

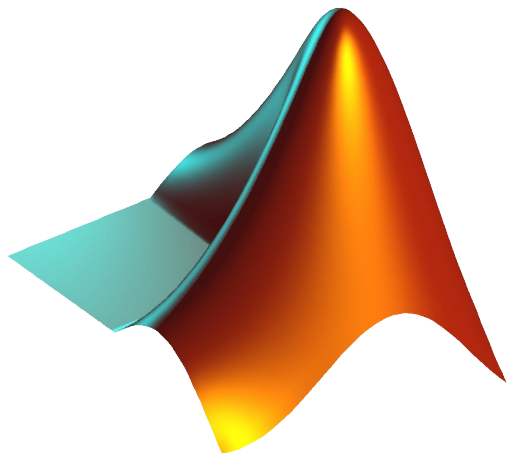
CS 1112 Introduction to Computing Using MATLAB

Instructor: Dominic Diaz

Website:

<https://www.cs.cornell.edu/courses/cs1112/2022fa/>

Today: Functions part 2



Agenda and announcements

- Last time
 - Functions
- This time
 - More functions
- Announcements
 - Exercises checked off on MATLAB grader. You should have just been enrolled earlier today to MATLAB grader. Try out Exercise 0 before tomorrow's section!
 - Project 1 grades released soon...
 - Project 3 released later this week

Tips for succeeding in this class

- This is the point in the class where things start to snowball.
 - Everything we have covered until now will continue to show up
 - Project 3 is released in a few days... take this time to make sure you understand everything until now
- Don't let your project partner do all of the work
 - Make sure you have at least some knowledge about how each project problem is solved.
- Never approach a problem with “I don't know” or “I'll just go to office hours and figure it out with the TA”.
 - Try to understand what the problem is asking for before you talk to course staff.
 - Attempt each problem before you get help!

General form of a user-defined function

```
function [out1, out2, ...] = functionName(in1, in2, ...)  
% 1-line comment to describe the function  
% Additional lines of comments to describe the inputs and outputs  
  
[Code that, at some point, assigns values to output variables or  
does something]
```

If the function has no inputs:

```
function [out1, out2, ...] = functionName()
```

If the function has no outputs:

```
function functionName(in1, in2, ...)
```

General form of a user-defined function

```
function [out1, out2, ...] = functionName(in1, in2, ...)  
% 1-line comment to describe the function  
% Additional lines of comments to describe the inputs and outputs  
  
[Code that, at some point, assigns values to output variables or  
does something]
```

- `in1, in2, ...` must be defined when the function begins execution. Variables `in1, in2, ...` are called *input parameters* or *inputs* or *arguments* and they hold the function inputs used when the function is invoked (called).
- `out1, out2, ...` are not defined until the code in the function assigns values to them and are called the *output parameters* or *outputs*

Process for writing a function

Design: set the specifications

1. Identify candidates
 - a. Look for opportunity to reuse logic or improve clarity
2. Design interface
 - a. Choose function name, input parameters, outputs and/or effects
3. Implement function
 - a. Write code according to specifications
4. Test
 - a. Try it out (and try to break it—test for all possible cases)
5. Use

Returning versus printing

```
function [x, y] = polar2xy_out(r, theta)
% convert polar (r, theta) to cartesian
% (x,y)
```

```
rads = theta*pi/180;
x = r*cos(rads);
y = r*sin(rads);
```

Code to call the above function:

```
% Convert polar (r1, t1) to Cart (x1, y1)
r1 = 1; t1 = 30;
[x1, y1] = polar2xy_out(r1, t1);
plot(x1, y1, 'b*')
```

Outputs are useful when you need to use values outside of a function

```
function polar2xy_print(r, theta)
% convert polar (r, theta) to cartesian
% (x,y)
```

```
rads = theta*pi/180;
x = r*cos(rads);
y = r*sin(rads);
fprintf('x = %f, y = %f', x, y)
```

Code to call the above function:

```
% Convert polar (r1, t1) to Cart (x1, y1)
r1 = 1; t1 = 30;
polar2xy_print(r1, t1);
plot(x1, y1, 'b*')
```

This plot would not work because the cartesian coordinates were not outputs of the function

Subfunctions, also called “local functions”

- There can be more than one function in an m-file
- If you are writing a function file, the first function is the main function and must have the same names as the file name
 - Remaining functions are subfunctions, accessible only by the functions in the same m-file

