

# OPTIMIZATION

## Laboratory class

---

Rosario Maggistro

- E-mail address: `rosario.maggistro@deams.units.it`
- Material adapted from MIT course #6.094:  
<http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-094-introduction-to-matlab-january-iap-2010/lecture-notes/>

---

# Introduction to Programming in MATLAB

---

## **Lecture 1: Variables, Scripts, and Operations**

# Outline

---

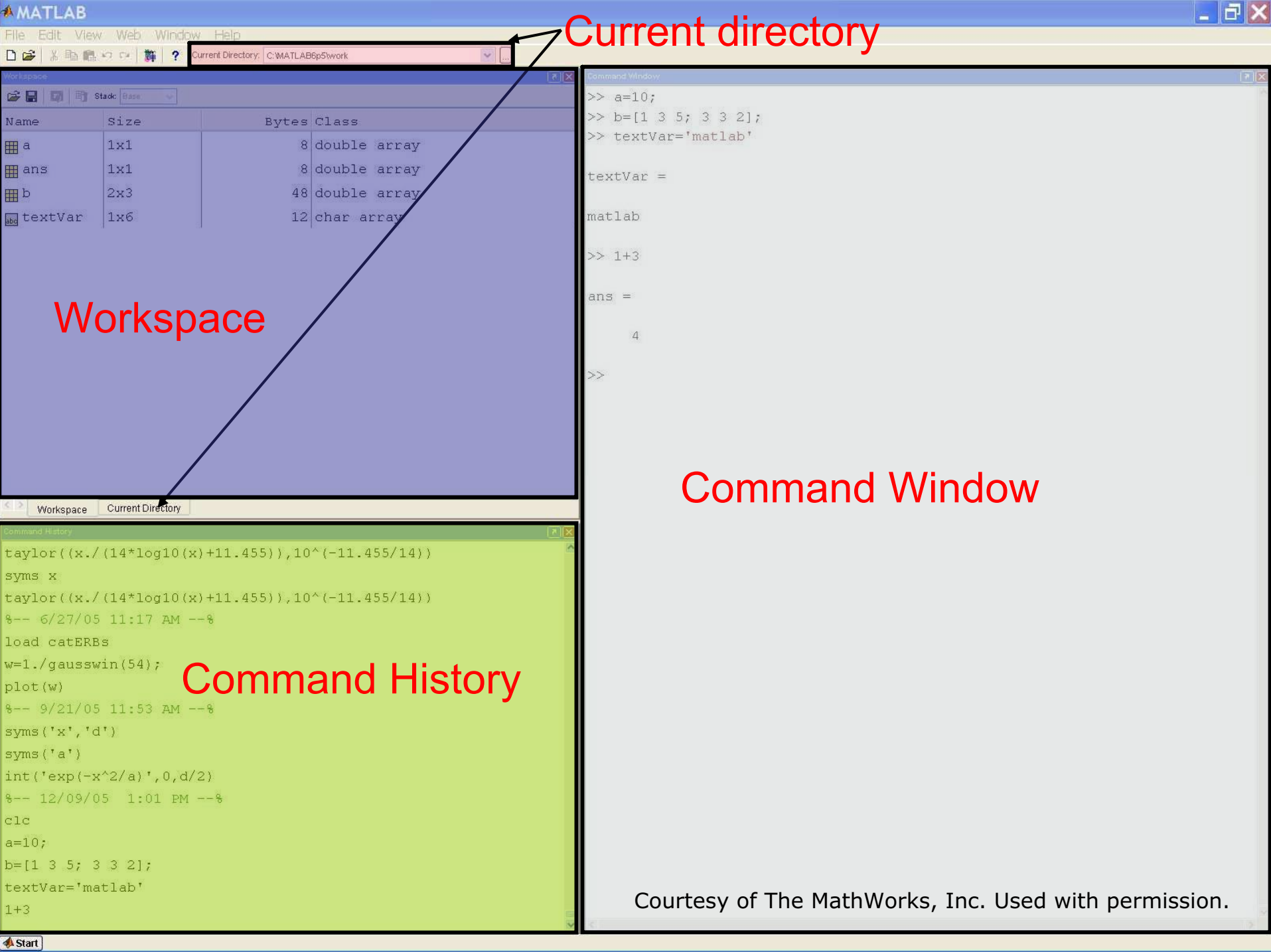
**(1) Getting Started**

(2) Scripts

(3) Making Variables

(4) Manipulating Variables

(5) Basic Plotting



Current directory

Workspace

Command Window

Command History

