random numbers between 0 and 1.

To produce a random number between 0 and 40:

```
    >> w = 40*rand
```

To produce a random number between -2 and 4:

```
    >> w = -2 + (4-(-2))*rand
```

Note: **`rand`** will NEVER give exactly 0 or exactly 1.

Complex Numbers

- Complex numbers take the form of a+b*i:
  - a is the real part
  - b is the imaginary part
  - and i = $\sqrt{-1}$
- Complex numbers can be created as follows:

```
>> a = 2; b = 3;
>> c = a+b*i           %method 1
>> c = complex(a,b)   %method 2

   c = 2.0000 + 3.0000i
```

- Note: both **`i`** and **`j`** are built-in MATLAB constants that equal $\sqrt{-1}$

Complex Numbers Continued

- Function to extract the real and imaginary components of a complex number:

```
real(c)
imag(c)
```

- Function to find the absolute value or modulus of a complex number:

```
abs(c) % = sqrt(real(c)^2 + imag(c)^2)
```

- Function to find the angle or argument (in radians) of a complex number:

```
angle(c) % = atan(imag(c)/real(c))
```

Other Useful Functions

- `clock %Outputs 1x6 array containing year, month, day, hour, min, sec.`
- `date      %Outputs date as string`
- `tic       %Start timer`
- `toc       %Output time elapsed`
- `pause(XX) %pause for XX seconds`

Exercises

- Find the modulus (magnitude, abs) and angle (argument) of the complex number 3+4i.
- Generate a 4x4 array of random numbers between 0 and 10. Sort each column of the array.
- Use MATLAB's help function to find built-in functions to determine:
  - The base 10 logarithm of 5
  - The secant of pi

ICE 1

Write a function called **circle** that takes a scalar input r. It needs to return an output called area that is the area of a circle with radius r and a second output, cf that is the circumference of the same circle. You are allowed to use the built-in function pi.

ICE 2

Write a function called even_index that takes a matrix, M, as input argument and returns a matrix that contains only those elements of M that are in even rows and columns. In other words, it would return the elements of M at indices (2,2), (2,4), (2,6), …, (4,2), (4,4), (4,6), …, etc. Note that both the row and the column of an element must be even to be included in the output. The following would not be returned: (1,1), (2,1), (1,2) because either the row or the column or both are odd. As an example, if M were a 5-by-8 matrix, then the output must be 2-by-4 because the function omits rows 1, 3 and 5 of M and it also omits columns 1, 3, 5, and 7 of M.

ICE 3

Write a function called flip_it that has one input argument, a row vector v, and one output argument, a row vector w that is of the same length as v. The vector w contains all the elements of v, but in the exact opposite order. For example, is v is equal to [1 2 3] then w must be equal to [3 2 1]. You are not allowed to use the built-in function flip.

ICE 4

Write a function called top_right that takes two inputs: a matrix N and a scalar
non-negative
integer n, in that order, where each dimension of N is greater than or equal to n.
The function returns the n-by-n square subarray of N located at the top right corner of N.

ICE 5

Write a function called peri_sum that computes the sum of the elements of an input matrix A that are on the “perimeter” of A. In other words, it adds together the elements that are in the first and last rows and columns. Note that the smallest dimension of A is at least 2, but you do not need to check this. Hint: do not double count any elements!

ICE 6

Write a function called light_speed that takes as input a row vector of distances in kilometers and returns two row vectors of the same length. Each element of the first output argument is the time in minutes that light would take to travel the distance specified by the corresponding element of the input vector. To check your math, it takes a little more than 8 minutes for sunlight to reach Earth which is 150 million kilometers away. The second output contains the input distances converted to miles. Assume that the speed of light is 300,000 km/s and that one mile equals 1.609 km.

