ENGR 1100

Week 07- User Defined Functions

(class lecture)

Learning Objectives

Upon completion this module, students will be able to:

1. Be able to explain what user defined functions are and why do we need them.
2. Demonstrate how to create user defined functions in MATLAB.
3. Apply anonymous function in MATLAB programming
4. Demonstrate how to use functional handles to pass a function into a function function

MATLAB Function

MATLAB: Overview of Function

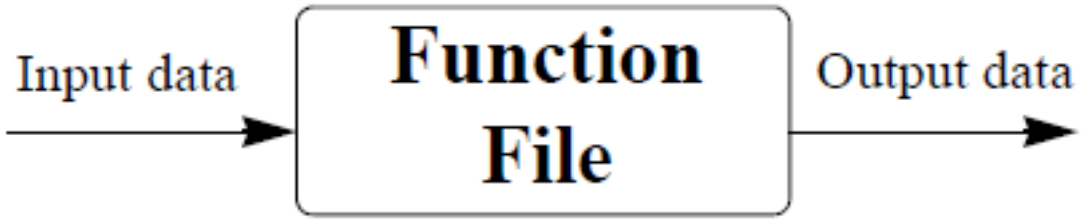
MATLAB: Create Functions in Files

MATLAB: Types of Functions

What is a Function?

A *function* is a MATLAB program that can accept inputs and produce outputs. Some functions don't take any inputs and/or produce outputs.

A function is *called* or *executed* (run) by another program or function. That program can pass the called function input and receive the called function's output.



Why use functions in MATLAB?

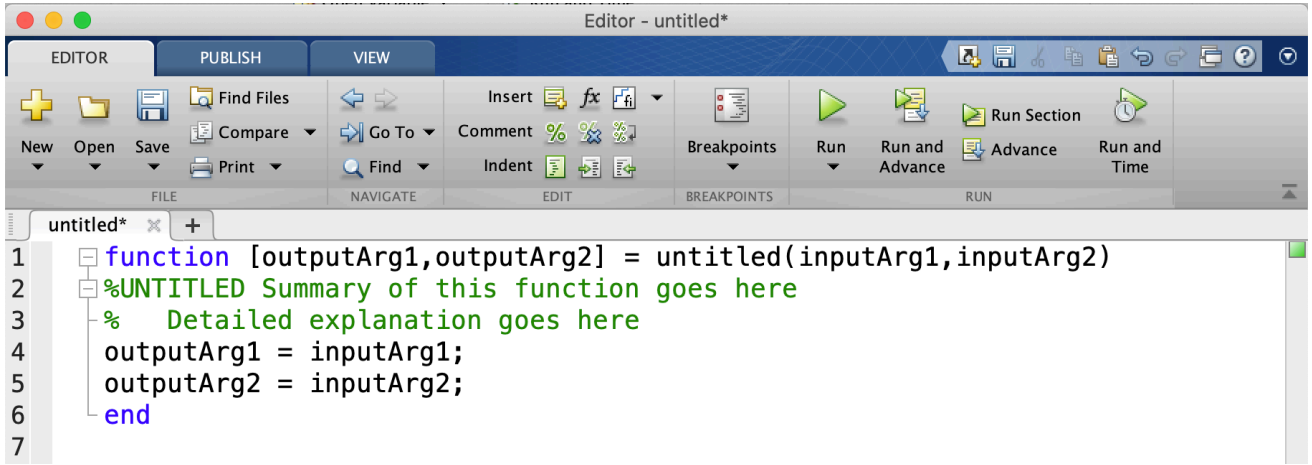
- Use same code in more than one place in program without rewriting code
- Reuse code by calling in different programs
- Make debugging easier by putting all of one kind of functionality in one place

