Financial Econometrics

UNC CHARLOTTE

BELK COLLEGE *of* BUSINESS

Lecture 1: Introduction to MATLAB

Adapted from
Introduction to Programming in MATLAB lecture notes,
Danilo Šćepanović, MIT OpenCourseWare

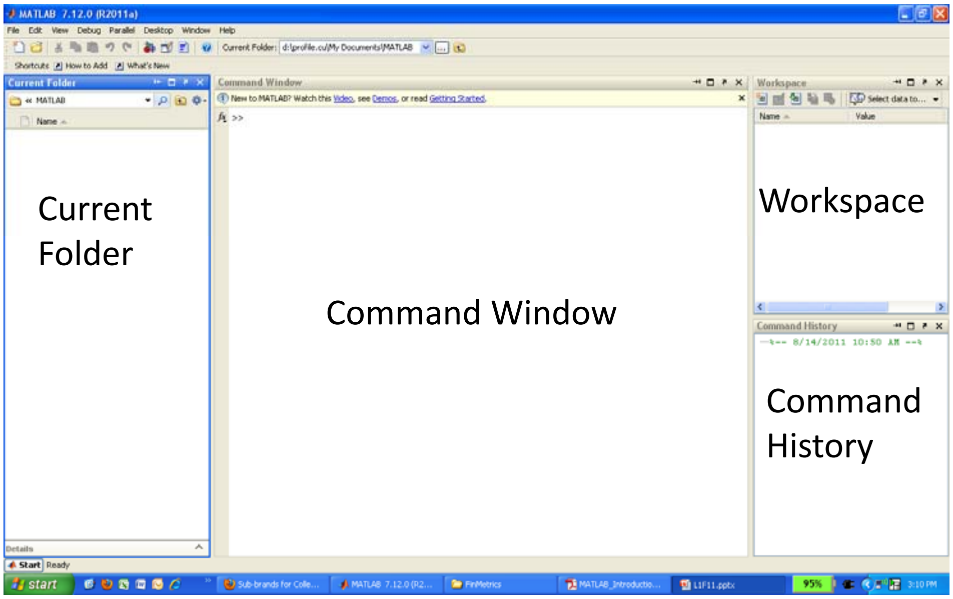
Getting Started

UNC CHARLOTTE

MATLAB Basics

- MATLAB can be thought of as a super-powerful graphing calculator
  - Remember the TI-83 from calculus?
  - With many more buttons (built-in functions)
- In addition it is a programming language
  - MATLAB is an interpreted language, like Java
  - Commands executed line by line

UNC CHARLOTTE

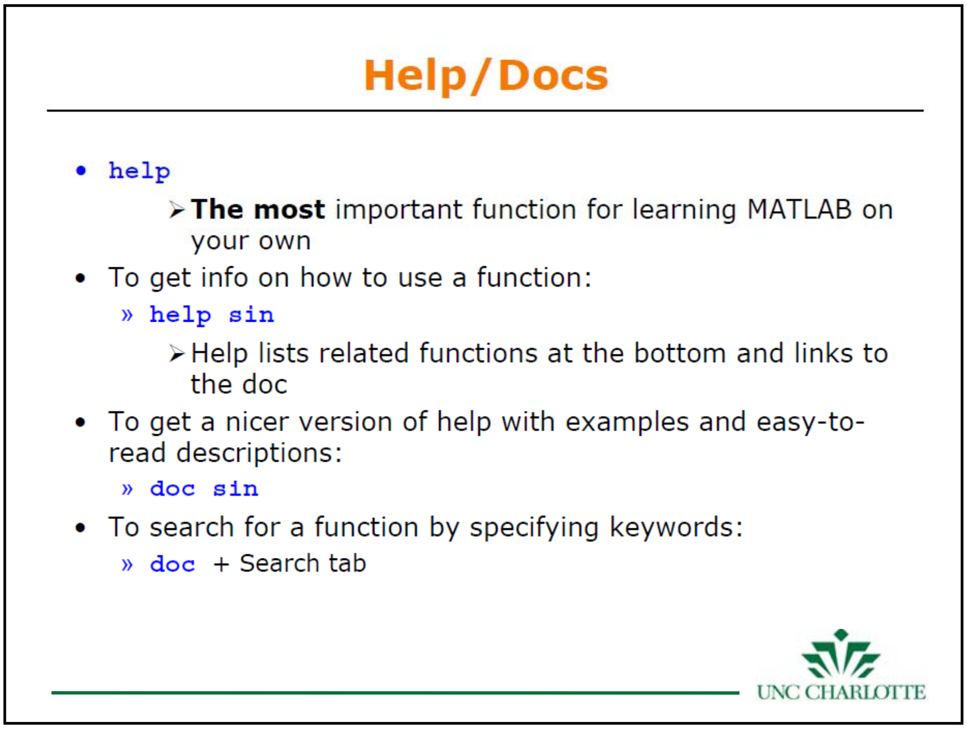


New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
>>
```

```
--%-- 8/14/2011 10:50 AM --%
```

UNC CHARLOTTE



Help/Docs

- `help`
  - ➢ **The most** important function for learning MATLAB on your own
- To get info on how to use a function:
  - » `help sin`
    - ➢ Help lists related functions at the bottom and links to the doc
- To get a nicer version of help with examples and easy-to-read descriptions:
  - » `doc sin`
- To search for a function by specifying keywords:
  - » `doc` + Search tab

UNC CHARLOTTE

Scripts

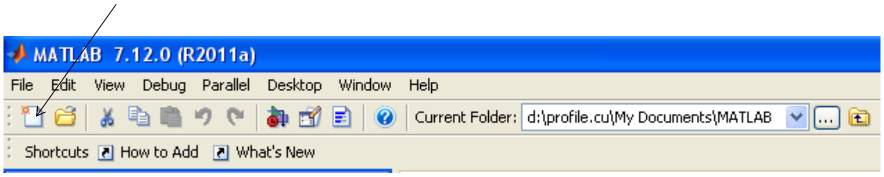
UNC CHARLOTTE

Scripts: Overview

- Scripts are
  - collection of commands executed in sequence
  - written in the MATLAB editor
  - saved as MATLAB files (.m extension)
- To create an MATLAB file from command-line

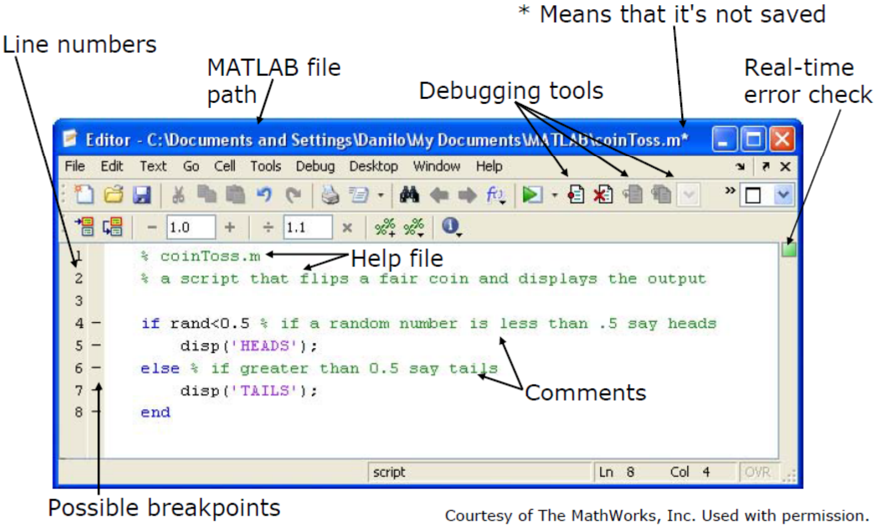
```
» edit helloWorld.m
```

- or click


MATLAB 7.12.0 (R2011a)
File Edit View Debug Parallel Desktop Window Help
Current Folder: d:\profile.cu\My Documents\MATLAB
Shortcuts
How to Add
What's New

UNC CHARLOTTE

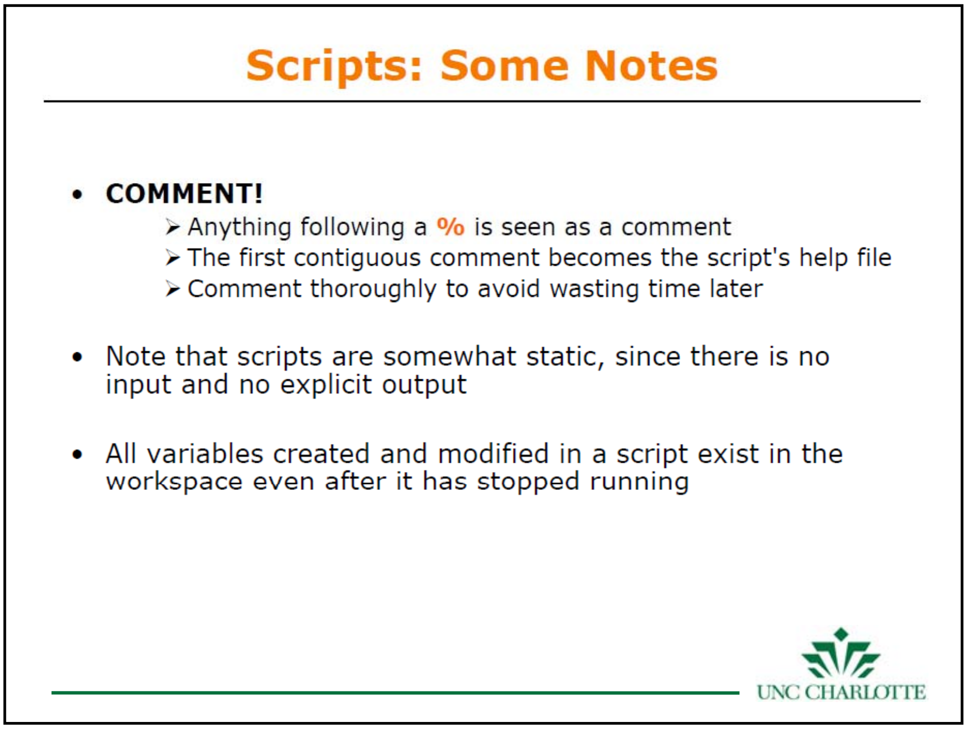
Scripts: the Editor


* Means that it's not saved
Line numbers
MATLAB file path
Debugging tools
Real-time error check
Editor - C:\Documents and Settings\Danilo\My Documents\MATLAB\coinToss.m*
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
Help file
Comments
Possible breakpoints
script
Ln 8
Col 4
OVR