Courtesy of The MathWorks, Inc. Used with permission.

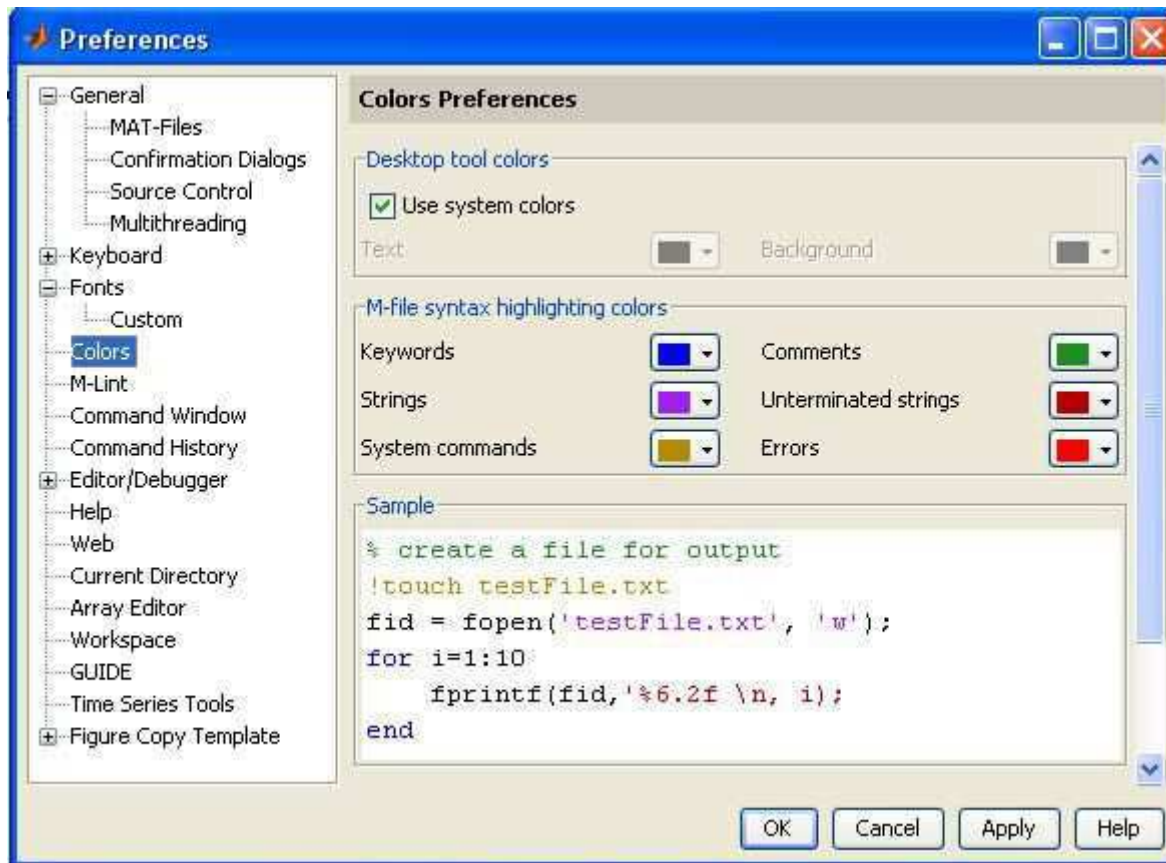
# Making Folders

---

- Use folders to keep your programs organized
- To make a new folder, click the 'Browse' button next to 'Current Directory'
- Click the 'Make New Folder' button, and change the name of the folder. **Do NOT use spaces** in folder names. In the MATLAB folder, make two new folders: `MATLAB\lab1`
- Highlight the folder you just made and click 'OK'
- The current directory is now the folder you just created
- To see programs outside the current directory, they should be in the Path. Use File-> Set Path to add folders to the path