Create a function

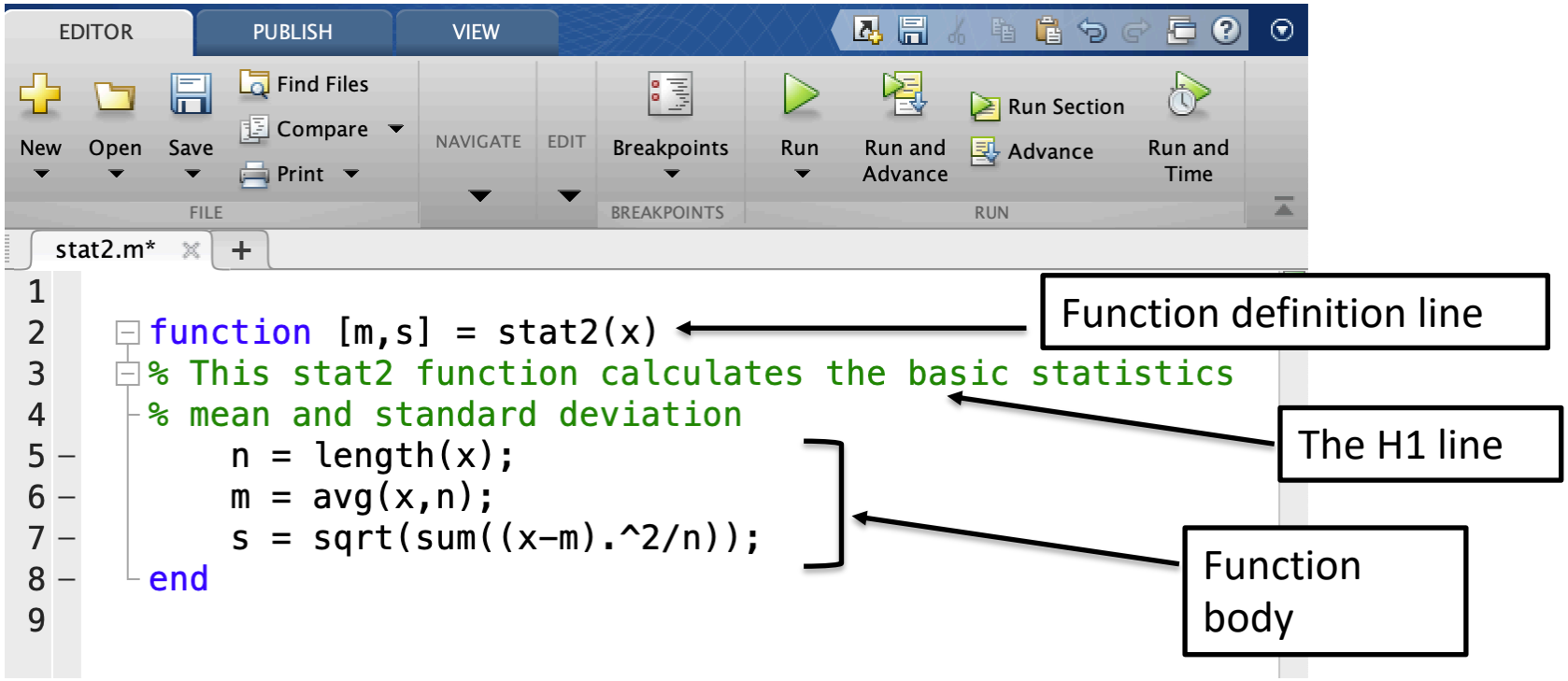
Code for a function is in an *m-file*

- Can make the file in any text editor but easiest to do it with MATLAB Editor Window

To create a new function m-file, on MATLAB desktop, click on **New** icon, then select **Function**. MATLAB opens a window opens that looks like

