Note about functions in files

- Whenever possible, write your functions in their own files
  - ➢ e.g. myfun should be in a file by itself, and the file should be called myfun.m*
  - ➢ If you include more than one function per file, only the first function is accessible in other scripts
  - ➢ More info here: https://www.mathworks.com/help/matlab/matlab_prog/create-functions-in-files.html

* If filename and function name differs, MATLAB recognizes your function by its filename**, not the function name

** yes, this is very confusing :(

Outline

(1) **Probability and Statistics**

(2) Data Structures

(3) Images

(4) File I/O

Statistics

- Whenever analyzing data, you have to compute statistics

```
» scores = 100*rand(1,100);   % random data
```

- 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

```
» hist(scores,20);
```

  - ➢ makes a histogram with 20 bins

```
» 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')
```

Random Numbers

- Many probabilistic processes rely on random numbers

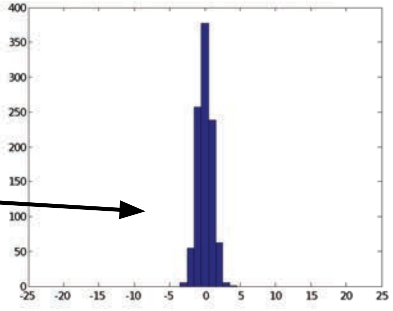
- MATLAB contains the common distributions built in
  - » `rand`
    - ➢ draws from the uniform distribution from 0 to 1
  - » `randn`
    - ➢ draws from the standard normal distribution (Gaussian)
  - » `random`
    - ➢ can give random numbers from many more distributions
    - ➢ see **help random**

- You can also seed the random number generators

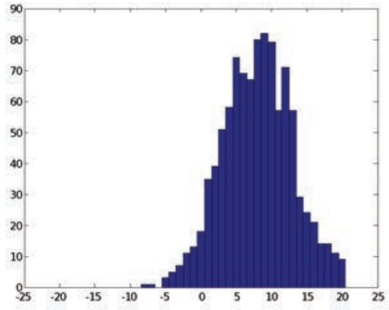
```
» rand('state',0); rand(1); rand(1);
  rand('state',0); rand(1);  % same random number
```

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 6 and 15. floor or ceil is better to use here than round
    - ➢ you can also use randi([6,15],1,100)



  - » y=randn(1,1000)
  - » y2=y*5+8
    - ➢ increases std to 5 and makes the mean 8



Exercise: Probability

- We will simulate Brownian motion in 1 dimension. Call the script ‘brwn’
- Make a 10,001 element vector of zeros
- Write a loop to keep track of the particle’s position at each time
- Assume middle of the vector is position 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. Keep count of how many times each position is visited.
- Plot a 50 bin histogram of the positions.

Outline

(1) Probability and Statistics

(2) Data Structures

(3) Images

(4) File I/O

Advanced Data Structures

- We have used 2D matrices
  - ➢ Can have n-dimensions (e.g., RGB images)
  - ➢ Every element must be the same type (ex. integers, doubles, characters…)
  - ➢ Matrices are space-efficient and convenient for calculation
  - ➢ Large matrices with many zeros can be made sparse
    - – More on this later this lecture