# Customization

- File → Preferences
  - Allows you personalize your MATLAB experience



Courtesy of The MathWorks, Inc. Used with permission.

# 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

# 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



# Outline

---

(1) Getting Started

**(2) Scripts**

(3) Making Variables

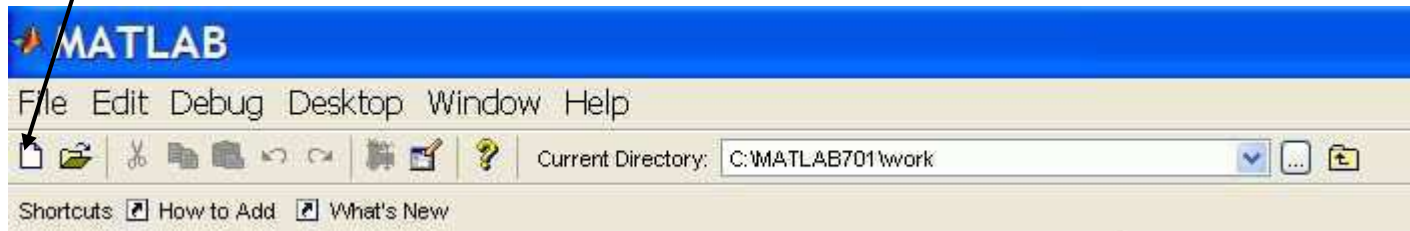
(4) Manipulating Variables

(5) Basic Plotting

# Scripts: Overview

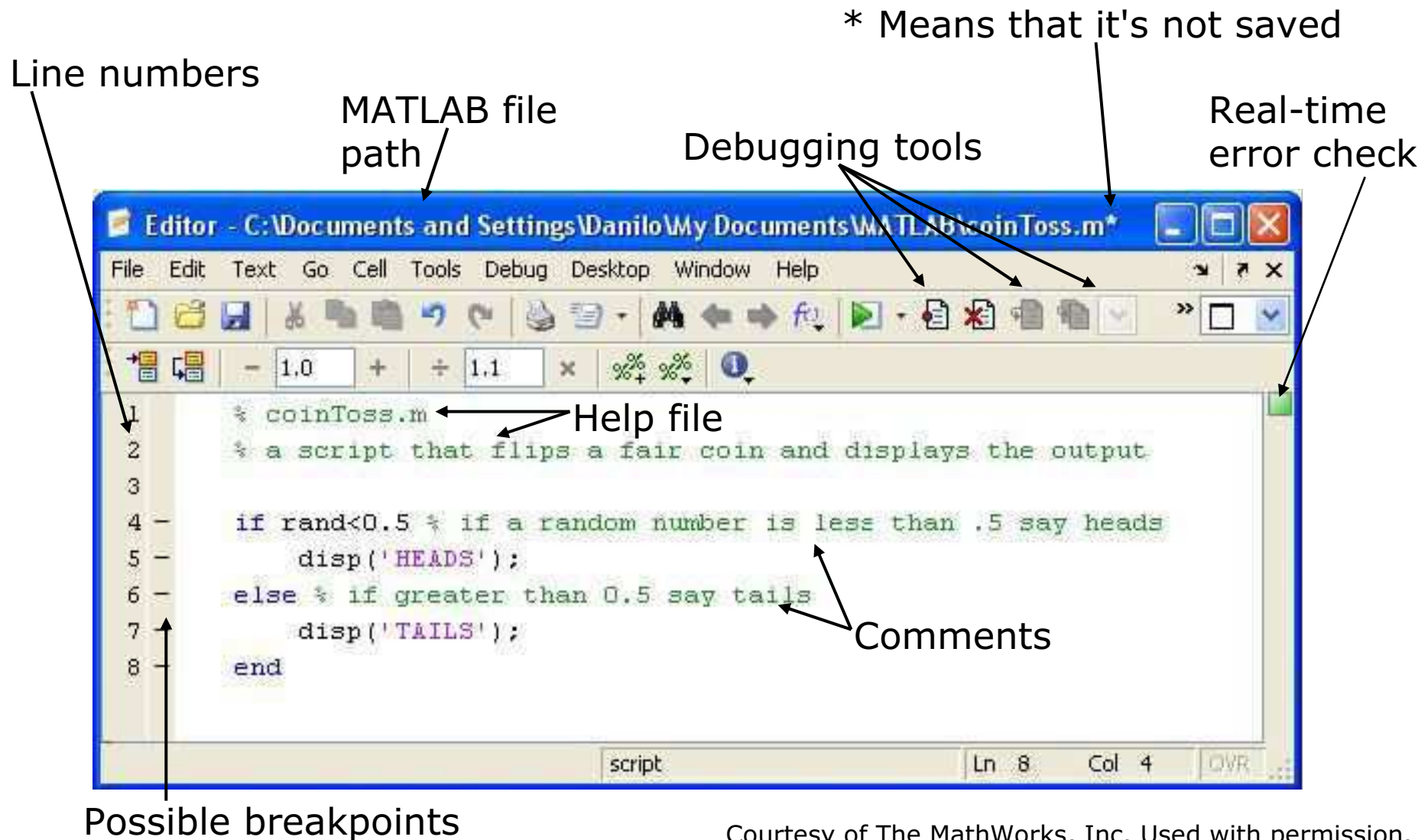
---

- 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



Courtesy of The MathWorks, Inc. Used with permission.

# Scripts: the Editor



Courtesy of The MathWorks, Inc. Used with permission.

# 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

# 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'`

# 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!');`

# Outline

---

(1) Getting Started

(2) Scripts

**(3) Making Variables**

(4) Manipulating Variables

(5) Basic Plotting

# Variable Types

---

- MATLAB is a weakly typed language
  - No need to initialize variables!
- MATLAB supports various types, the most often used are
  - » `3.84`
    - 64-bit double (default)
  - » `'a'`