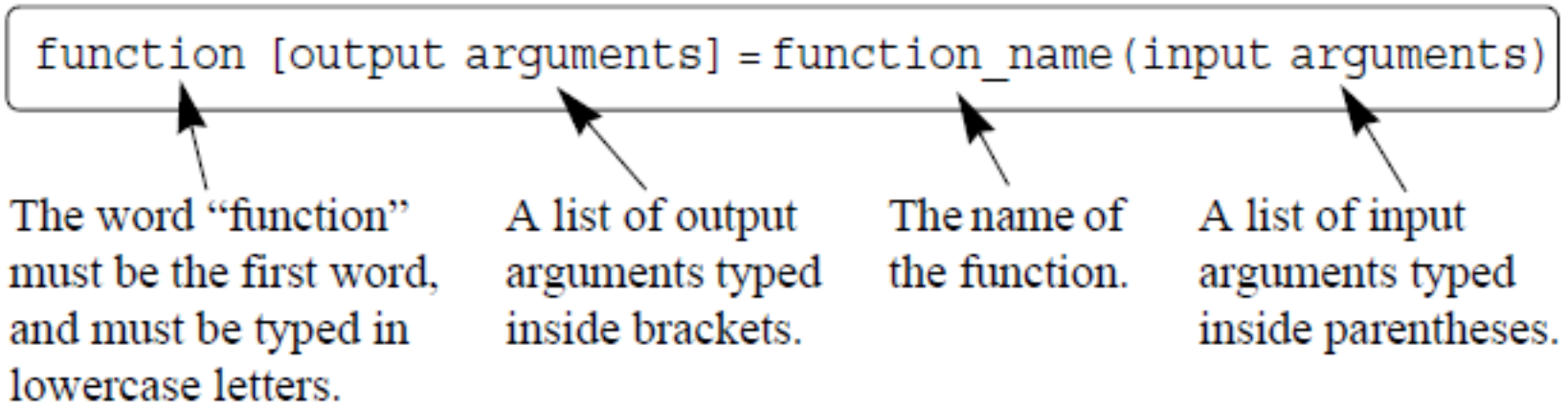


Example of a complete function file



Elements of a Function

- An *executable line* is a line of code that MATLAB can run
- A *function definition line*
  - Must be first executable line in function file, if not, MATLAB considers file to be a script file
  - Defines file as function file
  - Defines name of function
  - Defines number and order of input and output arguments



Function outputs

- Output arguments
  - Used to transfer data out of function to calling program
  - Can be zero or more output arguments
    - If zero arguments, can omit assignment operator (=)
    - If only one output argument, can omit brackets around it
    - If multiple arguments, separate them with **commas**
  - Can be scalars, vectors, or arrays
  - Function must assign values to all output arguments before it finishes running

Function definition lines

Examples of function definition lines

Function definition line	**Comments**
function [mpay,tpay] = loan(amount,rate,years)	Three input arguments, two output arguments.
function [A] = RectArea(a,b)	Two input arguments, one output argument.
function A = RectArea(a,b)	Same as above; one output argument can be typed without the brackets.
function [V, S] = SphereVolArea(r)	One input variable, two output variables.
function trajectory(v,h,g)	Three input arguments, no output arguments.

Lookfor command

If we want name of MATLAB command that does some particular computation, we can search for name by specifying a word or phrase likely to describe that computation. Do this with `lookfor` command:

```
lookfor 'word or phrase'
```

Example: Suppose want to find command that computes square root of a number

```
>> lookfor 'square'
magic - Magic square.
realsqrt - Real square root.
sqrt - Square root.
lscov - Least squares with known covariance.
sqrtm - Matrix square root.
cgs - Conjugate Gradients Squared Method.
```

H1 line

A more precise, likely phrase is "square root"

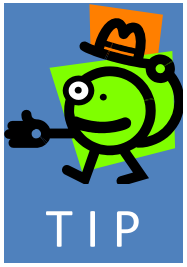
```
>> lookfor 'square root'
realsqrt - Real square root.
sqrt - Square root.
sqrtm - Matrix square root.
```

Good news – easy to make `lookfor` work on functions you write, not just MATLAB functions, by using H1 lines

- The *H1 line* is the first comment line after the function definition line
  - Optional
  - Only one line long
  - Usually contains function name and short definition of function

H1 line

`lookfor 'word'` searches in H1 line of all the files that it knows about (including ones you write) for "word". If it finds "word" it prints out file name and H1 line.