```
% coinToss.m
% a script that flips a fair coin and displays the output

if rand<0.5 % if a random number is less than .5 say heads
    disp('HEADS');
else % if greater than 0.5 say tails
    disp('TAILS');
end
```

Courtesy of The MathWorks, Inc. Used with permission.

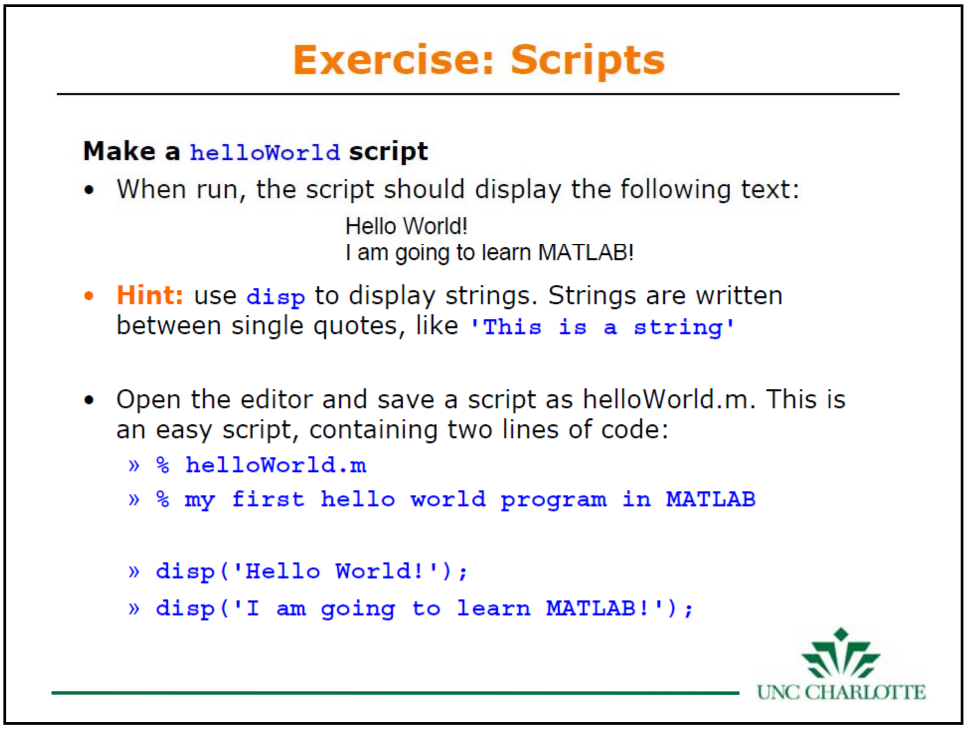
UNC CHARLOTTE



Scripts: Some Notes

- **COMMENT!**
  - Anything following a % is seen as a comment
  - The first contiguous comment becomes the script's help file
  - Comment thoroughly to avoid wasting time later
- Note that scripts are somewhat static, since there is no input and no explicit output
- All variables created and modified in a script exist in the workspace even after it has stopped running

UNC CHARLOTTE



Exercise: Scripts

Make a `helloWorld` script

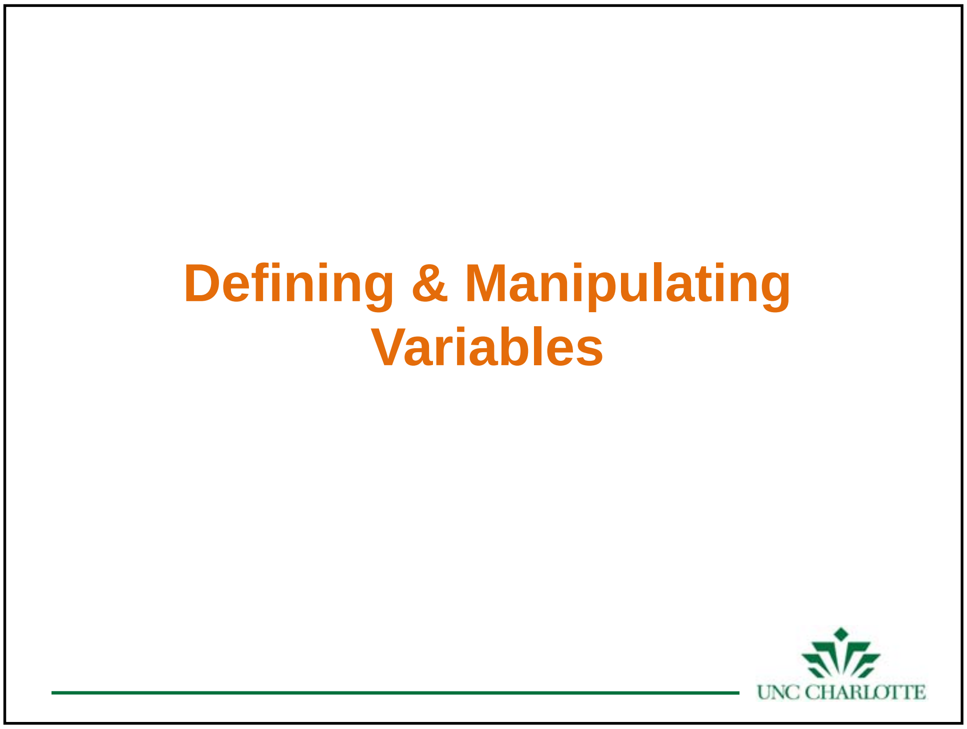
- When run, the script should display the following text:

```
Hello World!
I am going to learn MATLAB!
```

- **Hint:** use `disp` to display strings. Strings are written between single quotes, like `'This is a string'`
- Open the editor and save a script as helloWorld.m. This is an easy script, containing two lines of code:

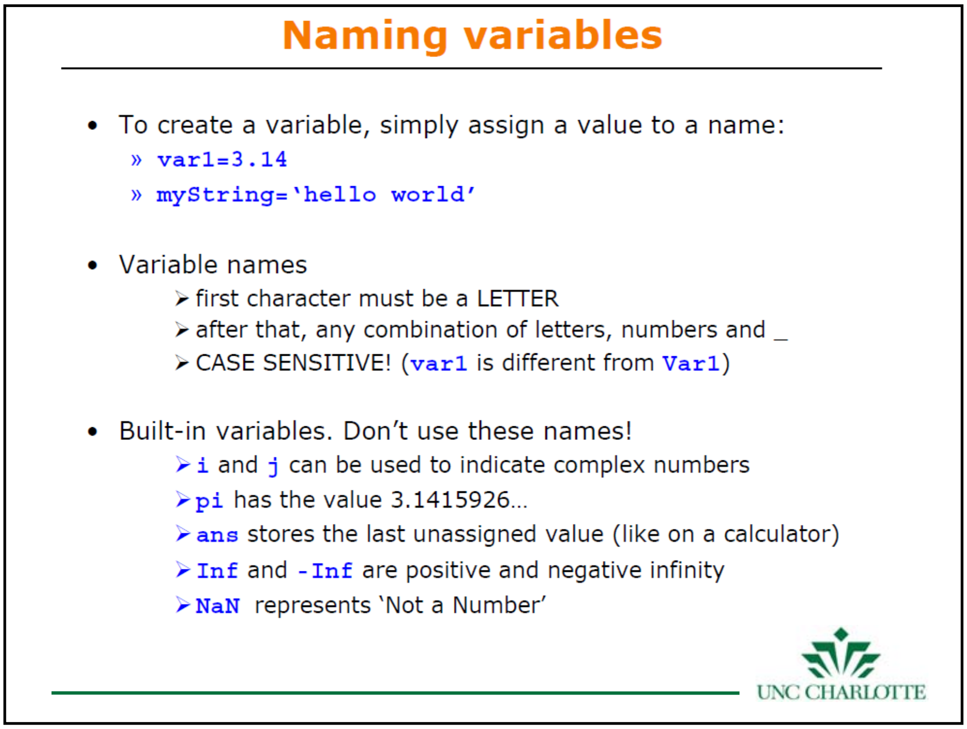
```
» % helloWorld.m
» % my first hello world program in MATLAB

» disp('Hello World!');
» disp('I am going to learn MATLAB!');
```

UNC CHARLOTTE



Defining & Manipulating Variables

UNC CHARLOTTE



Naming variables

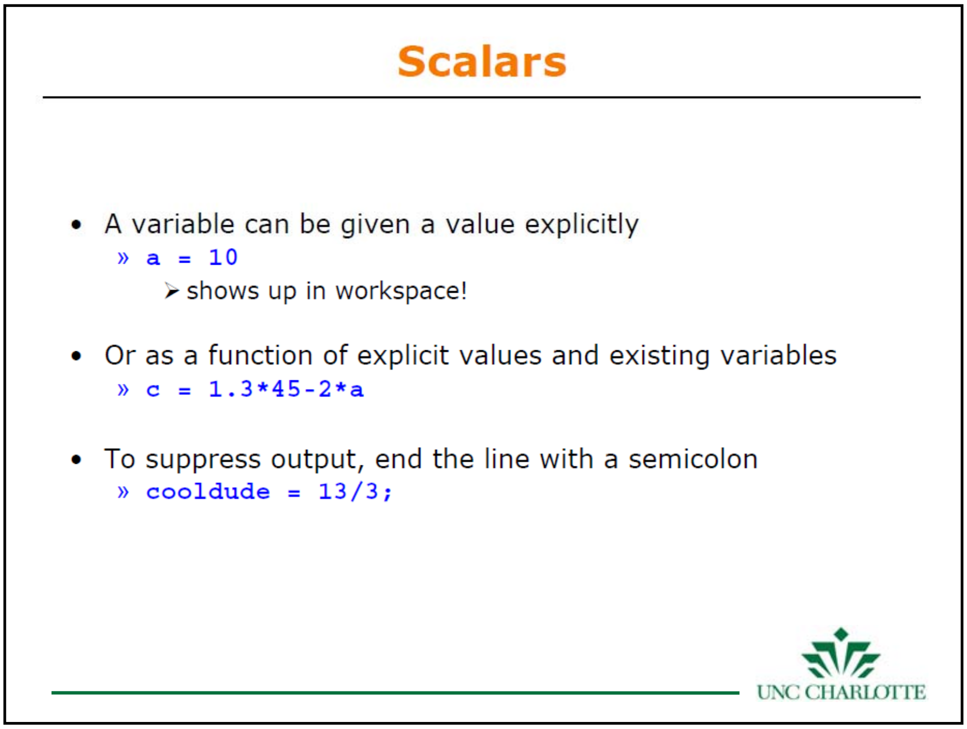
- To create a variable, simply assign a value to a name:

```
» var1=3.14
» myString='hello world'
```

- Variable names
  - first character must be a LETTER
  - after that, any combination of letters, numbers and _
  - CASE SENSITIVE! (`var1` is different from `Var1`)

- Built-in variables. Don't use these names!
  - `i` and `j` can be used to indicate complex numbers
  - `pi` has the value 3.1415926…
  - `ans` stores the last unassigned value (like on a calculator)
  - `Inf` and `-Inf` are positive and negative infinity
  - `NaN` represents 'Not a Number'

UNC CHARLOTTE



Scalars

- A variable can be given a value explicitly

```
» a = 10
```

  - ➢ shows up in workspace!

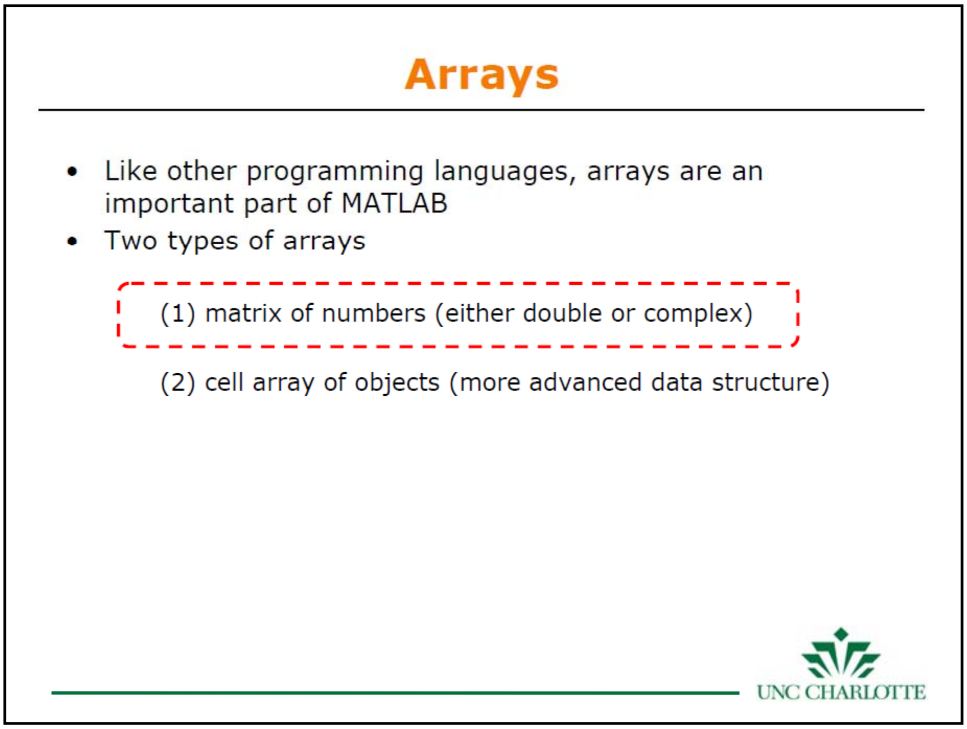
- Or as a function of explicit values and existing variables

```
» c = 1.3*45-2*a
```

- To suppress output, end the line with a semicolon

```
» cooldude = 13/3;
```

UNC CHARLOTTE



Arrays

- Like other programming languages, arrays are an important part of MATLAB
- Two types of arrays

(1) matrix of numbers (either double or complex)

(2) cell array of objects (more advanced data structure)

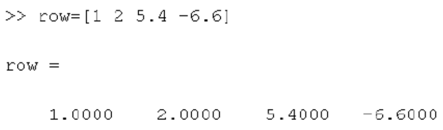
UNC CHARLOTTE

Row Vectors

- Row vector: comma or space separated values between brackets

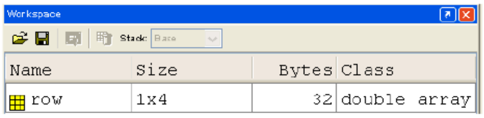
```
» row = [1 2 5.4 -6.6]
» row = [1, 2, 5.4, -6.6];
```

- Command window:



```
>> row=[1 2 5.4 -6.6]

row =

    1.0000    2.0000    5.4000   -6.6000
```

- Workspace:



Name	Size	Bytes	Class
row	1x4	32	double array

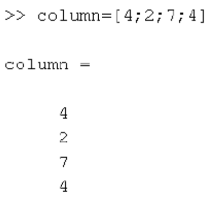
UNC CHARLOTTE

Column Vectors

- Column vector: semicolon separated values between brackets

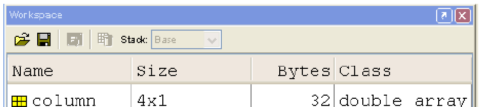
```
» column = [4;2;7;4]
```

- Command window:



```
>> column=[4;2;7;4]

column =

     4
     2
     7
     4
```

- Workspace:

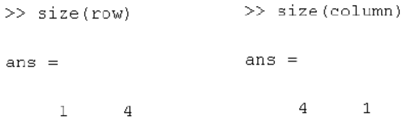


Name	Size	Bytes	Class
column	4x1	32	double array

UNC CHARLOTTE

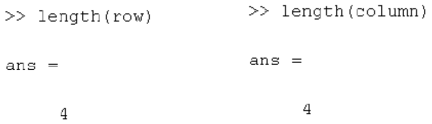
size & length

- You can tell the difference between a row and a column vector by:
  - ➢ Looking in the workspace
  - ➢ Displaying the variable in the command window
  - ➢ Using the size function



```
>> size(row)            >> size(column)

ans =                   ans =

     1     4                 4     1
```

- To get a vector's length, use the length function



```
>> length(row)          >> length(column)

ans =                   ans =

     4                       4
```

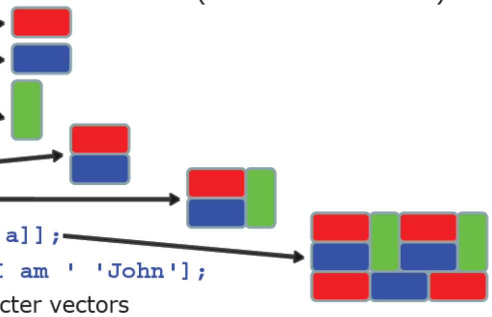
Matrices

- Make matrices like vectors
- Element by element
  - » a= [1 2;3 4];

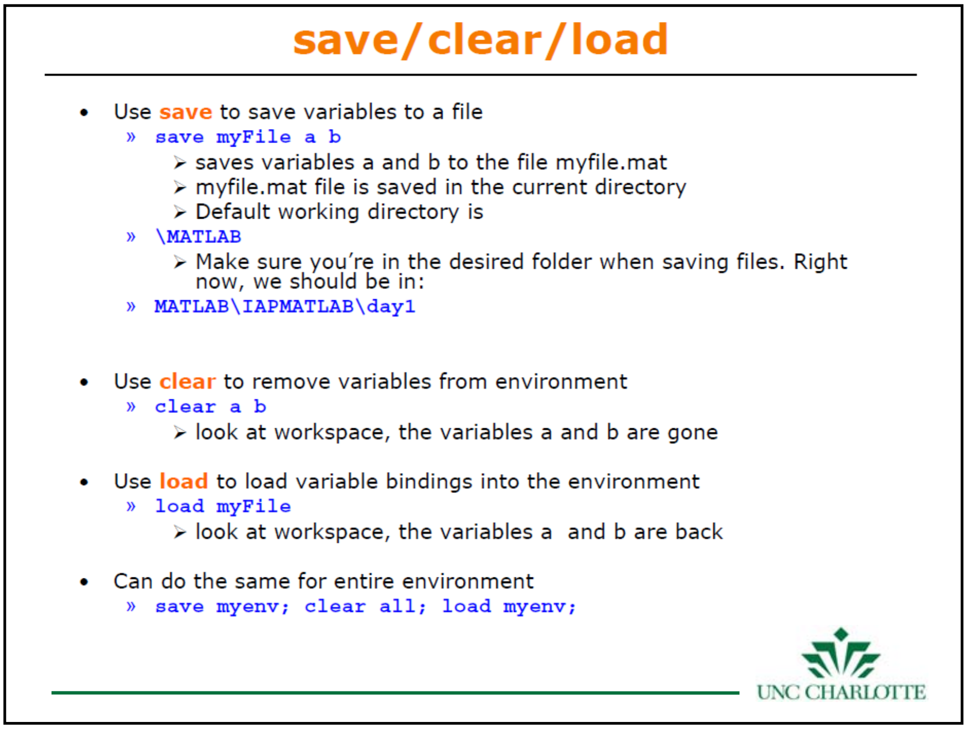
$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- By concatenating vectors or matrices (dimension matters)

```
» a = [1 2];
» b = [3 4];
» c = [5;6];

» d = [a;b];
» e = [d c];
» f = [[e e];[a b a]];
» str = ['Hello, I am ' 'John'];
```



  - ➢ Strings are character vectors



save/clear/load

- Use **save** to save variables to a file

```
» save myFile a b
```

  - ➢ saves variables a and b to the file myfile.mat
  - ➢ myfile.mat file is saved in the current directory
  - ➢ Default working directory is

```
» \MATLAB
```

  - ➢ Make sure you're in the desired folder when saving files. Right now, we should be in:

```
» MATLAB\IAPMATLAB\day1
```

- Use **clear** to remove variables from environment

```
» clear a b
```

  - ➢ look at workspace, the variables a and b are gone

- Use **load** to load variable bindings into the environment

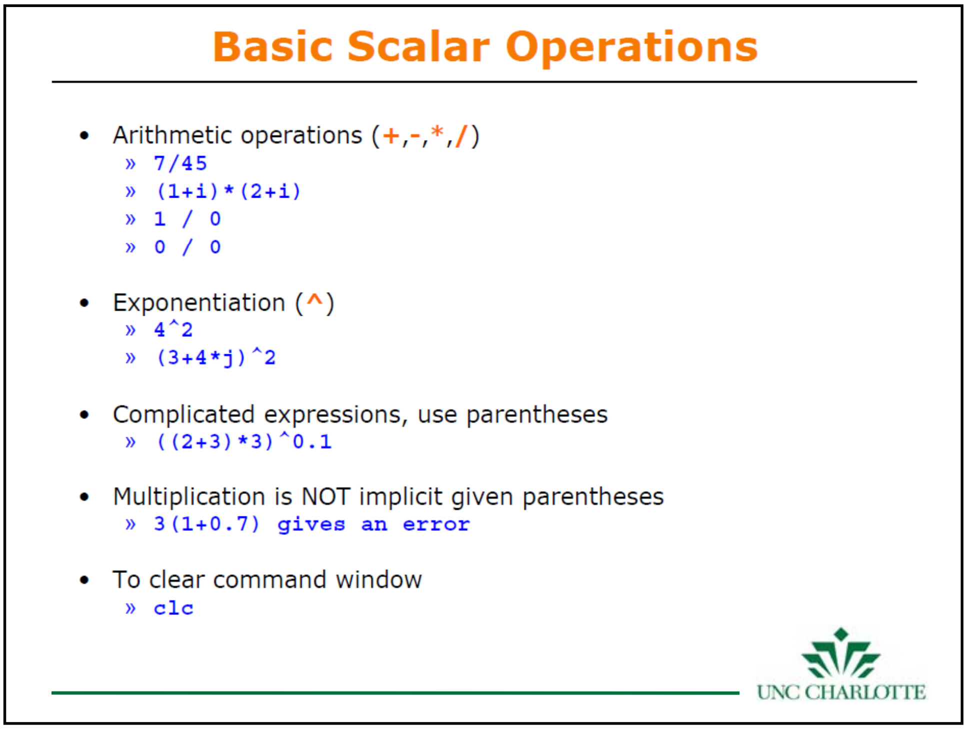
```
» load myFile
```

  - ➢ look at workspace, the variables a and b are back

- Can do the same for entire environment

```
» save myenv; clear all; load myenv;
```

UNC CHARLOTTE



Basic Scalar Operations

- Arithmetic operations (+,-,*,/)

```
» 7/45
» (1+i)*(2+i)
» 1 / 0
» 0 / 0
```

- Exponentiation (^)

```
» 4^2
» (3+4*j)^2
```

- Complicated expressions, use parentheses

```
» ((2+3)*3)^0.1
```

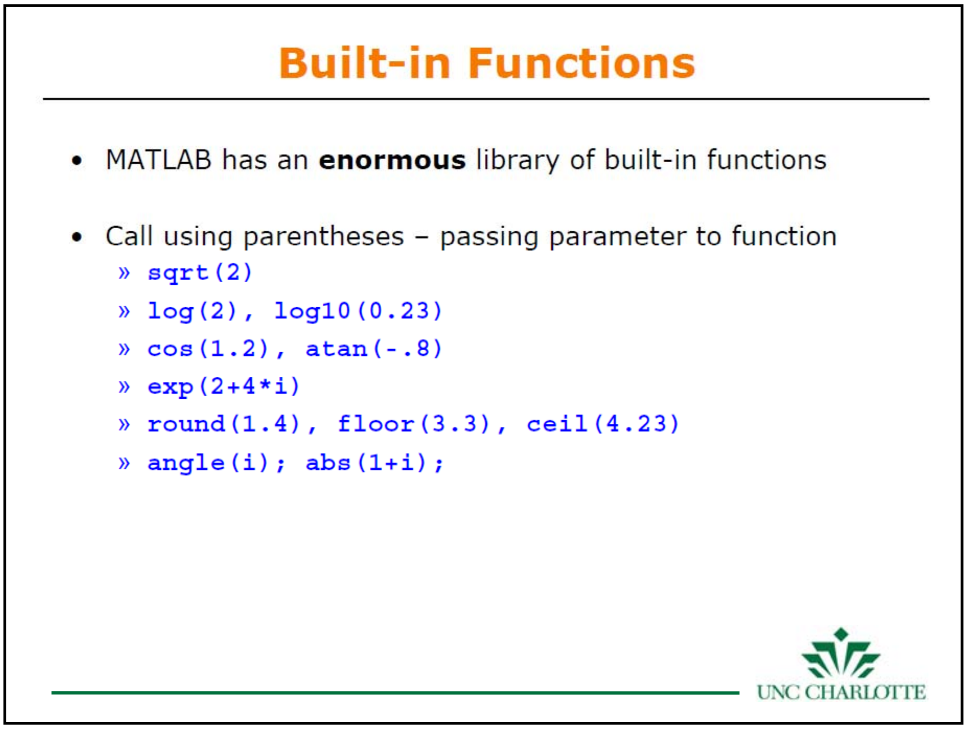
- Multiplication is NOT implicit given parentheses

```
» 3(1+0.7) gives an error
```

- To clear command window

```
» clc
```

UNC CHARLOTTE

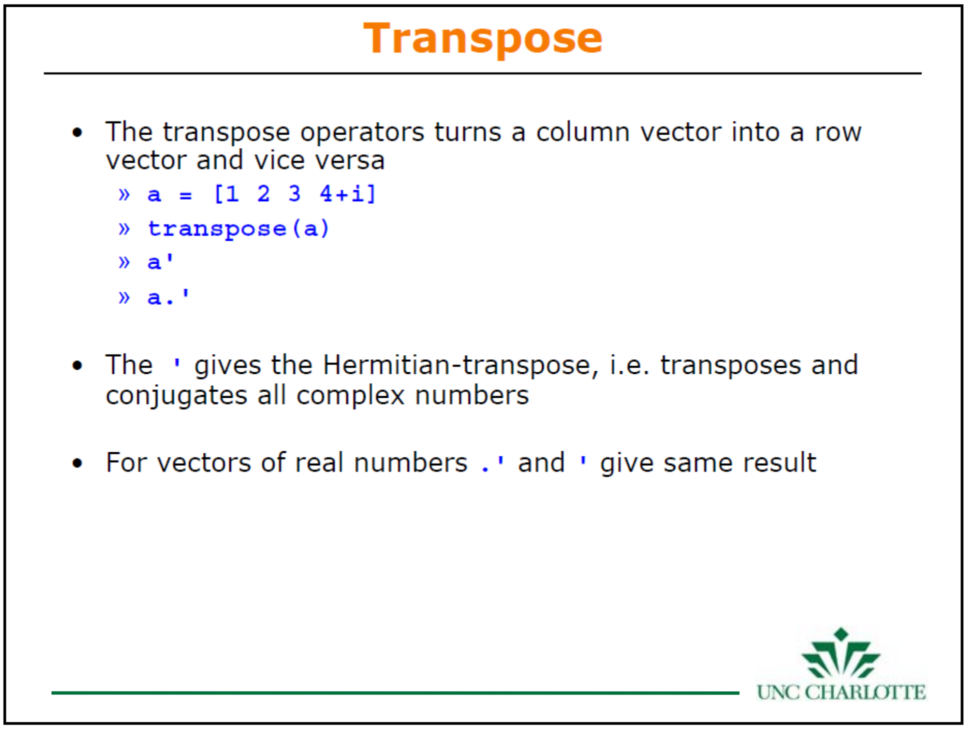


Built-in Functions

- MATLAB has an **enormous** library of built-in functions
- Call using parentheses – passing parameter to function

