EEE 208 – Programming for EEE

Assist. Prof. Dr. Engin Mendi

Cell Arrays

- Numeric arrays (class name is *double*) are rectangular arrays of numbers.
- Character arrays (class name is *char*) are rectangular arrays of characters.
- Cell arrays (class name is *cell*) are rectangular arrays of containers. Their contents could be
  - double arrays
  - character arrays
  - or even cell arrays

Intro to Cell Arrays: Creating

- Use curly brackets { and } to wrap a variable in a container.
- Example: Create a 1-by-1 container with a 1-by-12 character array 'Andy Packard'
- `>> name = { 'Andy Packard' }`
- `>> size(name)`
- `>> class(name)`
- Create two more 1-by-1 containers with different contents
- `>> SID = { 12345678 };`
- `>> scores = { [82 71 64 88 99] };`

Cell Arrays: Concatenation

- Put all three containers next to each other to create a 1-by-3 array of containers.

```
>> ClassInfo = [ name SID scores ];
```

- ClassInfo is a 1-by-3 array of containers. It is called a cell array.
- `>> size(ClassInfo)`
- `>> class(ClassInfo)`
- `>> whos`

Cell Arrays: Class and Size

- ClassInfo is a 1-by-3 array of containers: a cell array.

```
>> size(ClassInfo)
>> class(ClassInfo)
>> whos
```

- Using parenthesis () to access parts of the array keeps the containers (remember, it is an array of containers).

```
>> ClassInfo(1)
>> tmp = 'Andy Packard'
>> isequal(ClassInfo(1), 'Andy Packard')
>> size(ClassInfo(1))
>> class(ClassInfo(1))
```

A review of is* functions

- isnumeric: isnumeric(A) returns logical 1 (true) if A is a numeric array and logical 0 (false) otherwise. For example, single and double variables are numeric, but while strings, cell arrays, structure arrays and logicals are not.

```
isnumeric('by')
```

- isprime: isprime(x) is 1 (true) for the elements of x that are prime, 0 (false) otherwise.
- isempty: isempty(x) returns 1 (true) if x is an empty array and 0 (false) otherwise. An empty array has no elements, that is prod(size(X))==0.
- isequal: isequal(x,y) returns 1 (true) if x is equal to y.

```
isequal(x, 'ch')
```

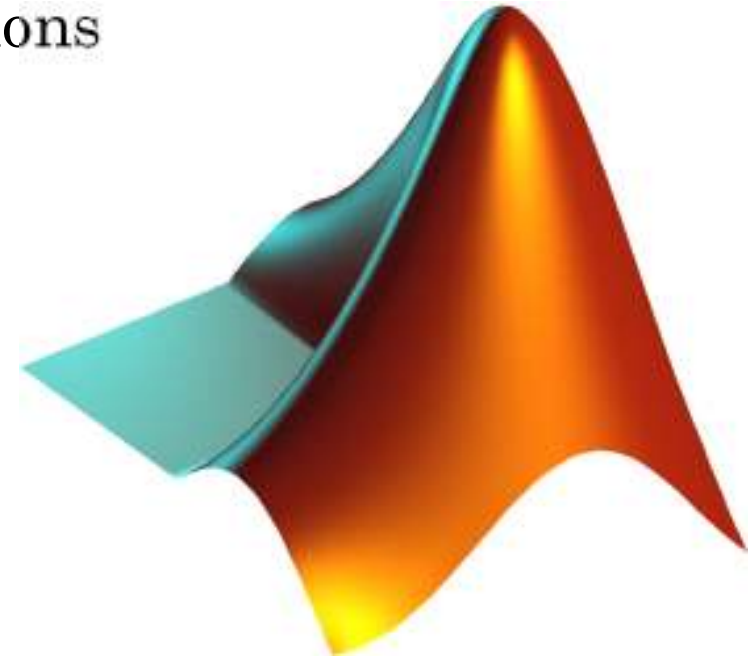
- Input arguments could be scalar, numeric arrays, strings, or vectors of characters

Functions for Cells

- num2cell
- cell2mat
  - C = {[1] [2 3 4]; [5; 9] [6 7 8; 10 11 12]}
  - M = cell2mat(C)
- mat2cell
- celldisp
- iscell