When you write an H1 line, put in words that people will likely use in `lookfor`

- Good example – uses "square" and "root"

```
% mySqrt(x): computes square root of x using an algorithm
```

- Bad example – uses "half" and "power"

```
% mySqrt(x): computes x to half power using an algorithm
```

Example

```
function y = lowpass( x, fHigh )
% LOWPASS - lowpass filter with specified cutoff frequency
% USAGE: y = lowpass( x, fHigh )
*%
% AUTHOR: Joe Blow
 X = fft( x );
...
```

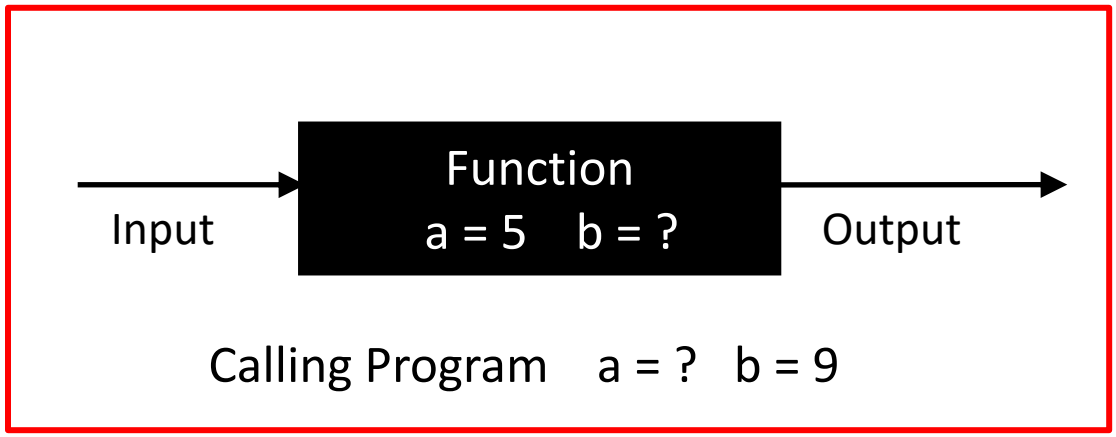
```
>> help lowpass
LOWPASS - lowpass filter with specified cutoff
frequency
USAGE: y = lowpass( x, fHigh )

AUTHOR: Joe Blow
```

← Blank line produced by line * above

Function body

- Contains code that does computations
  - All MATLAB programming features studied before
  - Special MATLAB commands relevant only to functions
- Variables declared inside a function are ***local*** to that function, i.e., can only be used inside that function
  - Calling program can't see (access) any of the variables inside the function
  - Function can't see any variables in the calling program



Example: Variables

Variables inside and outside function can have same names but are still different variables. Changing one doesn't change other

```
function test( n )
fprintf( 'On entering function n = %d\n', n );
n = 10;
fprintf( 'Before leaving function n = %d\n', n );
```

```
>> n = 5
n = 5
>> test(n)
On entering function n = 5
Before leaving function n = 10
>> n
n = 5
```

← n still 5 after function call

Global variables

By default, variables are local. To make a variable be recognized among various functions and/or the workspace, you must declare it to be ***global***. Do this with the line

```
global variable_name
```

- You can declare multiple global variables in one line by separating with spaces, e.g.,

```
global v1 v2 v3
```

- `global` command has to be entered in Command Window and/or in script file for variable to be recognized in workspace
- It is common to use long descriptive names (or all capital letters) for global variables in order to distinguish them from regular variables

NOTE – use of global variables no longer considered good practice. Avoid them when possible

How to run functions

You must save a file before you can use it

- Click on Save icon
- **Strongly recommend** to give file same name as function in it and followed by ".m"
- To use function its file must be in current folder or in search path
- Examples

Function definition line	File name
function [mpay,tpay] = loan(amount,rate,years)	loan.m
function [A] = RectArea(a,b)	RectArea.m
function [V, S] = SphereVolArea(r)	SphereVolArea.m
function trajectory(v,h,g)	trajectory.m

How to call user-defined functions

User-defined functions called same way as built-in functions

- From Command Window
- From script file
- From another function

Example: How much money will I pay monthly and in total between now and when my kid turns 18 on a 6% loan of $10,000?

```
function [mpay tpay]=loan(amount,rate,years)
>> kidAge = 11;
>> rate = 6;
>> [mpay tpay] = loan( 10000, rate, 18-kidAge )
```

Function vs Script

Characteristic	Function	Script
File extension	.m	.m
First executable line	Function definition line	Any
Variables recognized in Command Window?	No	Yes
Can use variables defined in workspace?	No	Yes
Can accept input arguments?	Yes	No
Can return output arguments?	Yes	No

MATLAB Example 1

Write a function file (name it *myFunction*) for the below math function $f(x) = \frac{x^4\sqrt{3x+5}}{(x^2+1)^2}$.