```
» sqrt(2)
» log(2), log10(0.23)
» cos(1.2), atan(-.8)
» exp(2+4*i)
» round(1.4), floor(3.3), ceil(4.23)
» angle(i); abs(1+i);
```

UNC CHARLOTTE



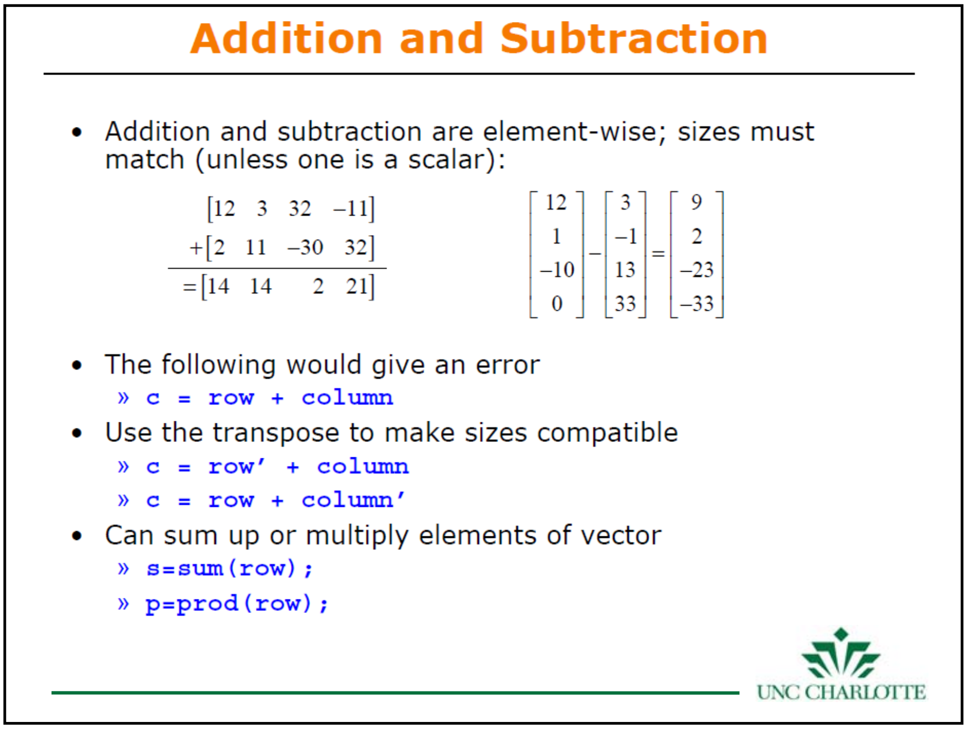
Transpose

- The transpose operators turns a column vector into a row vector and vice versa

```
» a = [1 2 3 4+i]
» transpose(a)
» a'
» a.'
```

- The `'` gives the Hermitian-transpose, i.e. transposes and conjugates all complex numbers
- For vectors of real numbers `.'` and `'` give same result

UNC CHARLOTTE



Addition and Subtraction

- Addition and subtraction are element-wise; sizes must match (unless one is a scalar):

$$\begin{array}{r} [12 \quad 3 \quad 32 \quad -11] \\ +[2 \quad 11 \quad -30 \quad 32] \\ \hline =[14 \quad 14 \quad 2 \quad 21] \end{array} \qquad \begin{bmatrix} 12 \\ 1 \\ -10 \\ 0 \end{bmatrix} - \begin{bmatrix} 3 \\ -1 \\ 13 \\ 33 \end{bmatrix} = \begin{bmatrix} 9 \\ 2 \\ -23 \\ -33 \end{bmatrix}$$

- The following would give an error

```
» c = row + column
```

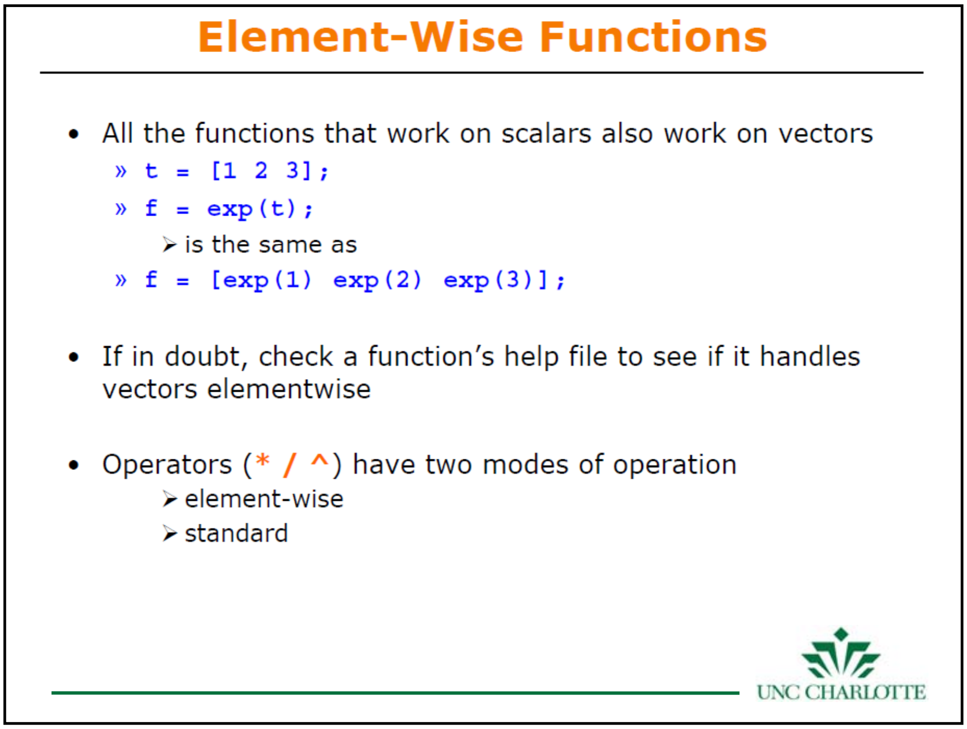
- Use the transpose to make sizes compatible

```
» c = row’ + column
» c = row + column’
```

- Can sum up or multiply elements of vector

```
» s=sum(row);
» p=prod(row);
```

UNC CHARLOTTE



Element-Wise Functions

- All the functions that work on scalars also work on vectors

```
» t = [1 2 3];
» f = exp(t);
```

  - ➢ is the same as

```
» f = [exp(1) exp(2) exp(3)];
```

- If in doubt, check a function’s help file to see if it handles vectors elementwise

- Operators (* / ^) have two modes of operation
  - ➢ element-wise
  - ➢ standard

UNC CHARLOTTE

Operators: element-wise

- To do element-wise operations, use the dot: `.` (`.*`, `./`, `.^`). BOTH dimensions must match (unless one is scalar)!

```
» a=[1 2 3];b=[4;2;1];
» a.*b, a./b, a.^b → all errors
» a.*b', a./b', a.^(b') → all valid
```

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}.*\begin{bmatrix}4\\2\\1\end{bmatrix}=ERROR$$

$$\begin{bmatrix}1\\2\\3\end{bmatrix}.*\begin{bmatrix}4\\2\\1\end{bmatrix}=\begin{bmatrix}4\\4\\3\end{bmatrix}$$

$$3\times1.*3\times1=3\times1$$

$$\begin{bmatrix}1&1&1\\2&2&2\\3&3&3\end{bmatrix}.*\begin{bmatrix}1&2&3\\1&2&3\\1&2&3\end{bmatrix}=\begin{bmatrix}1&2&3\\2&4&6\\3&6&9\end{bmatrix}$$

$$3\times3.*3\times3=3\times3$$

$$\begin{bmatrix}1&2\\3&4\end{bmatrix}.\hat{}2=\begin{bmatrix}1^2&2^2\\3^2&4^2\end{bmatrix}$$

Can be any dimension

Operators: standard

- Multiplication can be done in a standard way or element-wise
- Standard multiplication (*) is either a dot-product or an outer-product
  - Remember from linear algebra: inner dimensions must MATCH!!
- Standard exponentiation (^) can only be done on square matrices or scalars
- Left and right division (/ \) is same as multiplying by inverse
  - Our recommendation: just multiply by inverse (more on this later)

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}*\begin{bmatrix}4\\2\\1\end{bmatrix}=11$$

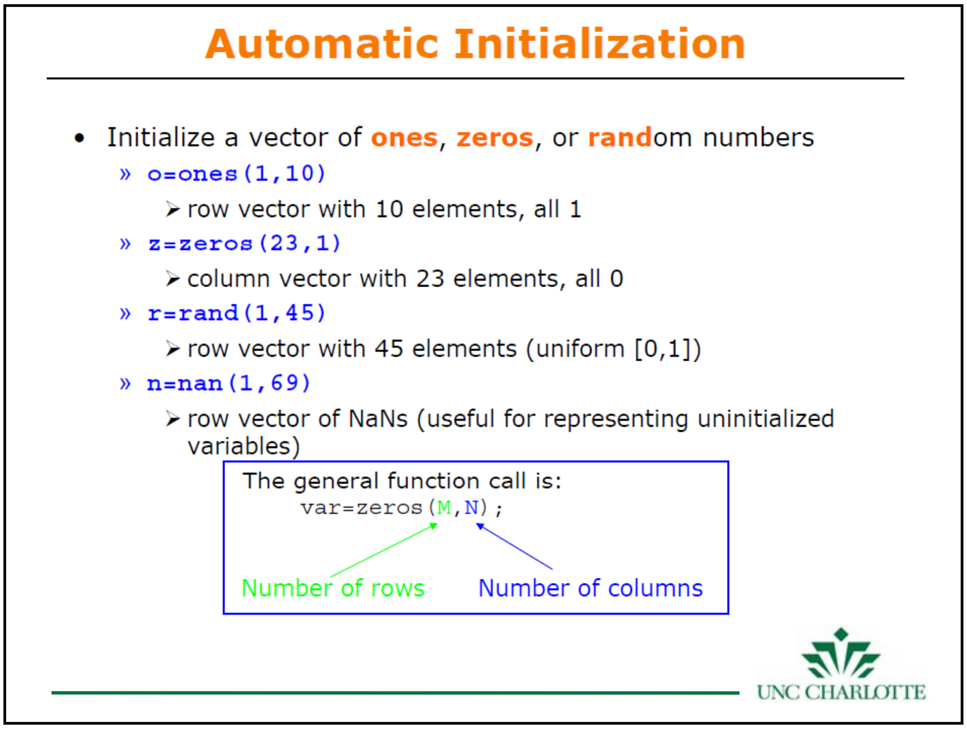
$$1\times3*3\times1=1\times1$$

$$\begin{bmatrix}1&2\\3&4\end{bmatrix}\hat{}2=\begin{bmatrix}1&2\\3&4\end{bmatrix}*\begin{bmatrix}1&2\\3&4\end{bmatrix}$$

Must be square to do powers

$$\begin{bmatrix}1&1&1\\2&2&2\\3&3&3\end{bmatrix}*\begin{bmatrix}1&2&3\\1&2&3\\1&2&3\end{bmatrix}=\begin{bmatrix}3&6&9\\6&12&18\\9&18&27\end{bmatrix}$$

$$3\times3*3\times3=3\times3$$



Automatic Initialization

- Initialize a vector of **ones**, **zeros**, or **rand**om numbers

```
» o=ones(1,10)
```

  - row vector with 10 elements, all 1

```
» z=zeros(23,1)
```

  - column vector with 23 elements, all 0

```
» r=rand(1,45)
```

  - row vector with 45 elements (uniform [0,1])

```
» n=nan(1,69)
```

  - row vector of NaNs (useful for representing uninitialized variables)

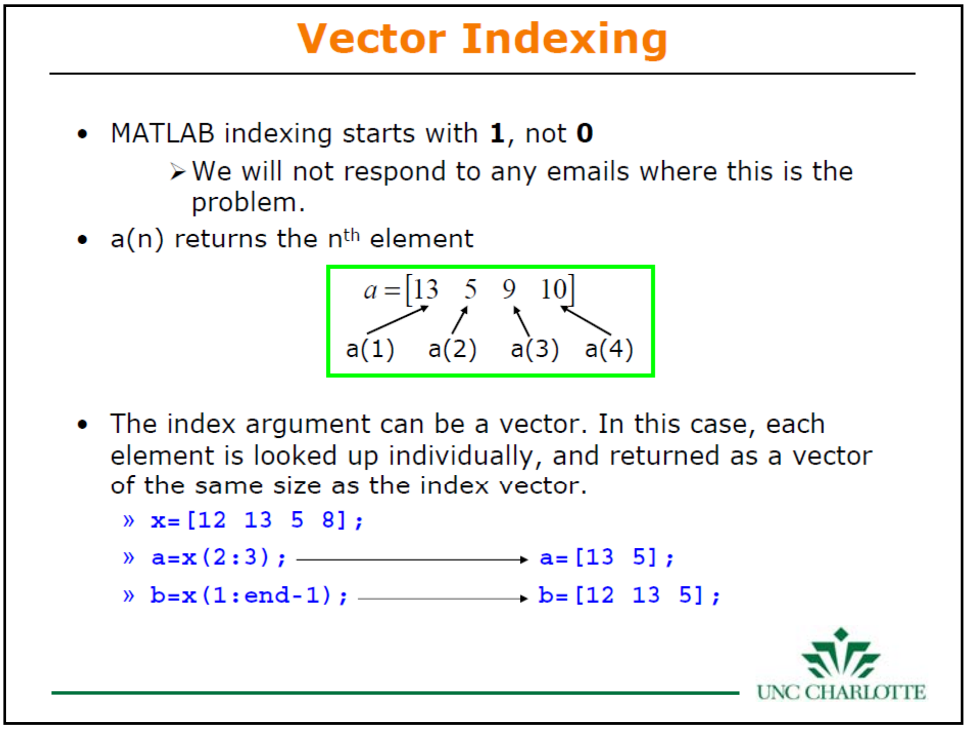
The general function call is:

```
var=zeros(M,N);
```

Number of rows

Number of columns

UNC CHARLOTTE



Vector Indexing

- MATLAB indexing starts with **1**, not **0**
  - We will not respond to any emails where this is the problem.
- a(n) returns the nth element

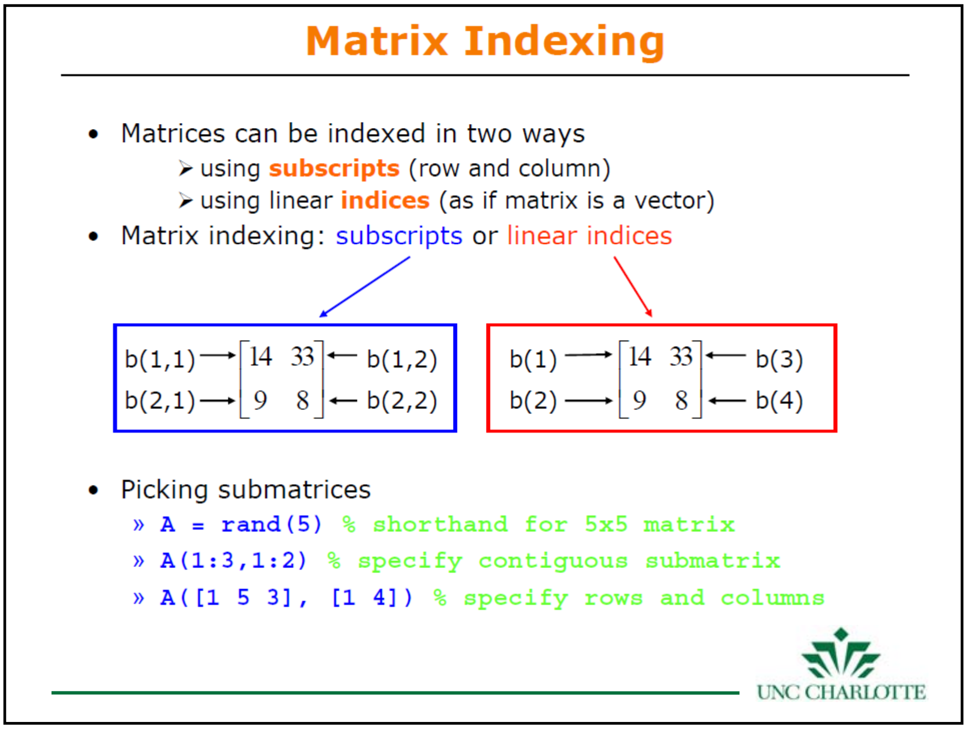
$$a = [13 \quad 5 \quad 9 \quad 10]$$

a(1) a(2) a(3) a(4)

- The index argument can be a vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector.

```
» x=[12 13 5 8];
» a=x(2:3);        → a=[13 5];
» b=x(1:end-1);    → b=[12 13 5];
```

UNC CHARLOTTE



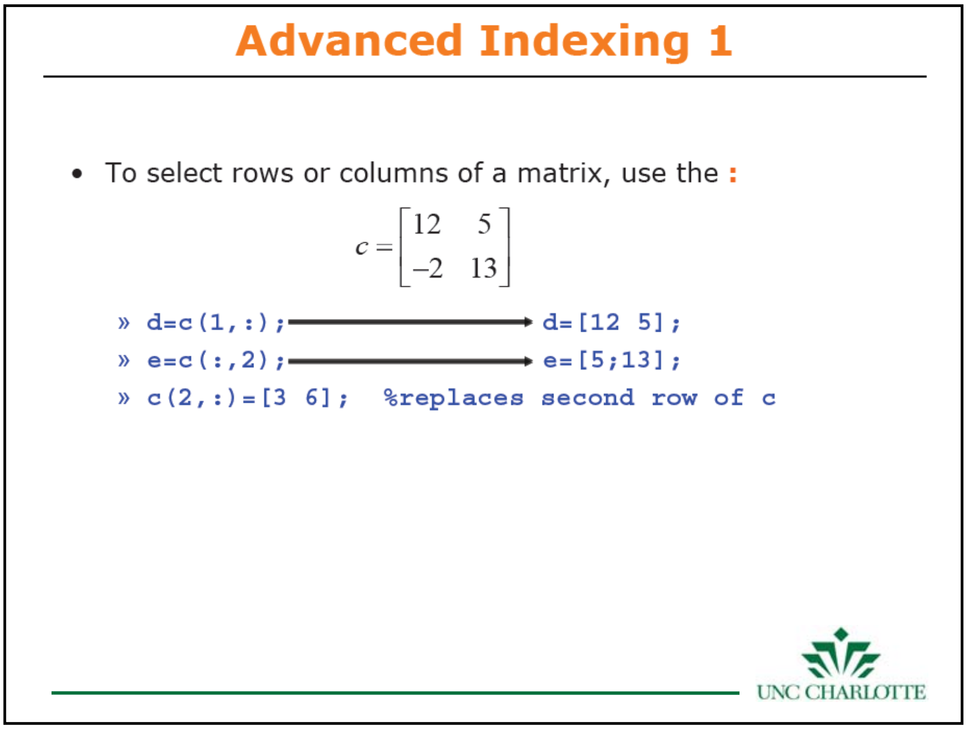
Matrix Indexing

- Matrices can be indexed in two ways
  - using **subscripts** (row and column)
  - using linear **indices** (as if matrix is a vector)
- Matrix indexing: subscripts or linear indices

$$\begin{matrix} b(1,1) \rightarrow \\ b(2,1) \rightarrow \end{matrix} \begin{bmatrix}14 & 33\\ 9 & 8\end{bmatrix} \begin{matrix} \leftarrow b(1,2) \\ \leftarrow b(2,2)\end{matrix} \qquad \begin{matrix} b(1) \rightarrow \\ b(2) \rightarrow \end{matrix} \begin{bmatrix}14 & 33\\ 9 & 8\end{bmatrix} \begin{matrix} \leftarrow b(3) \\ \leftarrow b(4)\end{matrix}$$

- Picking submatrices

```
» A = rand(5) % shorthand for 5x5 matrix
» A(1:3,1:2) % specify contiguous submatrix
» A([1 5 3], [1 4]) % specify rows and columns
```



Advanced Indexing 1

- To select rows or columns of a matrix, use the **:**

$$c = \begin{bmatrix}12 & 5\\ -2 & 13\end{bmatrix}$$

```
» d=c(1,:); ──────────→ d=[12 5];
» e=c(:,2); ──────────→ e=[5;13];
» c(2,:)=[3 6];  %replaces second row of c
```

Advanced Indexing 2

- MATLAB contains functions to help you find desired values within a vector or matrix

```
» vec = [5 3 1 9 7]
```

- To get the minimum value and its index:

```
» [minVal,minInd] = min(vec);
```

  - **max** works the same way

- To find any the indices of specific values or ranges

```
» ind = find(vec == 9);
» ind = find(vec > 2 & vec < 6);
```

  - **find** expressions can be very complex, more on this later

- To convert between subscripts and indices, use **ind2sub**, and **sub2ind**. Look up **help** to see how to use them.

UNC CHARLOTTE

Making Figures

UNC CHARLOTTE

Plotting

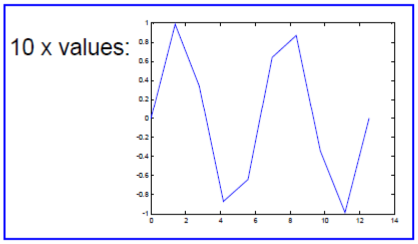
- Example
  ```
  » x=linspace(0,4*pi,10);
  » y=sin(x);
  ```
- Plot values against their index
  ```
  » plot(y);
  ```
- Usually we want to plot y versus x
  ```
  » plot(x,y);
  ```

UNC CHARLOTTE

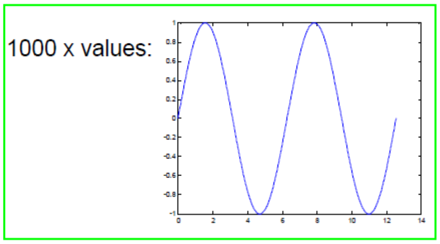
What does plot do?

- **plot** generates dots at each (x,y) pair and then connects the dots with a line
- To make plot of a function look smoother, evaluate at more points
  ```
  » x=linspace(0,4*pi,1000);
  » plot(x,sin(x));
  ```
- x and y vectors must be same size or else you'll get an error
  ```
  » plot([1 2], [1 2 3])
  ```
    - ➢ error!!

10 x values:



1000 x values:



UNC CHARLOTTE

Plot Options

- Can change the line color, marker style, and line style by adding a string argument

```
» plot(x,y,'k.-');
```

color marker line-style

- Can plot without connecting the dots by omitting line style argument

```
» plot(x,y,'.')
```

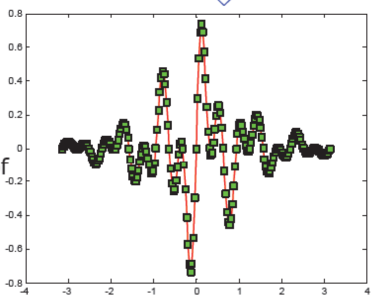
- Look at **help plot** for a full list of colors, markers, and linestyles

Line and Marker Options

- Everything on a line can be customized

```
» plot(x,y,'--s','LineWidth',2,...
          'Color', [1 0 0], ...
          'MarkerEdgeColor','k',...
          'MarkerFaceColor','g',...
          'MarkerSize',10)
```

You can set colors by using a vector of [R G B] values or a predefined color character like 'g', 'k', etc.



- See **doc line_props** for a full list of properties that can be specified

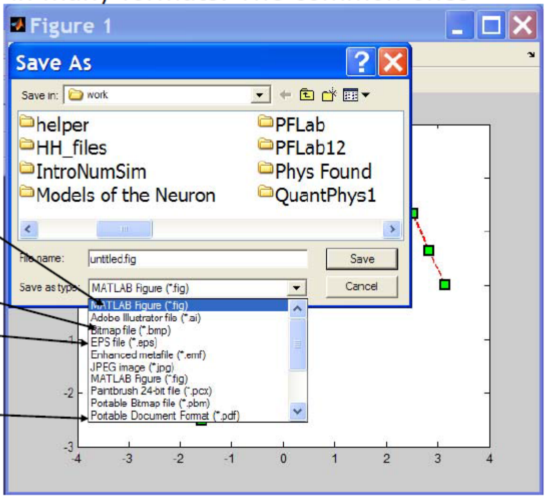
Multiple Plots in one Figure

- To have multiple axes in one figure
  - » `subplot(2,3,1)`
    - ➢ makes a figure with 2 rows and three columns of axes, and activates the first axis for plotting
    - ➢ each axis can have labels, a legend, and a title
  - » `subplot(2,3,4:6)`
    - ➢ activating a range of axes fuses them into one
- To close existing figures
  - » `close([1 3])`
    - ➢ closes figures 1 and 3
  - » `close all`
    - ➢ closes all figures (useful in scripts/functions)

UNC CHARLOTTE

Saving Figures

- Figures can be saved in many formats. The common ones are:



File name: untitled.fig

Save as type: MATLAB Figure (*.fig)

.fig preserves all information

.bmp uncompressed image

.eps high-quality scaleable format

.pdf compressed image

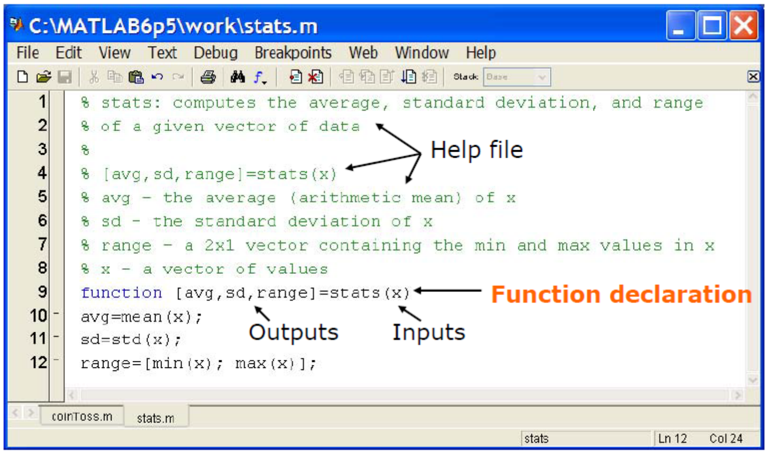
UNC CHARLOTTE

Defining New Functions

UNC CHARLOTTE

User-defined Functions

- Functions look exactly like scripts, but for **ONE** difference
  - Functions must have a function declaration

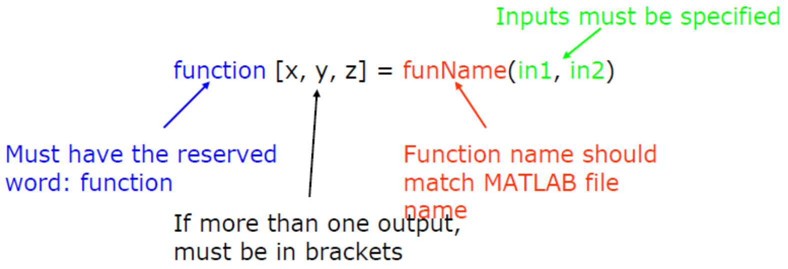