Two examples of subfunctions

```
function [avg, med] = mystats(x, y, z)
% Compute statistics of the numbers x, y, z
```

```
avg = mymean(x, y, z);
med = mymedian(x, y, z);
end
```

```
function a = mymean(x, y, z)
% Compute the mean of x, y, z
```

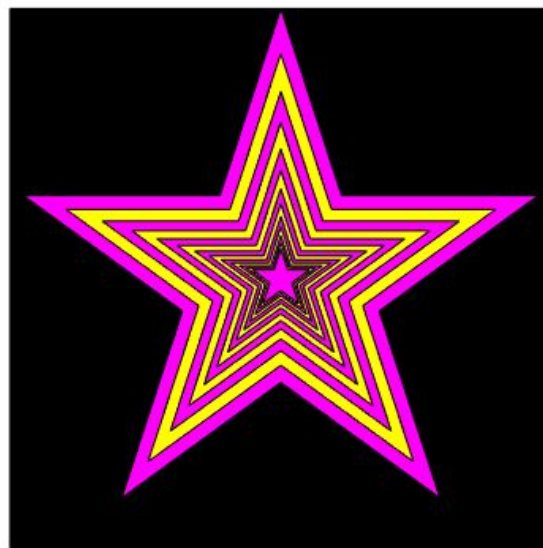
```
a = (x + y + z)/3;
end
```

```
function m = mymedian(x, y, z)
% Compute median of x, y, z
```

```
[code]
end
```


Reasons to use functions

- A function can be tested easily and lets you break your larger problem into more manageable tasks
- Keep your main driver function/script clean by keeping the detailed code in functions—reflects top-down design
- More maintainable software

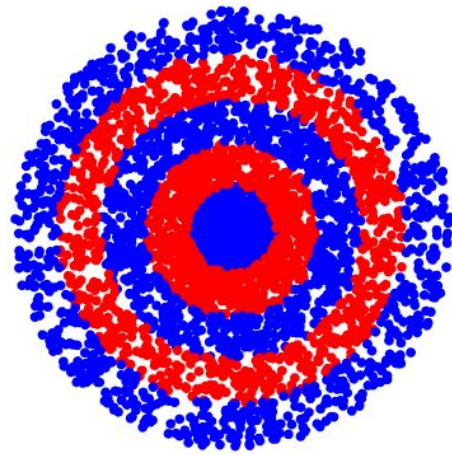


When we were tasked with drawing nested stars, we can break down the problem and first write a function that draws one star.

```
function DrawStar(xc,yc,r,c)
% Adds a 5-pointed star to the current window
...
end
```

Reasons to use functions

- A function can be tested easily and lets you break your larger problem into more manageable tasks
- Keep your main driver function/script clean by keeping the detailed code in functions—reflects top-down design
- More maintainable software



```
% Put dots in the area between circles with  
radii R and (R-1)  
for R = 1:c  
    % Draw d dots  
    for dotNum = 1:d  
        radius = rand + R-1;  
        theta = rand*(360);  
        [x, y] = Polar2xy(radius, theta);  
        DrawColorDot(x, y, rem(R,2));  
    end  
end
```

Reasons to use functions

- A function can be tested easily and lets you break your larger problem into more manageable tasks
- Keep your main driver function/script clean by keeping the detailed code in functions—reflects top-down design
- More maintainable software

Today: I write a function `ePerimeter(a,b)` that computes the perimeter of the ellipse $\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1$

During this year: You write software that makes extensive use of `ePerimeter(a,b)`. Imagine hundred of programs that call (use) `ePerimeter`.

Next year: I discover a better way to approximate ellipse perimeters. I change the implementation of `ePerimeter(a,b)`. You do **not** have to change your programs that call function `ePerimeter` at all!

Script versus function

A script is executed line-by-line just as if you are typing it into the Command Window

- The value of a variable in a script is stored in the Command Window Workspace

```
% Convert polar (r1, t1) to Cart (x1, y1)
r1 = 1; t1 = 30;
[x1, y1] = polar2xy_out(r1, t1);
plot(x1, y1, 'b*')
```

A function has its own private (local) function workspace that does not interact with the workspace of other functions or the Command Window workspace

- Variables are not shared between workspaces even if they have the same name

```
function polar2xy_print(r, theta)
% convert polar (r, theta) to cartesian
% (x,y)
```

```
rads = theta*pi/180;
x = r*cos(rads);
y = r*sin(rads);
fprintf('x = %f, y = %f', x, y)
```