ICE 7

Write a function that is called like this: amag = accelerate(F1,F2,m). F1 and F2 are three-element column vectors that represent two forces applied to a single object. The argument m equals the mass of the object in units of kilograms. The three elements of each force equal the x, y,and z components of the force in Newtons. The output variable amag is a scalar that is equal to the magnitude of the object's acceleration. The function calculates the object's acceleration vector a by using Newton's law: F = ma, where F is the sum of F1 and F2. Then it returns the magnitude of a.

Hint: we are talking about physical vectors here, so the Pythagorean theorem will come in handy.

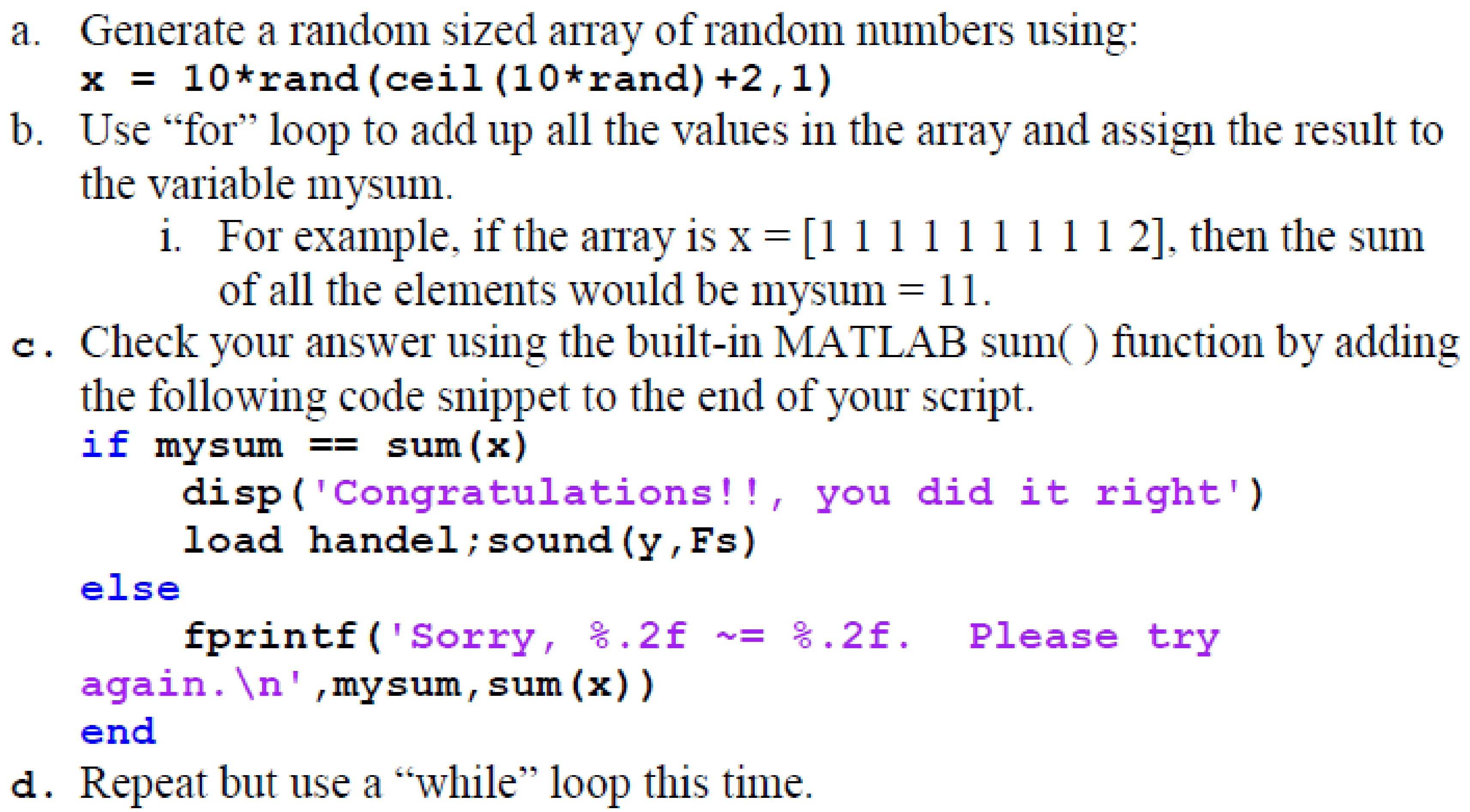
ICE 8

Write a function called income that takes two row vectors of the same length as input arguments. The first vector, rate contains the number of various products a company manufactures per hour simultaneously. The second vector, price includes the corresponding per item price they sell the given product for. The function must return the overall income the company generates in a week assuming a 6-day work week and two 8-hour long shifts per day.

ICE 9

