- **Previous lecture**
  - User-defined functions
    - Function header
    - Input parameters and return variables

- **Today’s lecture**
  - User-defined functions
    - local memory space
    - Subfunction
  - 1-dimensional array and plot

- **Announcement**
  - Discussion this week in classrooms as listed in Student Center
  - Make use of consulting/office hours
  - Please give us feedback on the mid-semester TA evaluation administered by Engineering
General form of a user-defined function

```matlab
function [out1, out2, ...] = functionName (in1, in2, ...)
% 1-line comment to describe the function
% Additional description of function

Executable code that at some point assigns
values to output parameters out1, out2, ...
```

- `in1, in2, ...` are defined when the function begins execution. Variables `in1, in2, ...` are called function *parameters* and they hold the function *arguments* used when the function is invoked (called).
- `out1, out2, ...` are not defined until the executable code in the function assigns values to them.
Returning a value ≠ printing a value

You have this function:

```matlab
function [x, y] = polar2xy(r, theta)
% Convert polar coordinates (r,theta) to
% Cartesian coordinates (x,y). Theta in degrees.
...
```

Code to call the above function:

```matlab
% Convert polar (r1,t1) to Cartesian (x1,y1)
r1 = 1; t1 = 30;
[x1, y1] = polar2xy(r1, t1);
polar2xy(r1, t1);
plot(x1, y1, 'b*')
...
```
Returning a value ≠ printing a value

You have this function:

```matlab
function [x, y] = polar2xy(r, theta)
% Convert polar coordinates (r,theta) to
% Cartesian coordinates (x,y). Theta in degrees.
...
```

Code to call the above function:

```matlab
% Convert polar (r1,t1) to Cartesian (x1,y1)
r1 = 1; t1 = 30;
[x1, y1] = polar2xy(r1, t1);
plot(x1, y1, 'b*')
```

Function prints instead of returns values.

Now, although you can see the coordinates, this script cannot use them.

... Not possible to do
Given this function:

```matlab
function m = convertLength(ft,in)
% Convert length from feet (ft) and inches (in)
% to meters (m).

% ... %
```

How many proper calls to `convertLength` are shown below?

```matlab
% Given f and n
d= convertLength(f,n);
d= convertLength(f*12+n);
d= convertLength(f+n/12);
x= min(convertLength(f,n), 1);
y= convertLength(pi*(f+n/12)^2);
```

A: 1  B: 2  C: 3  D: 4  E: 5 or 0
Comments in functions

- Block of comments after the function header is printed whenever a user types `help <functionName>` at the Command Window.

- 1st line of this comment block is searched whenever a user types `lookfor <someWord>` at the Command Window.

- Every function should have a comment block after the function header that says **what the function does concisely**.
Accessing your functions

For now*, put your related functions and scripts in the same directory.

MyDirectory

dotsInRings.m  polar2xy.m
randDouble.m  drawColorDot.m

Any script/function that calls polar2xy.m

*The path function gives greater flexibility
Why write user-defined function?

- Easy code re-use—great for “common” tasks
- A function can be tested independently easily
- Keep a **driver** program clean by keeping detail code in **functions**—separate, non-interacting files
- Facilitate top-down design
c = input('How many concentric rings? ');
d = input('How many dots? ');

% Put dots btwn circles with radii rRing and (rRing-1)
for rRing = 1:c
    % Draw d dots
    for count = 1:d

        % Generate random dot location (polar coord.)
        theta = _______
        r = _______

        % Convert from polar to Cartesian
        x = _______
        y = _______

        % Use plot to draw dot
    end
end

Each task becomes a function that can be implemented and tested independently.
Facilitates top-down design

1. Focus on how to draw the figure given just a specification of what the function `DrawStar` does.

2. Figure out how to implement `DrawStar`. 
To **specify** a function...

... you describe how to use it, e.g.,

```matlab
function DrawStar(xc,yc,r,c)
% Adds a 5-pointed star to the
% figure window. Star has radius r,
% center(xc,yc) and color c where c
% is one of 'r', 'g', 'y', etc.
```

*Given the specification, the user of the function doesn't need to know the detail of the function—they can just use it!*
To **implement** a function...