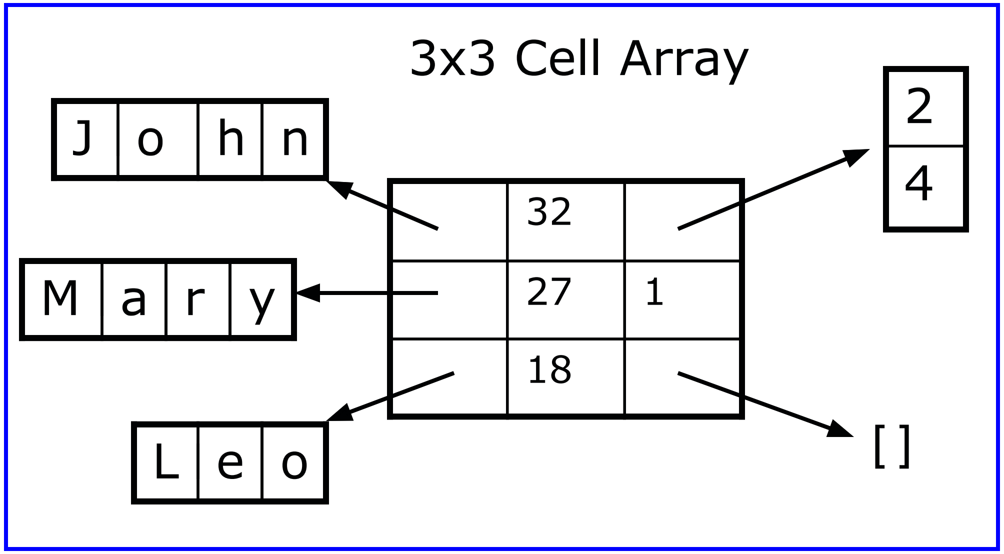
- Sometimes, more complex data structures are more appropriate
  - ➢ **Cell array**: it's like an array, but elements don't have to be the same type
  - ➢ **Structs**: can bundle variable names and values into one structure
    - – Like object oriented programming in MATLAB

Cells: organization

- A cell is just like a matrix, but each field can contain anything (even other matrices):

3x3 Matrix

1.2	-3	5.5
-2.4	15	-10
7.8	-1.1	4



- One cell can contain people's names, ages, and the ages of their children
- To do the same with matrices, you would need 3 variables and padding

Cells: initialization

- To initialize a cell, specify the size

  ```
  » a=cell(3,10);
  ```

  - a will be a cell with 3 rows and 10 columns

- or do it manually, with curly braces {}

  ```
  » c={'hello world',[1 5 6 2],rand(3,2)};
  ```

  - c is a cell with 1 row and 3 columns

- Each element of a cell can be anything

- To access a cell element, use curly braces {}

  ```
  » a{1,1}=[1 3 4 -10];
  » a{2,1}='hello world 2';
  » a{1,2}=c{3};
  ```

Exercise: Cells

- Write a script called `sentGen`
- Make a 2x3 cell, and put three **names** into the first row, and **adjectives** into the second row
- Pick two random integers (values 1 to 3)
- Display a sentence of the form '[name] is [adjective].'
- Run the script a few times

Structs

- Structs allow you to name and bundle relevant variables
    - ➢ Like C-structs, which are containers with fields

- To initialize an empty struct:

```
» s=struct;
```

- ➢ size(s) will be 1x1
- ➢ initialization is optional but is recommended when using large structs

- To add fields

```
» s.name = ‘Leo';
» s.age = 18;
» s.childAge = [];
```

- ➢ Fields can be anything: matrix, cell, even struct
- ➢ Useful for keeping variables together

- For more information, see **help struct**

Struct Arrays

- To initialize a struct array, give field, values pairs

```
» ppl=struct('name',{'John','Mary','Leo'},...
  'age',{32,27,18},'childAge',{[2;4],1,[]});
```

  - ➢ size(ppl)=1x3
  - ➢ every cell must have the same size

```
» person=ppl(2);
```

  - ➢ person is now a struct with fields name, age, children
  - ➢ the values of the fields are the second index into each cell

```
» ppl(3)=s;
```