    - 16-bit char
- Most variables you'll deal with will be vectors or matrices of doubles or chars
- Other types are also supported: complex, symbolic, 16-bit and 8 bit integers, etc. You will be exposed to all these types through the homework



# 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'

# 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;`

# 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)

**MATLAB makes vectors easy!  
That's its power!**



# Row Vectors

---

- Row vector: comma or space separated values between brackets

```
» row = [1 2 5.4 -6.6]
```

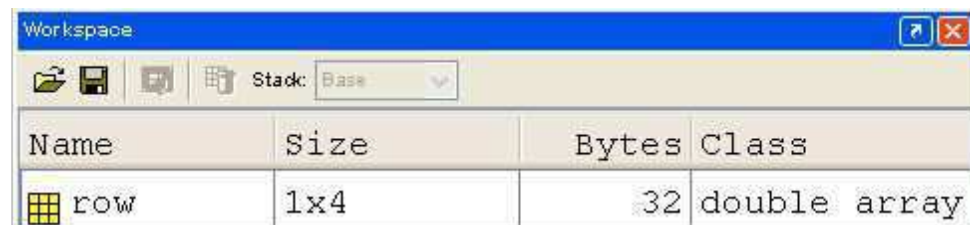
```
» row = [1, 2, 5.4, -6.6];
```

- Command window: `>> row=[1 2 5.4 -6.6]`

```
row =
```

```
1.0000    2.0000    5.4000   -6.6000
```

- Workspace:



The image shows a screenshot of the MATLAB Workspace window. The window has a blue title bar with the text 'Workspace'. Below the title bar is a toolbar with icons for saving, deleting, and other workspace actions. A dropdown menu labeled 'Stack' is set to 'Base'. The main area of the window is a table with four columns: 'Name', 'Size', 'Bytes', and 'Class'. There is one row in the table representing the variable 'row'.