Write a function called **circle** that takes a scalar input r. It needs to return an output called area that is the area of a circle with radius r and a second output, cf that is the circumference of the same circle. You are allowed to use the built-in function pi.

Problem 1



a. Generate a random sized array of random numbers using:
```
x = 10*rand(ceil(10*rand)+2,1)
```
b. Use "for" loop to add up all the values in the array and assign the result to the variable mysum.
   i. For example, if the array is x = [1 1 1 1 1 1 1 1 1 2], then the sum of all the elements would be mysum = 11.

c. Check your answer using the built-in MATLAB sum( ) function by adding the following code snippet to the end of your script.
```
if mysum == sum(x)
    disp('Congratulations!!, you did it right')
    load handel;sound(y,Fs)
else
    fprintf('Sorry, %.2f ~= %.2f.  Please try
again.\n',mysum,sum(x))
end
```
d. Repeat but use a "while" loop this time.

Problem 2

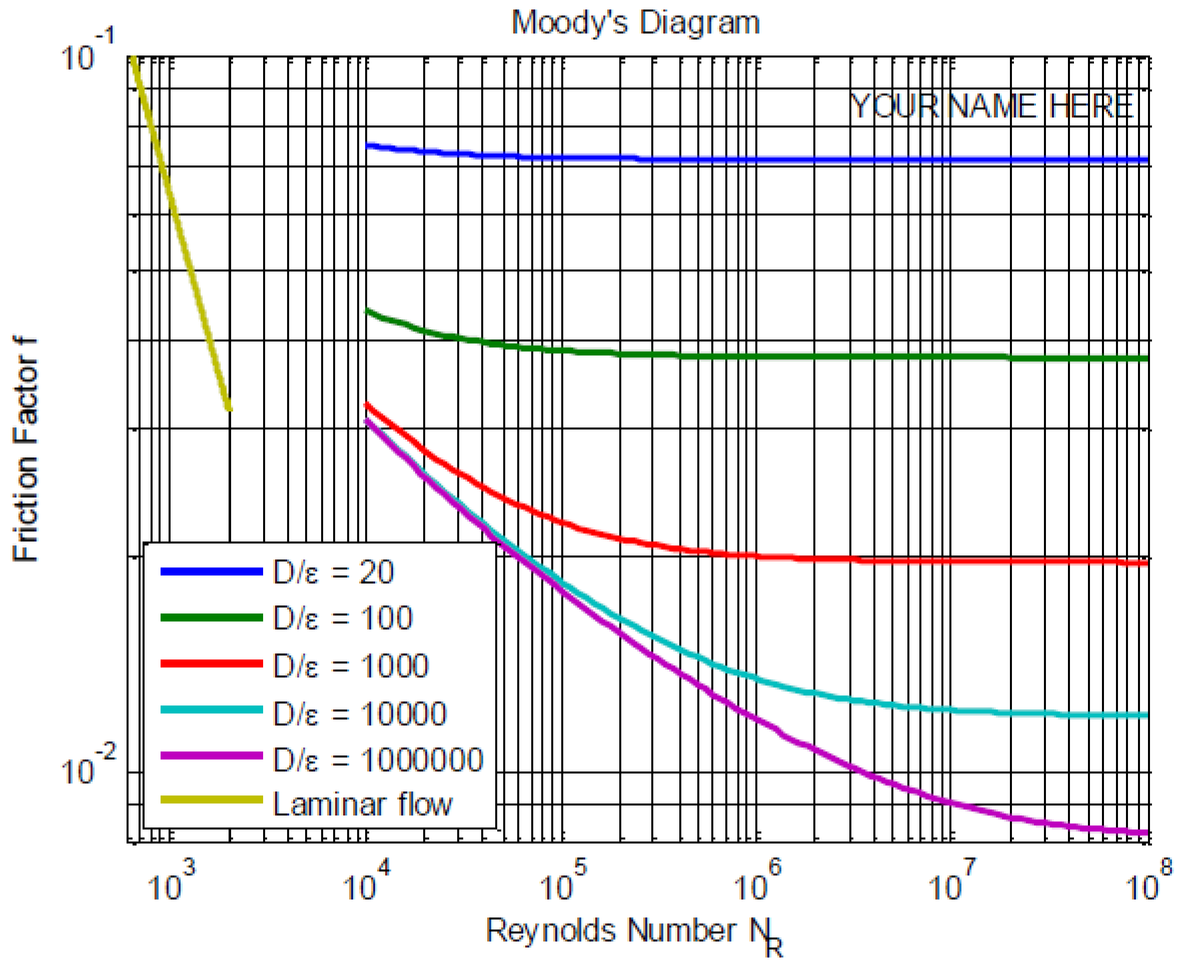
1. Create an array of the required *D/ε* values of 20, 100, 1000, 10,000, and 100,000.
2. Create a loop that:
   a. Generates a set of *f* values for each *D/ε* value (when done correctly, you will only have a single line in your code to calculate *f* instead of repeating the same equation 5 time like in your previous HW).