You can use the keyword `global` to make global variables, but we don't need to worry about that in CS 1112

Calling a function with outputs

When I run a script that calls a function, a separate workspace is created for the function variables:

```
x = 10; y = 20;  
z = funct1(x, y);  
disp(z)
```

Script workspace:

x	10
---	----

y	20
---	----

z	25
---	----

```
function z = funct1(v, w)  
x = 0.5*v;  
z = x + w;
```

Function workspace:

v	10
---	----

w	20
---	----

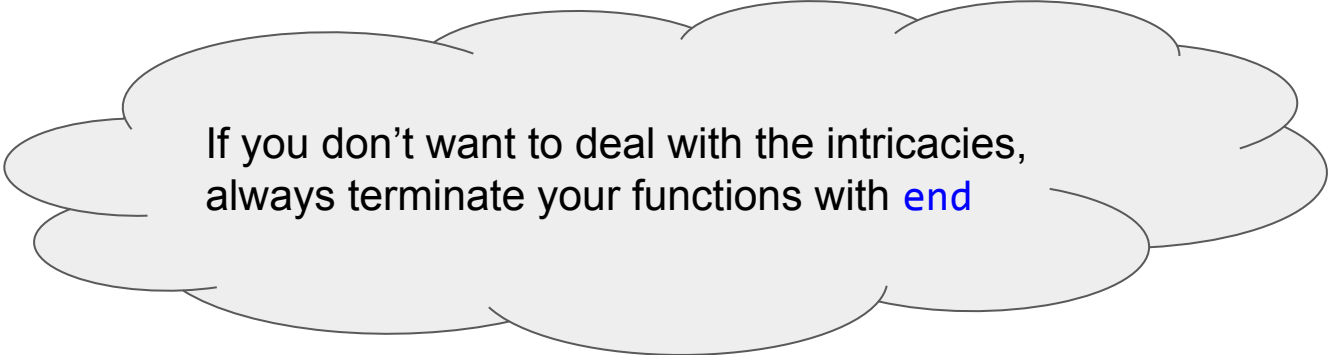
x	5
---	---

z	25
---	----

Ending functions

`end` can (and sometimes should) terminate a declared function. It's usually optional but use `end` for better code readability. `end` is required in these cases:

- If a file contains functions, and one function terminates with `end`, then every function in the file must be terminated with `end`
- If a file contains nested functions (won't need to do this in CS 1112), then every function must be terminated with `end`
- Functions in scripts must be terminated with `end`



If you don't want to deal with the intricacies,
always terminate your functions with `end`

Terminating with `end` examples

```
function [x, y] = polar2xy(r, theta)
% convert polar (r, theta) to cartesian
% (x,y). theta in degrees.

rads = theta*pi/180;
x = r*cos(rads);
y = r*sin(rads);

end
```

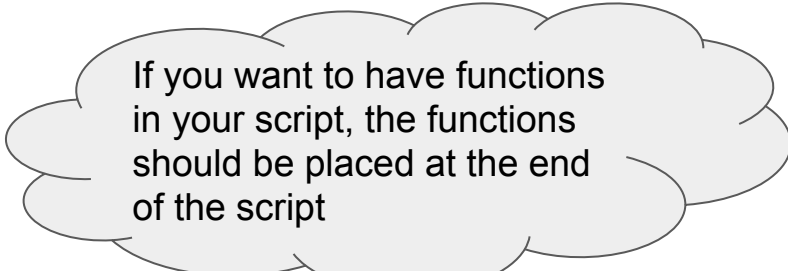
```
% script to compute cartesian coords.
% from polar coords.
```

```
r1 = 2;
theta1 = 30;
[x,y] = polar2xy(r1, theta1);
```

```
function [x, y] = polar2xy(r, theta)
% convert polar (r, theta) to cartesian
% (x,y). theta in degrees.
```

```
rads = theta*pi/180;
x = r*cos(rads);
y = r*sin(rads);
```

```
end
```



If you want to have functions in your script, the functions should be placed at the end of the script

Exercise to try on your own (similar to a previous prelim question)

<i>Script</i>	<i>Function (in foo.m)</i>
<pre>a = 2; b = 8; c = foo(b, a+1); fprintf('a is %d\n', a) fprintf('b is %d\n', b) fprintf('c is %d\n', c)</pre>	<pre>function c = foo(a, b) c = a + b; b = b - 7; a = b; b = a;</pre>

What does this script output?