Name	Size	Bytes	Class
row	1x4	32	double array

# 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:



The image shows a screenshot of the MATLAB Workspace window. It has a title bar 'Workspace' and a toolbar with icons for saving, deleting, and refreshing, along with a 'Stack' dropdown menu set to 'Base'. Below the toolbar is a table with four columns: 'Name', 'Size', 'Bytes', and 'Class'. The table contains one entry for the variable 'column', which is a 4x1 double array, occupying 32 bytes.

Name	Size	Bytes	Class
column	4x1	32	double array

# 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)
```

```
ans =
```

```
1    4
```

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

```
ans =
```

```
4    1
```

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

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

```
ans =
```

```
4
```

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

```
ans =
```

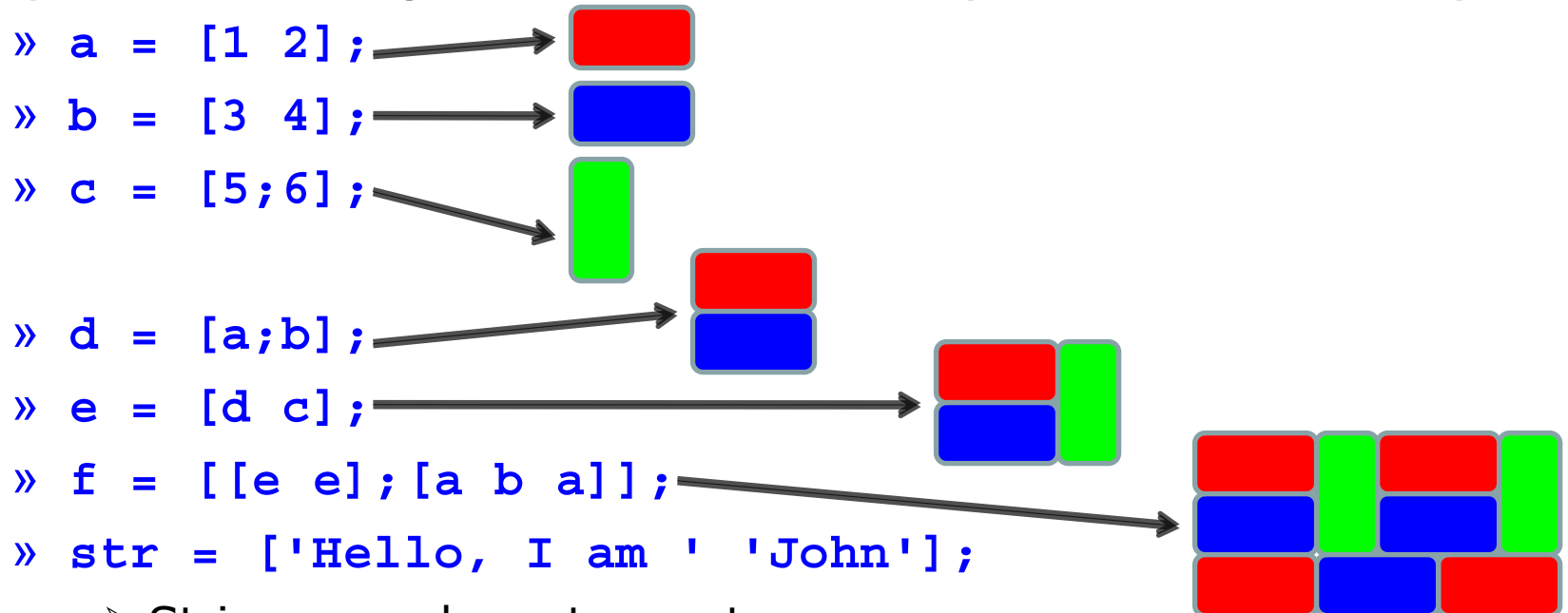
```
4
```

# 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)



➤ 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\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;`



# Exercise: Variables

---

## Get and save the current date and time

- Create a variable `start` using the function `clock`
- What is the size of `start`? Is it a row or column?
- What does `start` contain? See `help clock`
- Convert the vector `start` to a string. Use the function `datestr` and name the new variable `startString`
- Save `start` and `startString` into a mat file named `startTime`

# Exercise: Variables

---

## Get and save the current date and time

- Create a variable `start` using the function `clock`
- What is the size of `start`? Is it a row or column?
- What does `start` contain? See `help clock`
- Convert the vector `start` to a string. Use the function `datestr` and name the new variable `startString`
- Save `start` and `startString` into a mat file named `startTime`