$$f = \frac{0.25}{\left[\log\left(\frac{1}{3.7(D/\varepsilon)} + \frac{5.74}{N_R^{0.9}}\right)\right]^2}$$

   b. Stores the newly calculated *f* values as a new row in a variable
   c. When done, your array should have 5 rows where each row corresponds do a different D/ε value. For example:
      i. loop 1:
```
f = [0.0750  0.0748  0.0745   ...]
```
      ii. loop 2:
```
f = [0.0750  0.0748  0.0745   ...;
     0.0440  0.0436  0.0432   ...]
```
      iii. loop 5:
```
f = [0.0750  0.0748  0.0745   ...;
     0.0440  0.0436  0.0432   ...;
     0.0327  0.0320  0.0313   ...;
     0.0311  0.0304  0.0296   ...;
     0.0310  0.0302  0.0295   ...]
```

3. Plot the data generated so that it matches the plot below (including labels, legend, your name, axes range, etc).
4. Generate and plot the laminar line making sure you match the N_R range shown (doesn't need to be done inside your loop).

Notes: Your script should work even if you change the number of *D/ε* values you plot. For example, if you change the number of *D/ε* values to 6 or even 100 your script/code will still work without having to modify the plot, or any other, command(s). To test this, add a couple of *D/ε* values to your array and see how your script responds.



Problem 3

On January 1st, Cindy opens a savings account and deposits \$10,000. At the end of every month, she deposits \$1000 more into the account for the next 12 month (starting January 31). At the end of each month (before the \$1000 deposit), interest is calculated ansd added to her balance. The monthly interest rate varies depending on the account balance at the time interest is calculated.