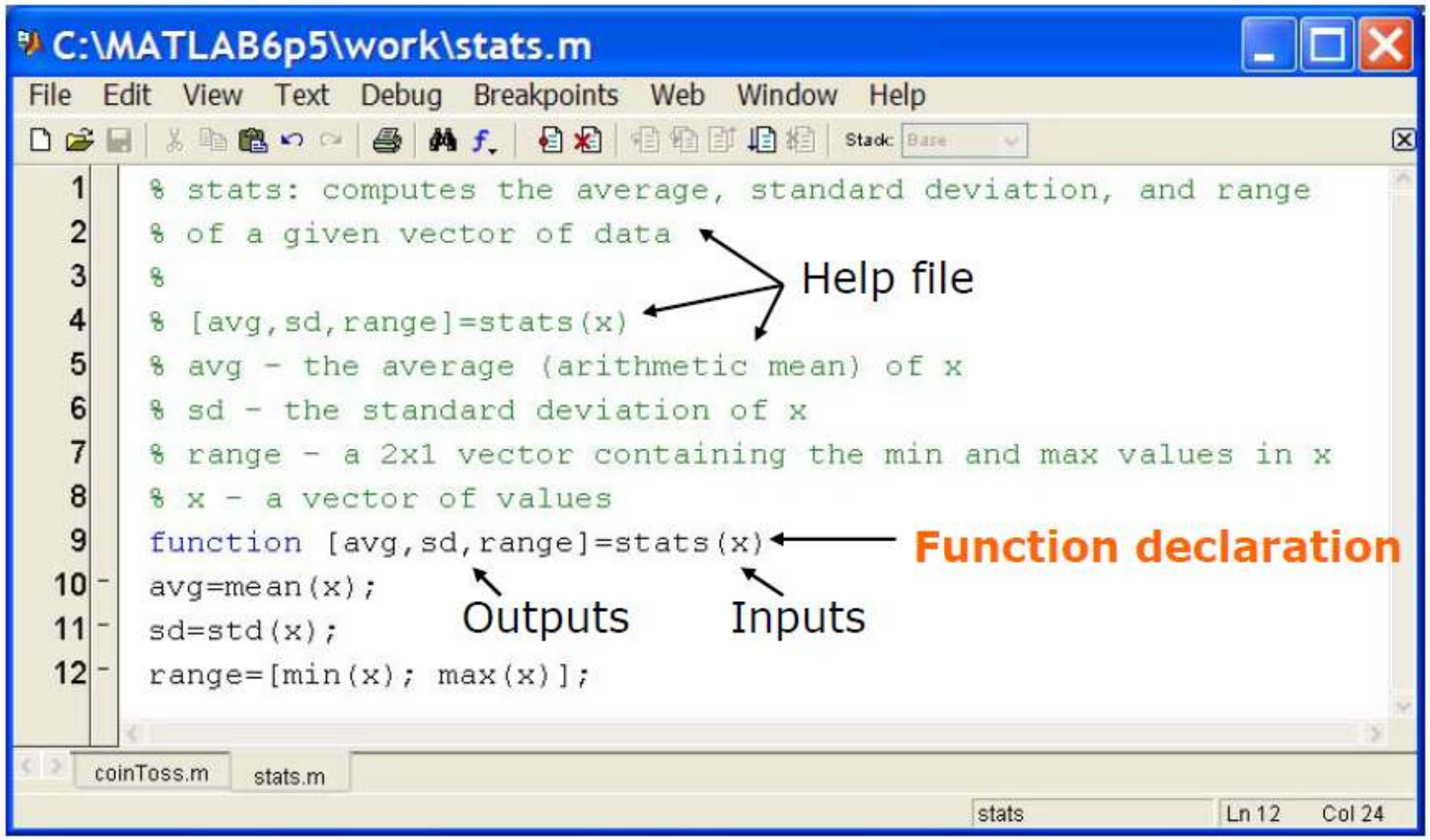
MATLAB Program Files

- Two forms:
  - Scripts (our m-files)
  - Functions
- They help with
  - Automating, Editing/debugging, writing and using codes as applications