```
» help clock
```

```
» start=clock;
```

```
» size(start)
```

```
» help datestr
```

```
» startString=datestr(start);
```

```
» save startTime start startString
```

# Outline

---

- (1) Getting Started
- (2) Scripts
- (3) Making Variables
- (4) Manipulating Variables**
- (5) Basic Plotting

# 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

# 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);`

# Exercise: Scalars

---

**You will learn MATLAB at an exponential rate! Add the following to your helloWorld script:**

- Your learning time constant is **1.5 days**. Calculate the number of **seconds** in 1.5 days and name this variable **tau**
- This class lasts 5 days. Calculate the number of seconds in 5 days and name this variable **endOfClass**
- This equation describes your knowledge as a function of time t:

$$k = 1 - e^{-t/\tau}$$

- How well will you know MATLAB at **endOfClass**? Name this variable **knowledgeAtEnd**. (use **exp**)
- Using the value of **knowledgeAtEnd**, display the phrase:

At the end of 6.094, I will know X% of MATLAB

- **Hint:** to convert a number to a string, use **num2str**

# Exercise: Scalars

---

```
» secPerDay=60*60*24;  
» tau=1.5*secPerDay;  
» endOfClass=5*secPerDay  
» knowledgeAtEnd=1-exp(-endOfClass/tau);  
» disp(['At the end of the course, I will know ' ...  
    num2str(knowledgeAtEnd*100) '% of MATLAB'])
```

# 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



# Addition and Subtraction

---

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

$$\begin{array}{r} [12 \quad 3 \quad 32 \quad -11] \\ + [2 \quad 11 \quad -30 \quad 32] \\ \hline = [14 \quad 14 \quad 2 \quad 21] \end{array}$$

$$\begin{bmatrix} 12 \\ 1 \\ -10 \\ 0 \end{bmatrix} - \begin{bmatrix} 3 \\ -1 \\ 13 \\ 33 \end{bmatrix} = \begin{bmatrix} 9 \\ 2 \\ -23 \\ -33 \end{bmatrix}$$

- 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) ;`

# 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

# 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

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

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

$$3 \times 1 .* 3 \times 1 = 3 \times 1$$

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

$$3 \times 3 .* 3 \times 3 = 3 \times 3$$

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

*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 ( $/$   $\backslash$ ) is same as multiplying by inverse
  - Our recommendation: just multiply by inverse (more on this later)

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

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

*Must be square to do powers*

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

# Automatic Initialization

---

- Initialize a vector of **ones**, **zeros**, or **random** 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

# Automatic Initialization

---

- To initialize a linear vector of values use **linspace**
  - » `a=linspace(0,10,5)`
    - starts at 0, ends at 10 (inclusive), 5 values