Balance (\$)	Interest
$B \le 15{,}000$	1%
$15{,}000 < B \le 20{,}000$	1.5%
$B > 20{,}000$	2%

Write a script that displays in the command window, for each of the 12 months, **with informative headers**, the:

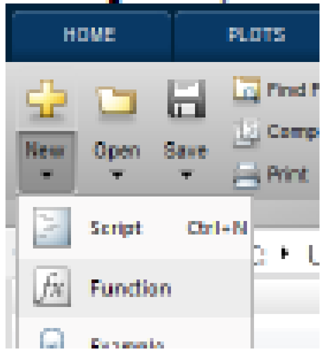
1. number of the current month
2. interest rate for current month as a percentage (e.g. 1.5 and not 0.015)
3. total amount of interest earned that month (with two decimal places)
4. new balance (with two decimal places)
5. total interest earned (running total of the cumulative interest earned from the opening deposit)

Your last row should be:
```
12   2.0%   $470.29   $24984.92   $2984.92
```

Problem 4

For this assignment, you will create a user defined function that places your name in the corner of any plot.

1. Create a new function file (as discussed, this file is similar but the not the same as a script file). Give your function a descriptive name of your choice.



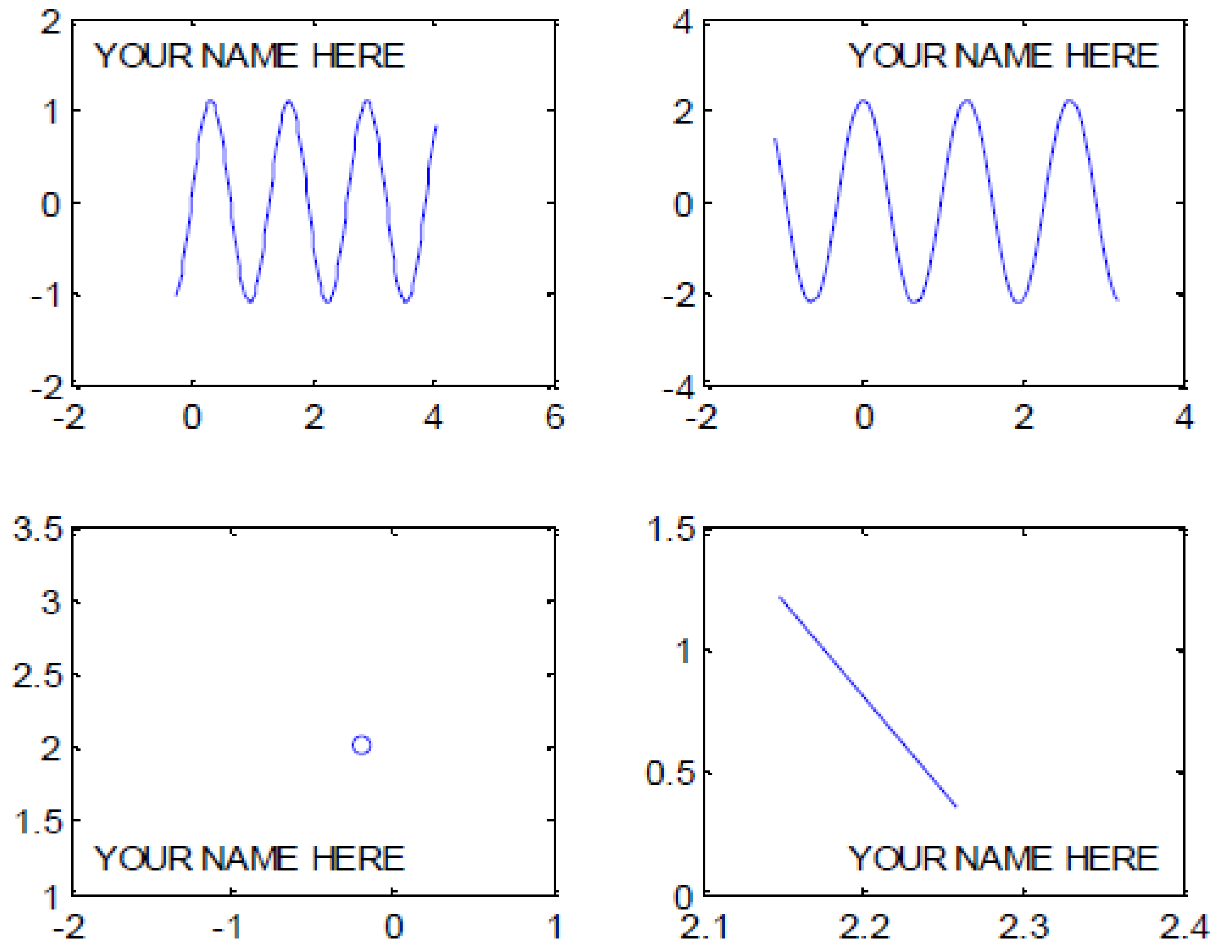
2. Determine if your function has any inputs or outputs and modify the first line accordingly (i.e. remove the inputs and/or outputs if they are not needed). You will need an input that will determine which corner of the plot your name will appear in.
3. Add a couple of lines of comments just below the line that starts with the word `function` that describes what the function does and what inputs (type and values) are expected.