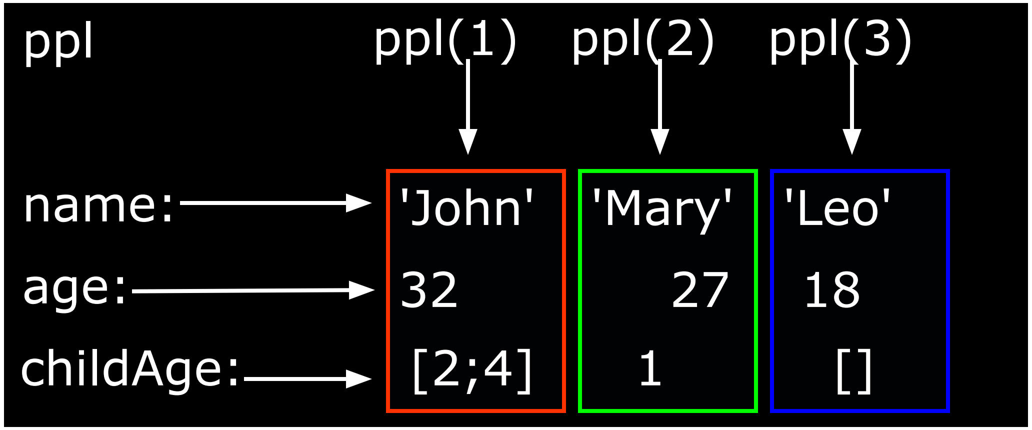
  - ➢ adds struct (fields must match)

```
» person.name
```

  - ➢ returns 'Mary'

```
» ppl(1).age
```

  - ➢ returns 32



Structs: Access

- To access 1x1 struct fields, give name of the field
  - » **stu=s.name;**
  - » **a=s.age;**
    - ➢ 1x1 structs are useful when passing many variables to a function. Put them all in a struct, and pass the struct

- To access nx1 struct arrays, use indices
  - » **person=ppl(2);**
    - ➢ person is a struct with name, age, and child age
  - » **personName=ppl(2).name;**
    - ➢ personName is 'Mary'
  - » **a=[ppl.age];**
    - ➢ a is a 1x3 vector of the ages; this may not always work, the vectors must be able to be concatenated

Exercise: Structs

- Modify the script **`sentGen`**
- Create a struct array with a field “name” and a field “adj” containing the values from the previous cell array
- Do not create it from scratch! Use the previously defined cell array!
- Modify the display command to use the struct array
- Run the script a few times

Outline

(1) Probability and Statistics

(2) Data Structures

(3) Images

(4) File I/O

Handles

- Manipulate graphics objects using ‘handles’

  » `L=plot(1:10,rand(1,10));`

  ➢ gets the handle for the plotted line

  » `A=gca;`

  ➢ gets the handle for the current axis

  » `F=gcf;`

  ➢ gets the handle for the current figure

- To see the current property values, use get

  » `get(L);`

  » `yVals=get(L,'YData');`

- To change the properties, use set

  » `set(A,'FontName','Arial','XScale','log');`

  » `set(L,'LineWidth',1.5,'Marker','*');`

- Everything you see in a figure is completely customizable through handles

Reading/Writing Images

- Images can be imported as a matrix of pixel values

```
» im=imread('myPic.jpg');
» imshow(im);
```

- Matlab supports almost all image formats
  - jpeg, tiff, gif, bmp, png, …
  - see **help imread** for details (e.g., pixel format and types)

- To write an image, give:
  - rgb matrix (0 to 1 doubles, or 0 to 255 uint8)

```
» imwrite(rand(300,300,3),'t1.jpg');
```

  - indices and colormap

```
» imwrite(ceil(rand(200)*256),jet(256),'t2.jpg');
```

  - see **help imwrite** for more options

MATLAB's built-in images

```
           AT3_1m4_01.tif           AT3_1m4_02.tif
           AT3_1m4_03.tif           AT3_1m4_04.tif
           AT3_1m4_05.tif           AT3_1m4_06.tif
           AT3_1m4_07.tif           AT3_1m4_08.tif
           AT3_1m4_09.tif           AT3_1m4_10.tif
               autumn.tif                  bag.png
                blobs.png                board.tif
            cameraman.tif                canoe.tif
                 cell.tif               circbw.tif
              circles.png              circuit.tif
                coins.png        concordaerial.png
    concordorthophoto.png                eight.tif
               fabric.png             football.jpg
               forest.tif          gantrycrane.png
                glass.png               greens.jpg
              hestain.png                 kids.tif
          liftingbody.png                 logo.tif
                  m83.tif                mandi.tif
                 moon.tif                  mri.tif
             office_1.jpg             office_2.jpg
             office_3.jpg             office_4.jpg
             office_5.jpg             office_6.jpg
                onion.png               paper1.tif
                pears.png              peppers.png
             pillsetc.png                 pout.tif
                 rice.png               saturn.png
               shadow.tif           snowflakes.png
                spine.tif                 tape.png
             testpat1.png                 text.png
                 tire.tif               tissue.png
                trees.tif    westconcordaerial.png
westconcordorthophoto.png
```