- Can also use colon operator (:)
  - » `b=0:2:10`
    - starts at 0, increments by 2, and ends at or before 10
    - increment can be decimal or negative
  - » `c=1:5`
    - if increment isn't specified, default is 1
- To initialize logarithmically spaced values use **logspace**
  - similar to **linspace**, but see **help**

# Exercise: Vector Functions

---

## Calculate your learning trajectory

- In helloWorld.m, make a linear time vector `tVec` that has 10,000 samples between 0 and `endOfClass`
- Calculate the value of your knowledge (call it `knowledgeVec`) at each of these time points using the same equation as before:

$$k = 1 - e^{-t/\tau}$$

# Exercise: Vector Functions

---

## Calculate your learning trajectory

- In helloWorld.m, make a linear time vector `tVec` that has 10,000 samples between 0 and `endOfClass`
- Calculate the value of your knowledge (call it `knowledgeVec`) at each of these time points using the same equation as before:

$$k = 1 - e^{-t/\tau}$$

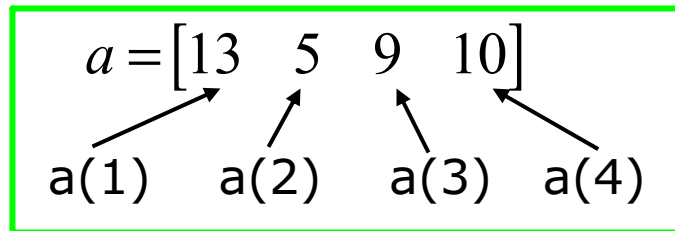
```
» tVec = linspace(0,endOfClass,10000);  
» knowledgeVec=1-exp(-tVec/tau);
```



# 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  $n^{\text{th}}$  element

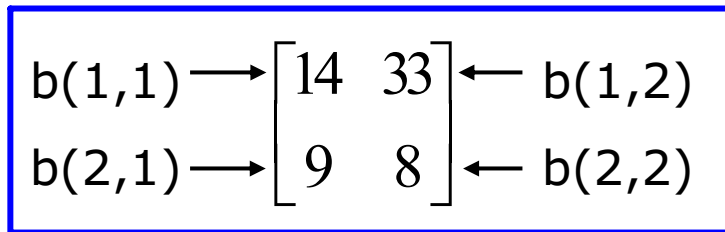


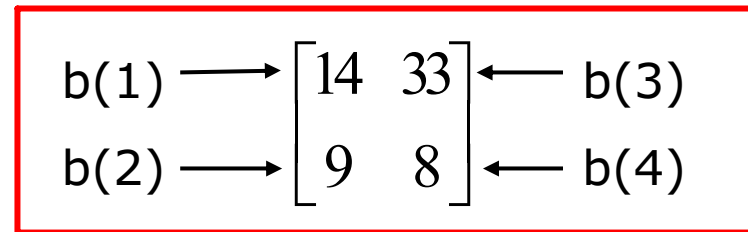
- The index argument can be a vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector.
  - »  $x = [12 \ 13 \ 5 \ 8];$
  - »  $a = x(2:3);$  —————→  $a = [13 \ 5];$
  - »  $b = x(1:end-1);$  —————→  $b = [12 \ 13 \ 5];$

# 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**


$$\begin{array}{lcl} b(1,1) \longrightarrow & \begin{bmatrix} 14 & 33 \end{bmatrix} & \longleftarrow b(1,2) \\ b(2,1) \longrightarrow & \begin{bmatrix} 9 & 8 \end{bmatrix} & \longleftarrow b(2,2) \end{array}$$


$$\begin{array}{lcl} b(1) \longrightarrow & \begin{bmatrix} 14 & 33 \end{bmatrix} & \longleftarrow b(3) \\ b(2) \longrightarrow & \begin{bmatrix} 9 & 8 \end{bmatrix} & \longleftarrow b(4) \end{array}$$

- 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.

# Exercise: Indexing

---

## When will you know 50% of MATLAB?

- First, find the index where `knowledgeVec` is closest to 0.5. Mathematically, what you want is the index where the value of  $|knowledgeVec - 0.5|$  is at a minimum (use `abs` and `min`).
- Next, use that index to look up the corresponding time in `tVec` and name this time `halfTime`.
- Finally, display the string: I will know half of MATLAB after X days  
Convert `halfTime` to days by using `secPerDay`

# Exercise: Indexing

---

## When will you know 50% of MATLAB?

- First, find the index where `knowledgeVec` is closest to 0.5. Mathematically, what you want is the index where the value of  $|knowledgeVec - 0.5|$  is at a minimum (use `abs` and `min`).
- Next, use that index to look up the corresponding time in `tVec` and name this time `halfTime`.
- Finally, display the string: I will know half of MATLAB after X days  
Convert `halfTime` to days by using `secPerDay`