```
% stats: computes the average, standard deviation, and range
% of a given vector of data
%
% [avg,sd,range]=stats(x)
% avg - the average (arithmetic mean) of x
% sd - the standard deviation of x
% range - a 2x1 vector containing the min and max values in x
% x - a vector of values
function [avg,sd,range]=stats(x)
avg=mean(x);
sd=std(x);
range=[min(x); max(x)];
```

UNC CHARLOTTE

User-defined Functions

- Some comments about the function declaration



Inputs must be specified

function [x, y, z] = funName(in1, in2)

Must have the reserved word: function

Function name should match MATLAB file name

If more than one output, must be in brackets

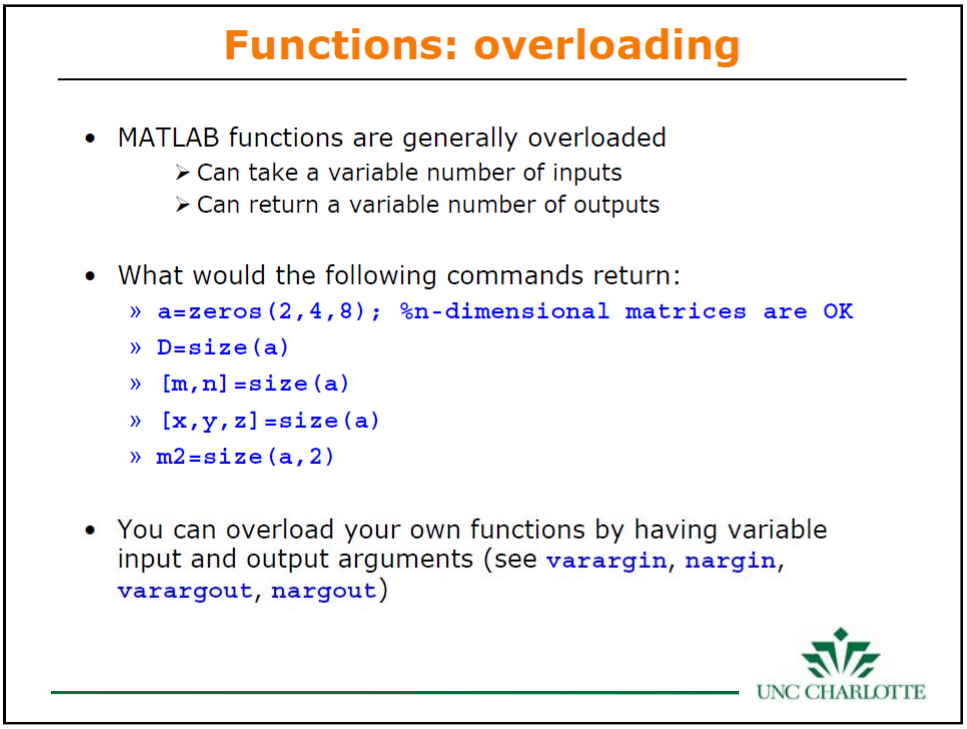
- No need for return: MATLAB 'returns' the variables whose names match those in the function declaration
- Variable scope: Any variables created within the function but not returned disappear after the function stops running

Functions: overloading

- We're familiar with
  - » **zeros**
  - » **size**
  - » **length**
  - » **sum**

- Look at the help file for size by typing
  - » **help size**

- The help file describes several ways to invoke the function
  - ➢ D = SIZE(X)
  - ➢ [M,N] = SIZE(X)
  - ➢ [M1,M2,M3,...,MN] = SIZE(X)
  - ➢ M = SIZE(X,DIM)



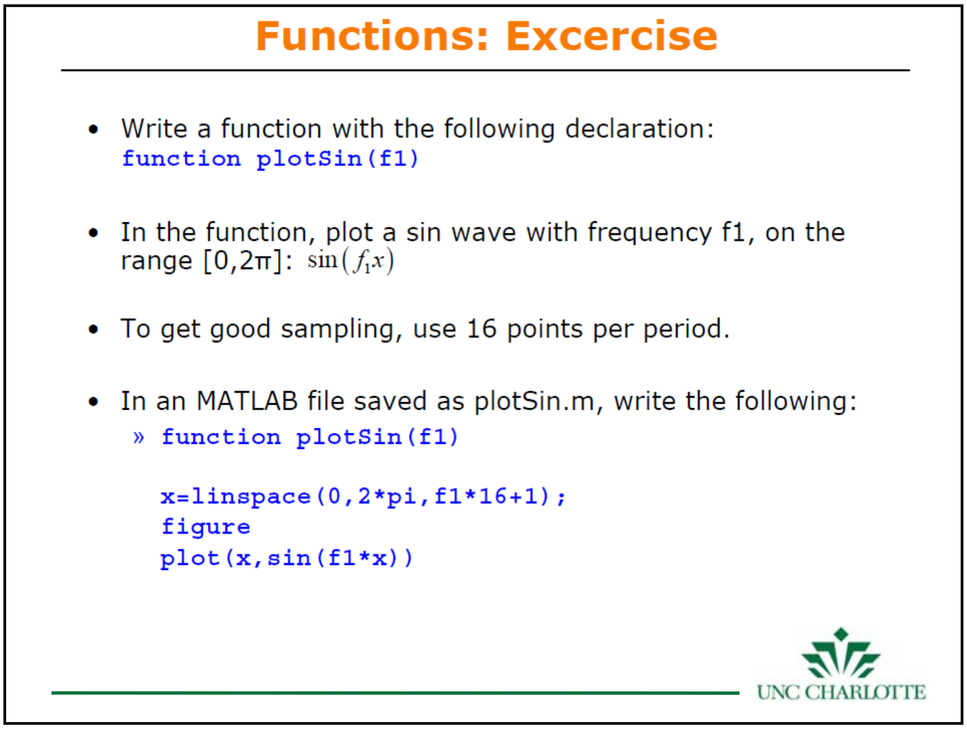
Functions: overloading

- MATLAB functions are generally overloaded
  - Can take a variable number of inputs
  - Can return a variable number of outputs
- What would the following commands return:

```
» a=zeros(2,4,8); %n-dimensional matrices are OK
» D=size(a)
» [m,n]=size(a)
» [x,y,z]=size(a)
» m2=size(a,2)
```

- You can overload your own functions by having variable input and output arguments (see `varargin`, `nargin`, `varargout`, `nargout`)

UNC CHARLOTTE



Functions: Excercise

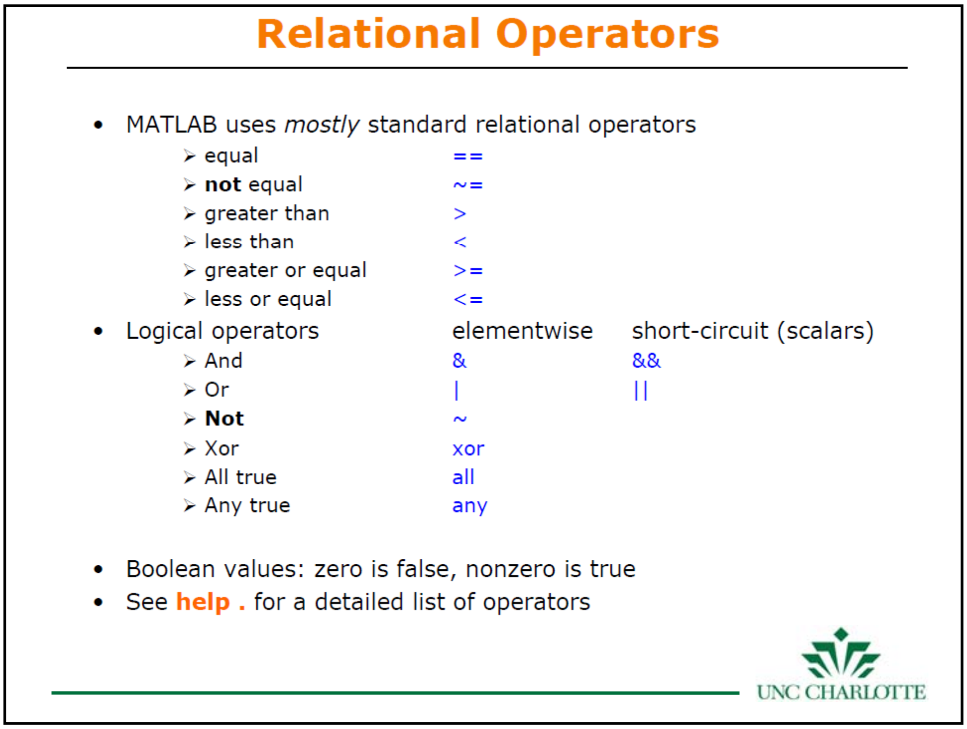
- Write a function with the following declaration: `function plotSin(f1)`
- In the function, plot a sin wave with frequency f1, on the range [0,2π]: $\sin(f_1 x)$
- To get good sampling, use 16 points per period.
- In an MATLAB file saved as plotSin.m, write the following:

```
» function plotSin(f1)