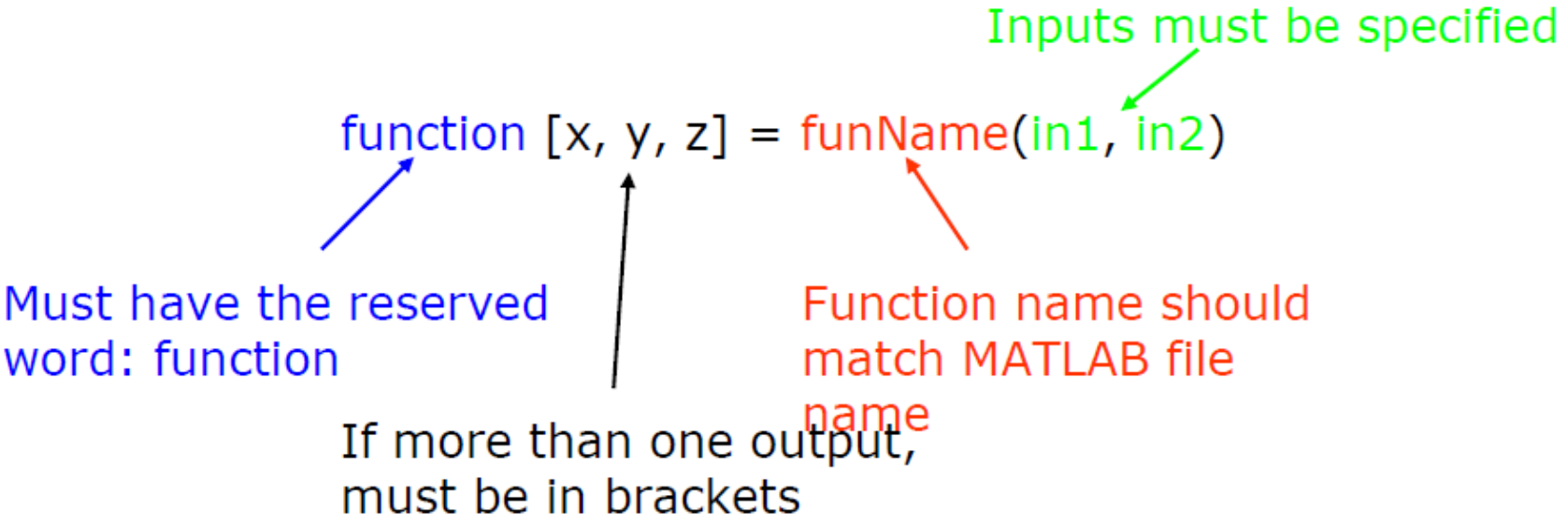


Introduction to Functions

- Functions look exactly like scripts, but with one important difference: You have to define (declare) them in a specific way



User-Defined Functions



Remember:

1. variables used just <u>inside</u> the functions are <u>not</u> shown or saved in the workspace
2. funName should be different from Matlab's predefined functions

Examples of Function Definition Lines

1. One input, one output:

```
function [area_square] = square(side)
```

2. Brackets are optional for one input, one output:

```
function area_square = square(side)
```

3. Two or more inputs, one output:

```
function [volume_box] = box(height,width,length)
```

4. One input, two outputs:

```
function [area_circle,circumf] = circle(radius)
```

5. No named output: `function sqplot(side)`

User-Defined Functions : Example

Example 1: `myfun1.m`

```
>> function y=myfun1(x)
y = cos(4*x).*sin(10*x).*exp(-abs(x));
```

Example 2: `myfun2.m`

```
>> function z = myfun2(x,y)
u = 3*x;
z = u + 6*y.^2;
```

- In this case, x and y could be scalars or vectors. Using .^ makes it possible to work with vectors
- Save it as `myfun2.m` (The file name and function name must exactly be the same)
- Run the function from the command window:
- `t = myfun2(2,5)`
- Or
- `b = myfun2(3, [4 7])`

Rules for Writing Functions 1

1. The word **function** appears as the first word in a function file. This is followed by an **output argument, an equal sign and the function name**. The input arguments come at the end inside parentheses.
2. The information that follows the function, beginning with the % sign, shows how the function is used and what arguments are passed. This information is displayed if help is requested for the function name.
3. MATLAB can accept multiple input arguments and multiple output arguments can be returned.

Rules for Writing Functions 2

4. If a function is going to return more than one value, all the values should be returned as a **vector** in the function statement. For example, function [mean, variance] = data_in(x) will return the mean and variance of a vector x.
5. If a function has multiple input arguments, the function statement must list the input arguments. For example, function [mean, variance] = data(x,n) will return mean and variance of a vector x of length n.
6. If there is more than one function in the code, each of them should finish with an **"end"** statement. Otherwise, no end is needed.