```
» [val, ind] = min(abs(knowledgeVec - 0.5));  
» halfTime = tVec(ind);  
» disp(['I will know half of MATLAB after ' ...  
    num2str(halfTime/secPerDay) ' days']);
```

# Outline

---

- (1) Getting Started
- (2) Scripts
- (3) Making Variables
- (4) Manipulating Variables
- (5) Basic Plotting**

# 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);`

**MATLAB makes visualizing data  
fun and easy!**

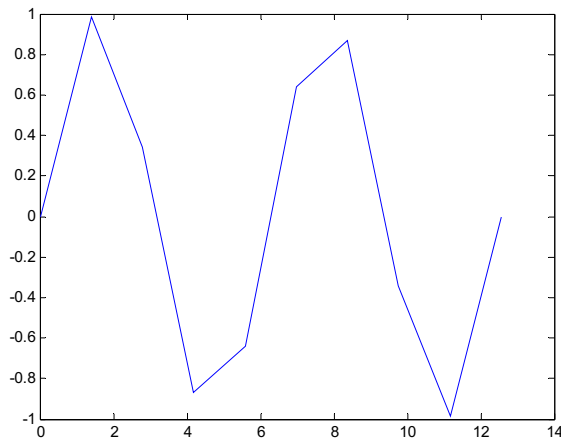




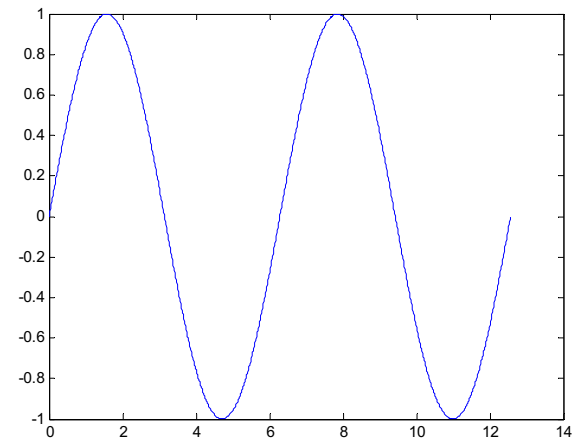
# What does plot do?

- **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:



# Exercise: Plotting

---

## Plot the learning trajectory

- In helloWorld.m, open a new figure (use `figure`)
- Plot the knowledge trajectory using `tVec` and `knowledgeVec`. When plotting, convert `tVec` to days by using `secPerDay=24*60*60`
- Zoom in on the plot to verify that `halfTime` was calculated correctly

# Exercise: Plotting

---

## Plot the learning trajectory

- In helloWorld.m, open a new figure (use `figure`)
- Plot the knowledge trajectory using `tVec` and `knowledgeVec`. When plotting, convert `tVec` to days by using `secPerDay=24*60*60`
- Zoom in on the plot to verify that `halfTime` was calculated correctly

```
» figure
```

```
» secPerDay = 24*60*60;
```

```
» plot(tVec/secPerDay, knowledgeVec);
```

# End of Lecture 1

---

- (1) Getting Started
- (2) Scripts
- (3) Making Variables
- (4) Manipulating Variables
- (5) Basic Plotting

Hope that wasn't too much!!