The input to this function is x and the output is $f(x)$. Write the function such that x can be a vector. Use the function to calculate

a) $f(x)$ for $x=6$

b) $f(x)$ for x=1,3,5,7,9 and 11

MATLAB Example 2

Write a **function** that asks the user to enter the values for the three constants of the quadratic equation (a, b, and c). Use an if-elseif-else-end structure to warm the user if $b^2 - 4ac > 0$, $b^2 - 4ac = 0$, or $b^2 - 4ac < 0$. If $b^2 - 4ac >= 0$, determine the solution. Use the following to double-check the functionality of your function:

- Use a = 1, b = 2, c = -1
- Use a = 1, b = 2, c = 1
- Use a = 10, b = 1, c = 20

Subfunctions

- Commonly used to make primary function clearer by doing some of its computations
- Commonly not useful on their own or for being called by code other than primary function
- Subfunctions are defined within the primary function

```
function [me SD] = stat(v)
n=length(v);
me=AVG(v,n);
SD=StandDiv(v,me,n);

function av=AVG(x,num)
av=sum(x)/num;

function Sdiv=StandDiv(x,xAve,num)
xdif=x-xAve;
xdif2=xdif.^2;
Sdiv= sqrt(sum(xdif2)/(num-1));
```

The primary function.

Subfunction.

Subfunction.

The user-defined function `stat` is then used in the Command Window for calculating the average and the standard deviation of the grades:

```
>> Grades=[80 75 91 60 79 89 65 80 95 50 81];
>> [AveGrade StanDeviation] = stat(Grades)
AveGrade =
   76.8182
StanDeviation =
   13.6661
```

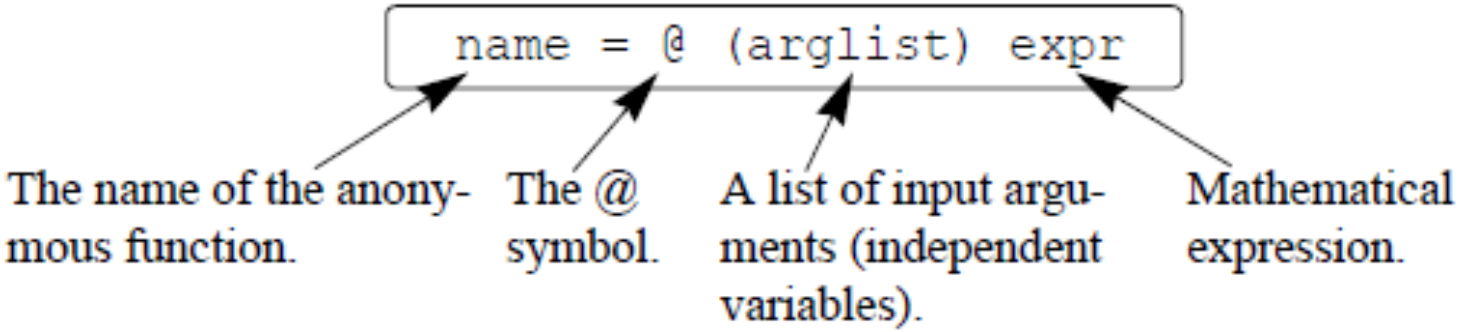
Anonymous Function

An *anonymous function* is a function defined without using a separate function file

- For short (one line!) functions only
- Usually written by user
- Specified within another function or body of code, not in separate file
- Often used with MATLAB functions that operate on other functions, e.g.,
  - Find area under a function
  - Find derivatives of a function
  - Find optimal value of a function

How to create anonymous function?