Programming Styles

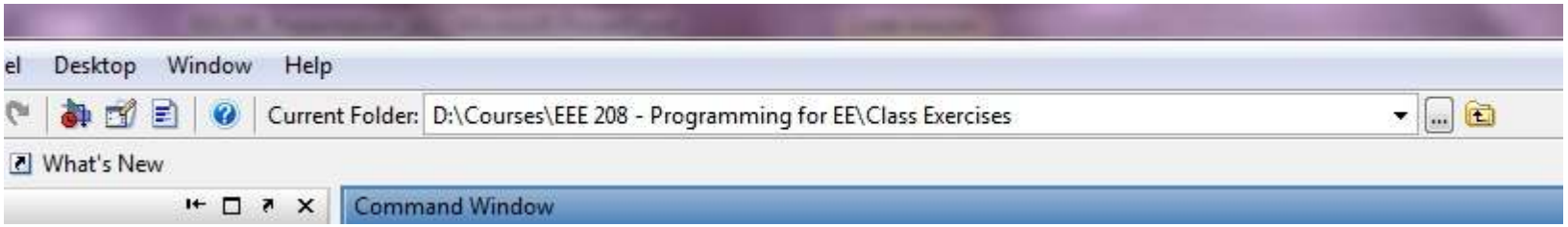
1. Comments section
   a. Include program's name and any key words in the first line.
   b. The date created, and the creators' names.
   c. The definitions of variable names for every input and output variable. Include definitions of variables used in the calculations and *units of measurement for all input and all output variables!*
   d. The name of every user-defined function called by the program.
2. Input section

   Include input data and/or the input functions and comments for documentation.
3. Calculation section
4. Output section

   This section might contain functions for displaying the output on the screen.

Loading Functions

- On the top of the main MATLAB window, you can find the address of Current Folder.



- MATLAB runs all of its own functions, but the user-defined functions that you want to call/load/run have to be inside the current folder.
- Otherwise, you'd receive an error:
- `??? Undefined function or method 'myfun2' for input arguments of type 'double'.`

Class Exercise 1

1. Write a function that calculates the surface area A and volume V of a sphere with the radius r. In other words, `r` is the input argument and `A` and `V` are the outputs.
2. Then save this function as yourlastname.m
3. Check the function with r = 10.

We can check and control number of input and output arguments

- Matlab has 4 functions for this purpose:
- Nargin: the number of input arguments
- varargin
- nargout
- varargout
- We can use them inside the body of functions

Functions and Plots?

- We generally don't want functions to plot things every time they run, so we usually write an m-file to call the function, calculate vectors of variables, and then plot a function
- But, there are exceptions

Example: plotSin

- Write a function with the following declaration: **function plotSin(f1)**. This function plots a graph, but does not calculate an output.
- In the function, plot a sin wave with frequency f1, on the range [0,2π]. To get a good sampling, use 16 points per period.

- `function plotSin(f1)`
- `x=linspace(0,2*pi,f1*16+1); % Uses 16 points for each interval`
- `figure`
- `plot(x,sin(f1*x))`

- Now play with this function. In the command window, write plotSin(1) or plotSin(10) or plotSin(100). What is different between these plots?

Example: plotSin2

```
function plotSin2(f1,f2)
% This function plots sin(f1*x) for one input
  argument. If there are two input arguments, it
  displays a message.
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
```

Now play with this function. In the command window, write plotSin2(1,2) or plotSin2(10,2)

Local and Global Variables 1

- **Local Variables**

1. The names of the input variables given in the function definition line are local to that function.
2. This means that other variable names can be used when you call the function.
3. All variables inside a function are erased after the function finishes executing, except when the same variable names appear in the output variable list used in the function call.

Local and Global Variables 2

- **Global Variables**

1. The global command declares certain variables global, and therefore their values are available to the basic workspace and to other functions that declare these variables global.
2. The syntax to declare the variables a, x, and q is
   ```
   global a x q
   ```
3. Any assignment to those variables, in any function or in the base workspace, is available to all the other functions declaring them global.

Function Handles

- You can create a function handle to any function by using the *at* sign, @, before the function name. You can then use the handle to refer to a the function.
- To create a handle to the function $y = x + 2e^{-x} - 3$, define the following function file:
- `function y = myfun3(x)`
- `y = x + 2*exp(-x) - 3;`

- You can use the **fzero** function to find the zero of a function of a single variable, which is denoted by x. One form of its syntax is
- `first_try=fzero(@myfun3,1) % finds the root of this function using 1 as a first guess`
- `second_try=fzero(@myfun3,-0.5) % finds the root of this function using 1 as a first guess`

Calculating f(x) for a vector of inputs

- Consider the same function
- `function y = myfun3(x)`
- `y = x + 2*exp(-x) - 3;`

- Now suppose you want to find its value over [-1,+1]. In the command window or an m-file write
- `t = -1:0.01:1;`
- `u= myfun3(t);`
- `plot(t,u)  % or plot(t,myfun3(t))`

- Make sure you understand this part. We will use these commands a lot!