  x=linspace(0,2*pi,f1*16+1);
  figure
  plot(x,sin(f1*x))
```

UNC CHARLOTTE

Program Flow Control

UNC CHARLOTTE



Relational Operators

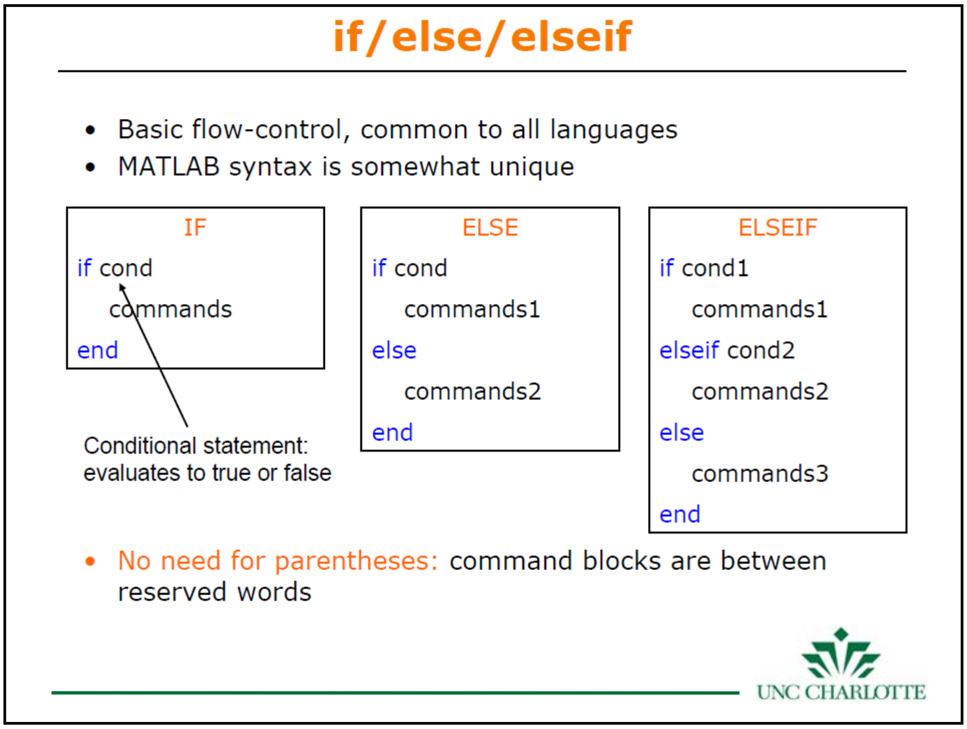
- MATLAB uses *mostly* standard relational operators

➢ equal	==
➢ **not** equal	~=
➢ greater than	>
➢ less than	<
➢ greater or equal	>=
➢ less or equal	<=

• Logical operators	elementwise	short-circuit (scalars)
➢ And	&	&&
➢ Or	|	||
➢ **Not**	~	
➢ Xor	xor	
➢ All true	all	
➢ Any true	any	

- Boolean values: zero is false, nonzero is true
- See **help .** for a detailed list of operators

UNC CHARLOTTE



if/else/elseif

- Basic flow-control, common to all languages
- MATLAB syntax is somewhat unique

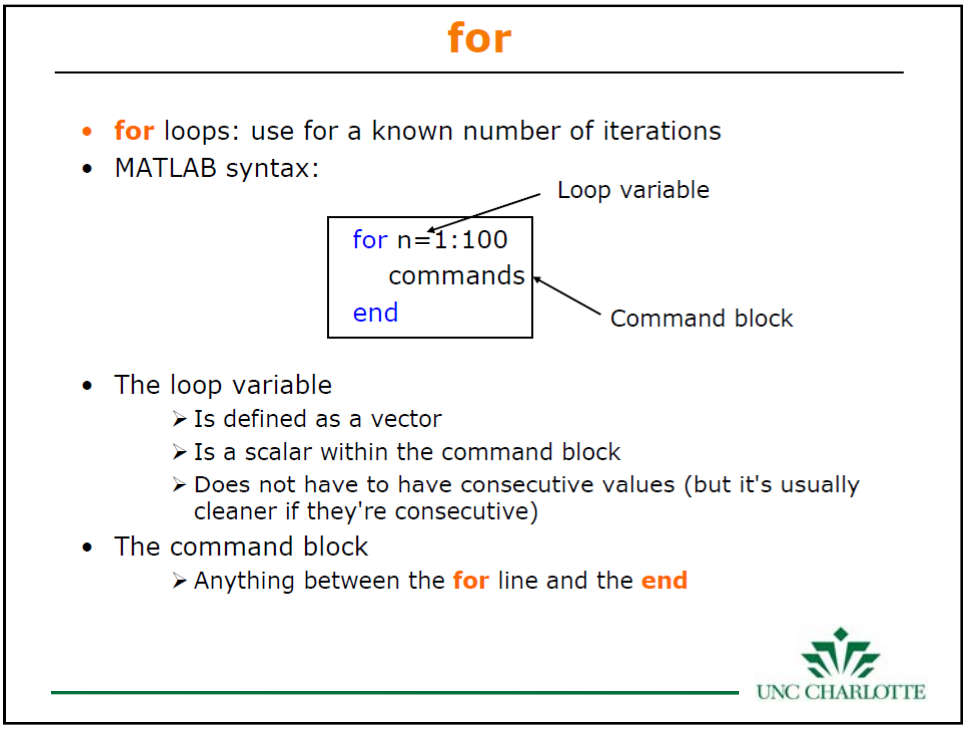
IF
```
if cond
    commands
end
```

Conditional statement: evaluates to true or false

ELSE
```
if cond
    commands1
else
    commands2
end
```

ELSEIF
```
if cond1
    commands1
elseif cond2
    commands2
else
    commands3
end
```

- No need for parentheses: command blocks are between reserved words



for

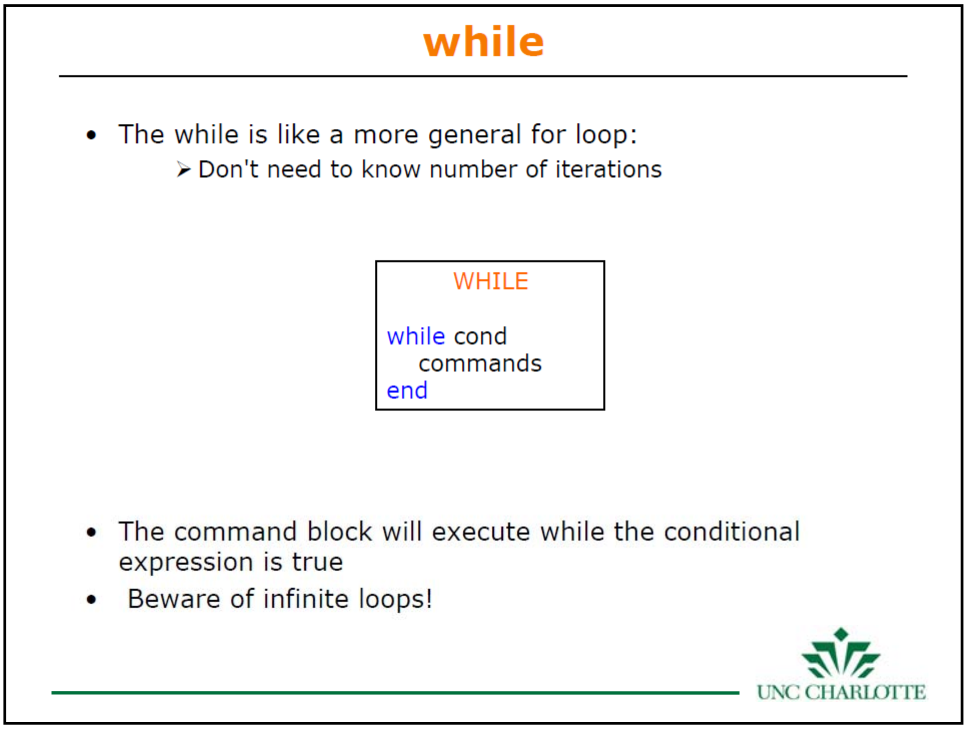
- **for** loops: use for a known number of iterations
- MATLAB syntax:

```
for n=1:100
    commands
end
```

Loop variable

Command block

- The loop variable
  - Is defined as a vector
  - Is a scalar within the command block
  - Does not have to have consecutive values (but it's usually cleaner if they're consecutive)
- The command block
  - Anything between the **for** line and the **end**

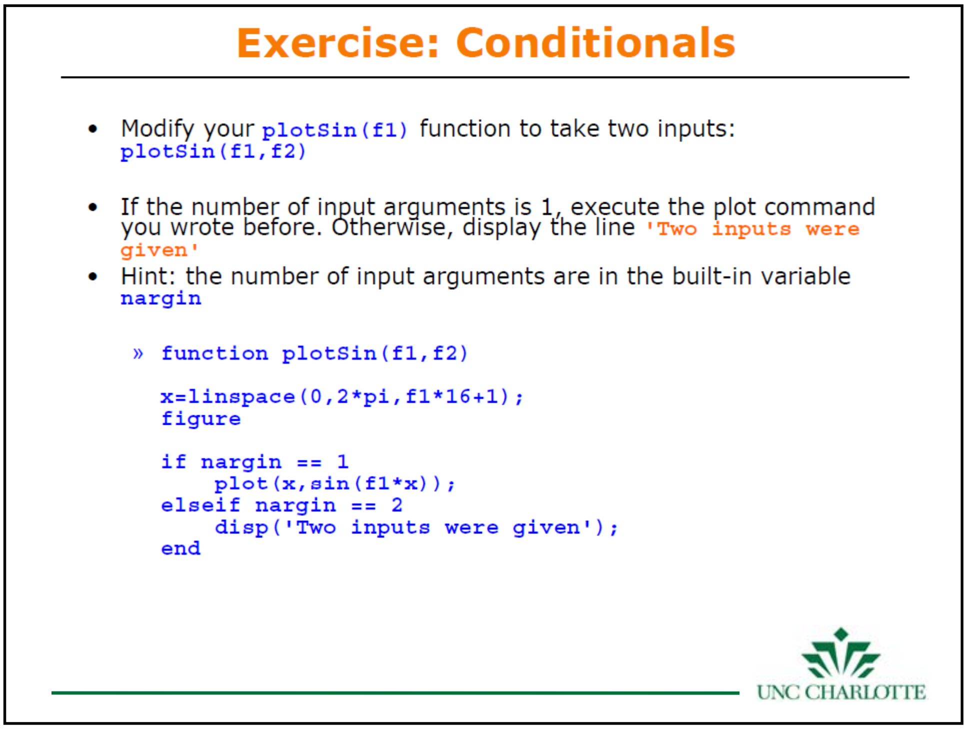


while

- The while is like a more general for loop:
  - ➢ Don't need to know number of iterations

```
WHILE

while cond
   commands
end
```

- The command block will execute while the conditional expression is true
- Beware of infinite loops!

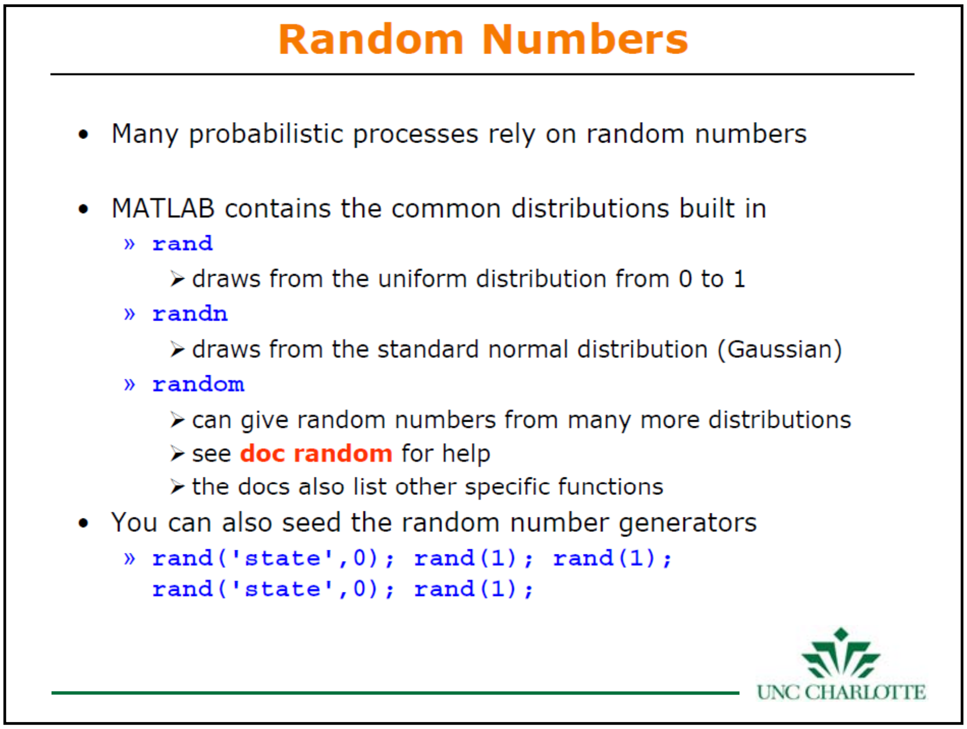


Exercise: Conditionals

- Modify your `plotSin(f1)` function to take two inputs: `plotSin(f1,f2)`
- If the number of input arguments is 1, execute the plot command you wrote before. Otherwise, display the line `'Two inputs were given'`
- Hint: the number of input arguments are in the built-in variable `nargin`

```
» function plotSin(f1,f2)

  x=linspace(0,2*pi,f1*16+1);
  figure

  if nargin == 1
      plot(x,sin(f1*x));
  elseif nargin == 2
      disp('Two inputs were given');
  end
```

Random Numbers & Basic Statistics

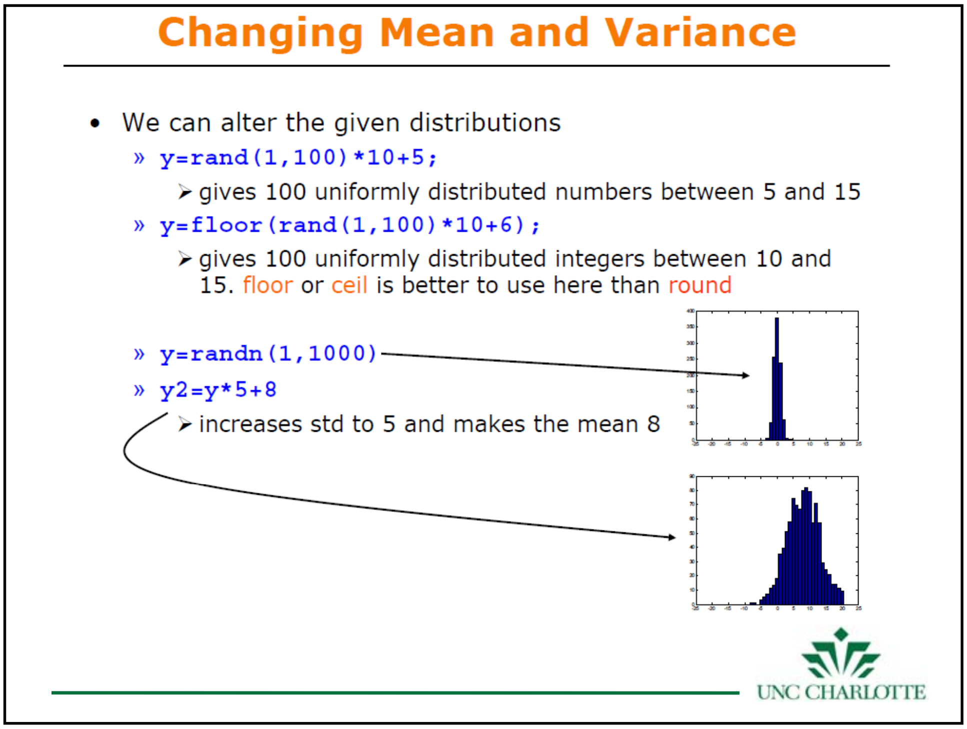


Random Numbers

- Many probabilistic processes rely on random numbers

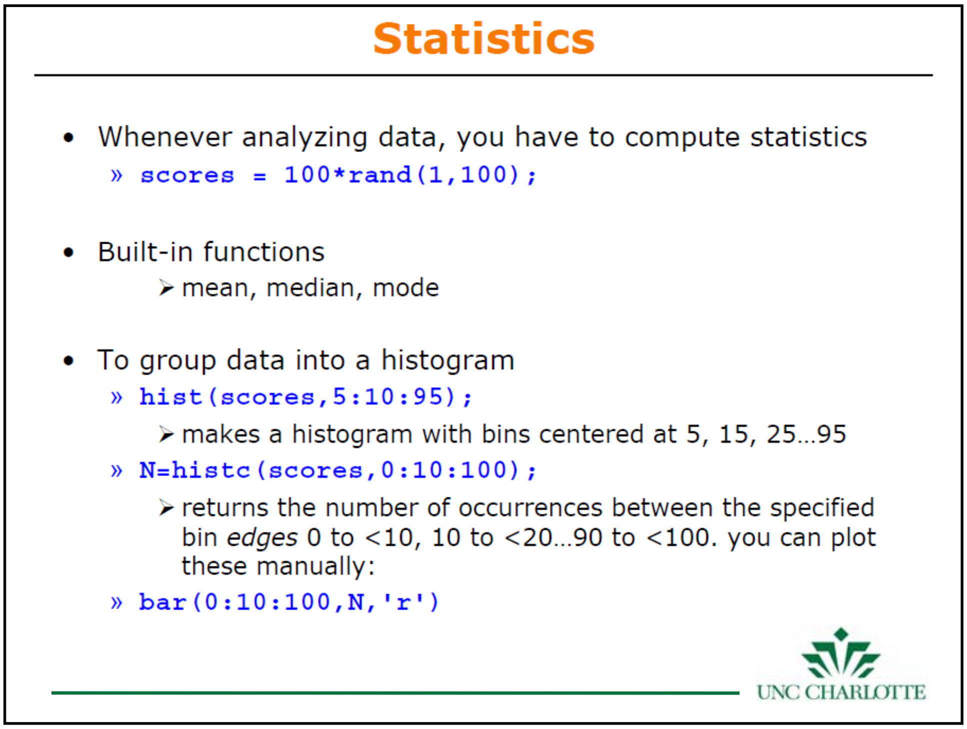
- MATLAB contains the common distributions built in
  - » `rand`
    - ➢ draws from the uniform distribution from 0 to 1
  - » `randn`
    - ➢ draws from the standard normal distribution (Gaussian)
  - » `random`
    - ➢ can give random numbers from many more distributions
    - ➢ see **doc random** for help
    - ➢ the docs also list other specific functions
- You can also seed the random number generators