Can make anonymous function in Command Window, script file, or inside user-defined function. Form is:



- `name` – name of function, using rules for names of user-defined functions
- `@` - a *function handle*, an object that has information about the function
- `arglist` – zero or more function arguments, defined like those for user-defined functions, independent variables are separated by comma
- `expr` – a single MATLAB expression

For example

```
cube = @(x) x^3
```

Example : Anonymous function

- Can be included in any built-in or user-defined functions
- Called same way as user-defined functions

```
>> triple = @(n) 3 * n
triple =
    @(n)3*n
>> triple( 4 )
ans = 12
>> x = 1.5;
>> y = triple( x )
y = 4.5000
```

An anonymous function can use in its body any (visible) variables that have been defined before the function itself is defined

Values/parameters that function uses are those that the variables have when function is defined (not when it is called)!

Common Pitfalls 1

- variable undefined before anonymous function defined

```
>> p = 5;
>> combo = @(x,y) p*x + q*y;
>> combo(2,8)
??? Undefined function or variable 'q'.
Error in ==> @(x,y)p*x+q*y
```

Solution

```
>> p = 5;
>> q = 4;
>> combo = @(x,y) p*x + q*y;
>> combo(2,8)
ans = 42
```

Common Pitfalls 2

- Anonymous function unaffected by variable change after function defined

```
>> p = 5;
>> q = 4;
>> combo = @(x,y) p*x + q*y;
>> combo( 2, 8 )
ans = 42
>> p = 0
p = 0
>> q = 0
q = 0
>> combo( 2, 8 )
ans = 42
```

MATLAB Exercise

Write an anonymous function to compute *sin(2πt)* and use the MATLAB command `quad` to find the area under the function from 0 to 1.

```
>> mysine = @(t) sin( 2*pi*t );
>> quad( mysine, 0, 1 )
ans =
     0
```

Function function

A MATLAB function that accepts another function as an input is called a *function function* (seriously!)

For example

- `quad` finds area under passed function
- `fzero` finds zeros of passed function

Function handles are used for passing functions to function functions

- Passed functions can be built-in, user defined, or anonymous

To make a **function handle** for built-in or user-defined function, put the @ symbol in front of function name

- e.g., function handle for MATLAB `cos` function is `@cos`

A name for the function that is passed in.

```
function xyout=funplot(Fun,a,b)
% funplot makes a plot of the function Fun which is passed in
% when funplot is called in the domain [a, b].
% Input arguments are:
% Fun:  Function handle of the function to be plotted.
% a:  The first point of the domain.
% b:  The last point of the domain.
% Output argument is:
% xyout: The values of x and y at x=a, x=(a+b)/2, and x=b
% listed in a 3 by 2 matrix.

x=linspace(a,b,100);
y=Fun(x);
xyout(1,1)=a; xyout(2,1)=(a+b)/2; xyout(3,1)=b;
xyout(1,2)=y(1);
xyout(2,2)=Fun((a+b)/2);
xyout(3,2)=y(100);
plot(x,y)
xlabel('x'), ylabel('y')
```

Using the imported function to calculate $f(x)$ at 100 points.

Using the imported function to calculate $f(x)$ at the midpoint.

Pass user-defined function by using @ symbol followed by function name

EXAMPLE

```
function y=Fdemo(x)
y=exp(-0.17*x).*x.^3-2*x.^2+0.8*x-3;
```

```
>> ydemo=funplot(@Fdemo,0.5,4)
ydemo =
    0.5000   -2.9852
    2.2500   -3.5548
    4.0000    0.6235
```

Enter a handle of the user-defined function Fdemo.

Pass anonymous function name without @ symbol

EXAMPLE

```
>> FdemoAnony=@(x) exp(-0.17*x).*x.^3-2*x.^2+0.8*x-3
FdemoAnony =
    @(x) exp(-0.17*x).*x.^3-2*x.^2+0.8*x-3

>> ydemo=funplot(FdemoAnony,0.5,4)
ydemo =
    0.5000   -2.9852
    2.2500   -3.5548
    4.0000    0.6235
```

Create an anonymous function for $f(x)$.

Enter the name of the anonymous function (FdemoAnony).