Load these like you'd load anything else in your current directory:

```
>> load(cameraman.tif');
```

Outline

(1) Probability and Statistics

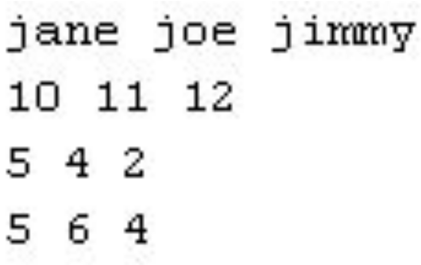
(2) Data Structures

(3) Images

(4) File I/O

Importing Data

- Matlab is a great environment for processing data. If you have a text file with some data:

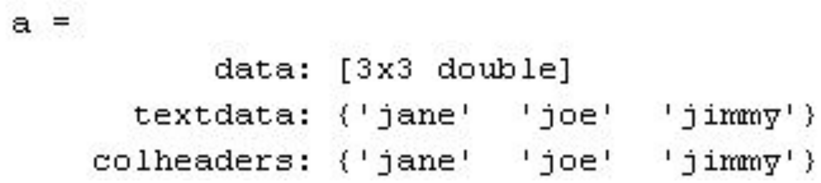


```
jane joe jimmy
10 11 12
5 4 2
5 6 4
```

- To import data from files on your hard drive, use importdata

```
» a=importdata('textFile.txt');
```

  ➢ a is a struct with `data`, `textdata`, and `colheaders` fields



```
a =

         data: [3x3 double]
     textdata: {'jane'   'joe'   'jimmy'}
   colheaders: {'jane'   'joe'   'jimmy'}
```

```
» x=a.data;
» names=a.colheaders;
```

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

- **importdata** is very robust, but sometimes it can have trouble. To read files with more control, use **fscanf** (similar to C/Java), **textscan**. See **help** for information on how to use these functions

Writing Excel Files

- Matlab contains specific functions for reading and writing Microsoft Excel files

- To write a matrix to an Excel file, use **xlswrite**

  ```
  » xlswrite('randomNumbers',rand(10));
  » xlswrite('randomNumbers',rand(10),...
    'Sheet1','C11:L20');
  ```

  ➢ Sheet name and range optional

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

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

- See **help xlswrite** for more usage options

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 **help xlsread** for even more fancy options

Reading ANY File

- You can read any file as binary data

- To read from a file, use fopen
  - **`fid = fopen(‘fileName’, ‘r’);`**
    - Returns a handle to a file
  - **`data = fread(fid, 10);`**
    - Reads the next 10 bytes from the file and stores them in **`data`**
  - **`fseek(fid, 5, 0);`**
    - Moves forward 5 bytes from the current position

- See **help fopen/fread/fwrite/ftell/fseek** for even more fancy options

Lecture 5

- Not mandatory – but highly recommended!

- More cool stuff Matlab has to offer

- Some things we can cover:
  - Animations
  - Build a GUI for your projects!
  - Use cool toolboxes
  - Interact with hardware (scopes, analyzers, Arduino, Raspberry PI, Lego Mindstorm…)
  - Use Simulink to graphically build complex systems and simulate
  - Do image processing
  - Plus… No Homework assignment!

Don’t Forget….

- Comment your code!
- help and Google are your best friends – use them!
- Vectorize whenever possible
- Matlab is powerful but it is not a substitute for your own insights

End of Lecture 4

(1) Probability and Statistics

(2) Data Structures

(3) Images

(4) File I/O

THE END (ALMOST)