```
» rand('state',0); rand(1); rand(1);
  rand('state',0); rand(1);
```



Changing Mean and Variance

- We can alter the given distributions
  - » `y=rand(1,100)*10+5;`
    - ➢ gives 100 uniformly distributed numbers between 5 and 15
  - » `y=floor(rand(1,100)*10+6);`
    - ➢ gives 100 uniformly distributed integers between 10 and 15. floor or ceil is better to use here than round
  - » `y=randn(1,1000)`
  - » `y2=y*5+8`
    - ➢ increases std to 5 and makes the mean 8



Statistics

- Whenever analyzing data, you have to compute statistics
  - » `scores = 100*rand(1,100);`
- Built-in functions
  - ➢ mean, median, mode
- To group data into a histogram
  - » `hist(scores,5:10:95);`
    - ➢ makes a histogram with bins centered at 5, 15, 25…95
  - » `N=histc(scores,0:10:100);`
    - ➢ returns the number of occurrences between the specified bin *edges* 0 to <10, 10 to <20…90 to <100. you can plot these manually:
  - » `bar(0:10:100,N,'r')`

Exercise: Probability

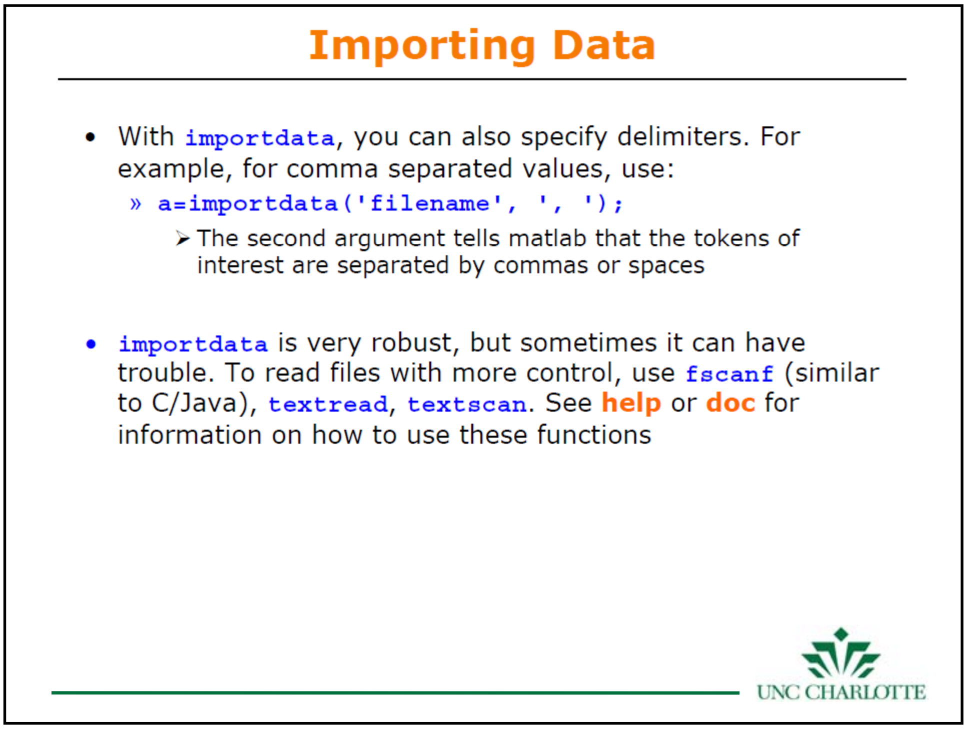
- We will simulate Brownian motion in 1 dimension. Call the script 'brown'
- Make a 10,000 element vector of zeros
- Write a loop to keep track of the particle's position at each time
- Start at 0. To get the new position, pick a random number, and if it's <0.5, go left; if it's >0.5, go right. Store each new position in the kth position in the vector
- Plot a 50 bin histogram of the positions.

```
» x=zeros(10000,1);
» for n=2:10000
»     if rand<0.5
»         x(n)=x(n-1)-1;
»     else
»         x(n)=x(n-1)+1;
»     end
» end
» figure;
» hist(x,50);
```

UNC CHARLOTTE

I/O Operations

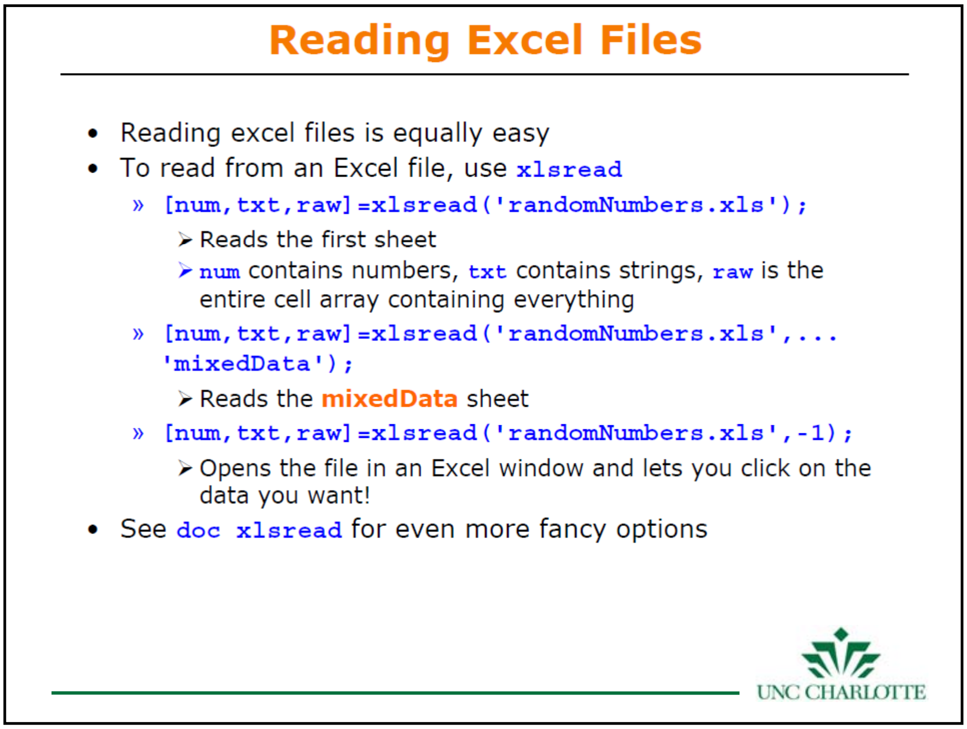
UNC CHARLOTTE



Importing Data

- With `importdata`, you can also specify delimiters. For example, for comma separated values, use:
  - » `a=importdata('filename', ', ');`
    - ➢ The second argument tells matlab that the tokens of interest are separated by commas or spaces
- `importdata` is very robust, but sometimes it can have trouble. To read files with more control, use `fscanf` (similar to C/Java), `textread`, `textscan`. See **help** or **doc** for information on how to use these functions

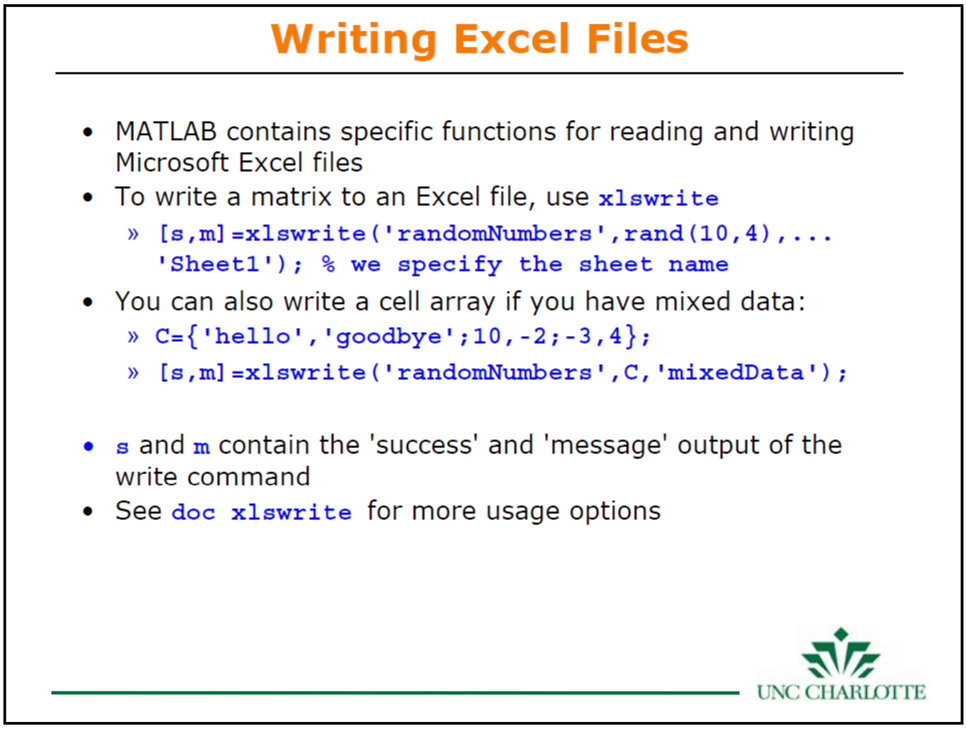
UNC CHARLOTTE



Reading Excel Files

- Reading excel files is equally easy
- To read from an Excel file, use `xlsread`
  - » `[num,txt,raw]=xlsread('randomNumbers.xls');`
    - ➢ Reads the first sheet
    - ➢ `num` contains numbers, `txt` contains strings, `raw` is the entire cell array containing everything
  - » `[num,txt,raw]=xlsread('randomNumbers.xls',... 'mixedData');`
    - ➢ Reads the **mixedData** sheet
  - » `[num,txt,raw]=xlsread('randomNumbers.xls',-1);`
    - ➢ Opens the file in an Excel window and lets you click on the data you want!
- See `doc xlsread` for even more fancy options

UNC CHARLOTTE



Writing Excel Files

- MATLAB contains specific functions for reading and writing Microsoft Excel files
- To write a matrix to an Excel file, use `xlswrite`

```
» [s,m]=xlswrite('randomNumbers',rand(10,4),...
  'Sheet1'); % we specify the sheet name
```

- You can also write a cell array if you have mixed data:

```
» C={'hello','goodbye';10,-2;-3,4};
» [s,m]=xlswrite('randomNumbers',C,'mixedData');
```

- `s` and `m` contain the 'success' and 'message' output of the write command
- See `doc xlswrite` for more usage options

UNC CHARLOTTE