... you write the code so that the function “lives up to” the specification. E.g.,

```matlab
r2 = r/(2*(1+sin(pi/10))); for k=1:11
    theta = (2*k-1)*pi/10;
    if 2*floor(k/2)~=k
        x(k) = xc + r*cos(theta);
        y(k) = yc + r*sin(theta);
    else
        x(k) = xc + r2*cos(theta);
        y(k) = yc + r2*sin(theta);
    end
end
fill(x,y,c)
```

Don’t worry—you’ll learn more about graphics functions and vectors soon.
Why write user-defined function?

- Easy code re-use—great for “common” tasks
- A function can be tested independently easily
- Keep a **driver** program clean by keeping detail code in **functions**—separate, non-interacting files
- Facilitate top-down design

Software management
I write a function

\[ \text{EPerimeter}(a,b) \]

that computes the perimeter of the ellipse

\[
\left( \frac{x}{a} \right)^2 + \left( \frac{y}{b} \right)^2 = 1
\]
Software Management

During this year:

You write software that makes extensive use of \texttt{EPerimeter(a,b)}

Imagine hundreds of programs each with several lines that reference \texttt{EPerimeter}
Software Management

Next year:

I discover a more efficient way to approximate ellipse perimeters. I change the implementation of $E_{\text{Perimeter}}(a,b)$

You do not have to change your software at all.
Script vs. 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

- 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
What will be printed?

A: -3  B: 3  C: error

% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is absolute value of p
if (p<0)
    p = -p;
end
q = p;
What will be printed?

% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is the absolute value of p
if (p<0)
    p = -p;
end
q = p;
What will be printed?

% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is the absolute value of p
if (p<0)
    p = -p;
end
q = p;

Command Window Workspace
p
-3
What will be printed?

% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is the absolute value of p
if (p<0)
    p = -p;
end
q = p;

Command Window Workspace

p  -3
What will be printed?

% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is the absolute value of p
if (p<0)
    p = -p;
end
q = p;

Command Window Workspace

Function absolute’s Workspace

p | -3
% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is the absolute value of p
if (p<0)
    p = -p;
end
q = p;

What will be printed?

Command Window Workspace

| p | -3 |

Function absolute's Workspace

| p | -3 |
What will be printed?

% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is the absolute value of p
if (p<0)
    p = -p;
end
q = p;

Command Window Workspace

Function absolute’s Workspace
What will be printed?

% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is the absolute value of p
if (p<0)
    p = -p;
end
q = p;

Command Window Workspace
p | -3

Function absolute’s Workspace
p | 3
What will be printed?

% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is the absolute value of p
if (p<0)
    p = -p;
end
q = p;

Command Window Workspace

| p | -3 |

Function absolute’s Workspace

| p | 3 |
| q | 3 |
What will be printed?

% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is the absolute value of p
if (p<0)
    p = -p;
end
q = p;

Command Window Workspace

| p | -3 |

Function absolute’s Workspace

| p | 3 |
| q | 3 |
What will be printed?

% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is the absolute value of p
if (p<0)
    p = -p;
end
q = p;

Command Window Workspace

| p | -3 |
| q | 3  |

Function absolute’s Workspace

| p | 3  |
| q | 3  |
What will be printed?

% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is the absolute value of p
if (p<0)
    p = -p;
end
q = p;

Command Window Workspace

| p | -3 |
| q |  3 |
What will be printed?

% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is the absolute value of p
if (p<0)
    p = -p;
end
q = p;

Command Window Workspace

| p | -3 |
| q | 3  |
% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
% q is the absolute value of p
if (p<0)
    p = -p;
end
q = p;

A value is passed to the function parameter when the function is called.

The two variables, both called p, live in different memory space and do not interfere.
% Script file
p = -3;
q = absolute(p);
disp(p)

function q = absolute(p)
    % q is the absolute value of p
    if (p<0)
        p = -p;
    end
    q = p;

When a function reaches the end of execution (and returns the output argument), the function space—local space—is deleted.
Execute the statement \( y = \text{foo}(x) \)

- Matlab looks for a function called \texttt{foo} (m-file called \texttt{foo.m})
- Argument (value of \( x \)) is copied into function \texttt{foo}'s local parameter
  - called “pass-by-value,” one of several argument passing schemes used by programming languages
- Function code executes within its own workspace
- At the end, the function’s output argument (value) is sent from the function to the place that calls the function. E.g., the value is assigned to \( y \).
- Function’s workspace is deleted
  - If \texttt{foo} is called again, it starts with a new, empty workspace