4. Your function should do the following:
    a. Extract the x and y limits of the current figure.
    b. Use the x and y limits to determine an x,y coordinate that is 5% of the total x range from the edge and 10% of the total y range from the edge of the plot. The exact values will depend on the input parameter.
    c. Depending on the input you will put your name on the current figure one of the following locations:
        i. Upper right corner
        ii. Upper left corner
        iii. Lower right corner
        iv. Lower left corner
    d. Use the "text" command and the coordinate you just calculated to print YOUR name on the plot.
5. In the comment section, describe what your function does, state the creation date and author (you), and anything else you feel is relevant.
6. Test your function with the following code (i.e. copy and paste the code into a script file (e.g. functionTest.m) that is DIFFERENT from your function file). Call your function from within the code by following the instructions in RED ALL-CAPS TEXT below. Run the test script multiple times to make sure your function works correctly.

```
%%
% This MATLAB script tests a user-defined function that places a name
% on randomly sized plots.

%% Generate randomized data to plot
clear all;clc;

% x data
xmin = (-10) + (10-(-10)).*rand; %Generate random number between -10
and 10
xrange = 2 + (5-2).*rand; %Generate random number between 2 and 5
xmax = xmin + xrange;
numPts = 150; %Number of data points
x = linspace(xmin,xmax,numPts);
x2 = x-0.2*xrange;

% y data
Amp = 0.5 + (2-0.5).*rand; %Generate random amplitude between 0.5 and 2
Freq = 0.5 + (1.5-0.5).*rand; %Generate random freq between 0.5 and 1.5
y = Amp*sin(2*pi*Freq*x);
y2 = 2*Amp*cos(2*pi*Freq*x2);

%% Plot data and test your function
r = 2; %number of subplot rows
c = 2; %number of subplot columns

subplot(r,c,1)
plot(x,y)
%****************************
%TYPE THE NAME OF YOUR FUNCTION HERE TO PUT YOUR NAME IN THE UPPER LEFT
%****************************
```

```
subplot(r,c,2)
plot(x2,y2)
%****************************
%TYPE THE NAME OF YOUR FUNCTION HERE TO PUT YOUR NAME IN THE UPPER
RIGHT
%****************************

subplot(r,c,3)
plot(-5*rand,3*rand,'o')
%****************************
%TYPE THE NAME OF YOUR FUNCTION HERE TO PUT YOUR NAME IN THE LOWER LEFT
%****************************

subplot(r,c,4)
plot([5*rand 5*rand],[2*rand,6*rand])
%****************************
%TYPE THE NAME OF YOUR FUNCTION HERE TO PUT YOUR NAME IN THE LOWER
RIGHT
%****************************
```

7. If your function works correctly, your name should always appear in the same position each time the test script is run and should look something like this (but with your name on the plots):



8. Save your function so you can use it to put your name on plots in future assignments.