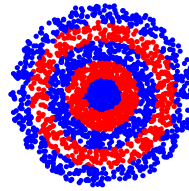


- Previous lecture
  - Finite/inexact arithmetic
  - Plotting continuous functions using vectors and vectorized code
  - User-defined functions
    - Function header
- Today's lecture
  - User-defined functions
    - Input parameters and return variables
    - Local memory space
    - Subfunction
- Announcements
  - Discussion this week in classrooms as listed on roster
  - Prelim 1 will be returned at end of lecture. If your paper isn't here, pick it up from CS1112 consultants in ACCEL during consulting hrs (starting today after 4pm)

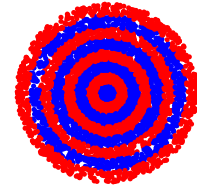
Lecture 10

2

### Draw a bulls eye figure with randomly placed dots



- Dots are randomly placed within concentric rings
- User decides how many rings, how many dots



Lecture 10

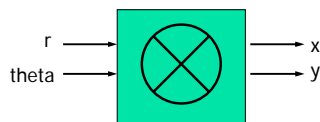
3

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

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

A function file  
polar2xy.m

Think of polar2xy as a factory



Lecture 10

6

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

% Generate random dot location
theta= _____
r= _____

% Convert from polar to Cartesian
rads= theta*pi/180;
x= r*cos(rads);
y= r*sin(rads);
[x, y] = polar2xy(r, theta);

% Draw the dot

end
end
```

Lecture 10

7

```
function [x, y] = polar2xy(r, theta)
```

Output  
parameter list  
enclosed in [ ]

Function name  
(This file's name is  
polar2xy.m)

Input parameter  
list enclosed in ( )

February 23, 2010

Lecture 10

8

### Returning a value ≠ printing a value

You have this function:

```
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:

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

Lecture 10

12

Given this function:

```
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?

```
% 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

Lecture 10

14

#### General form of a user-defined function

```
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.

Lecture 10

15

#### dotsInCircles.m

(functions with multiple input parameters)  
 (functions with a single output parameter)  
 (functions with multiple output parameters)  
 (functions with no output parameter)

Lecture 10

16

#### Comments in functions

- Block of **comments after the function header** is printed whenever a user types `help <functionName>` at the Command Window
- **1<sup>st</sup> line of this comment block** is searched whenever a user types `lookfor <someWord>` at the Command Window

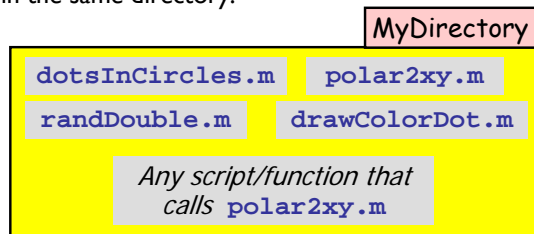
- ➡ Every function should have a comment block after the function header that says **what the function does** concisely

Lecture 10

17

#### Accessing your functions

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



\*The `path` function gives greater flexibility

Lecture 10

18

#### 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

Lecture 10

20

```

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

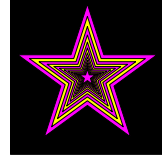
```

Each task becomes a function that can be implemented and tested independently

Lecture 10

21

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**.

Lecture 10

22

To specify a function...

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

```

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!

Lecture 10

23

To implement a function...

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

```

r2 = r/(2*(1+sin(pi/10)));
tau = pi/5;
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 soon.

Lecture 10

24

## Software Management

Today:

I write a function

**EPerimeter(a,b)**

that computes the perimeter of the ellipse

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1$$

Lecture 10

29

## Software Management

During this year :

You write software that makes extensive use of

**EPerimeter(a,b)**

Imagine hundreds of programs each with several lines that reference **EPerimeter**

Lecture 10

30

## Software Management

Next year:

I discover a more efficient way to approximate ellipse perimeters. I change the implementation of

`EPerimeter(a,b)`

You do **not** have to change your software at all.

Lecture 10

31

## 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

Lecture 10

33

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;
```

Lecture 10

35

Execute the statement `y= foo(x)`

- Matlab looks for a function called `foo` (m-file called `foo.m`)
- Argument (value of `x`) is copied into function `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 `foo` is called again, it starts with a new, empty workspace

Lecture 10

52

## Subfunction

- There can be more than one function in an M-file
- **top** function is the main function and has the name of the file
- remaining functions are **subfunctions, accessible only by the functions in the same m-file**
- Each (sub)function in the file begins with a **function header**
- Keyword **end** is not necessary at the end of a (sub)function

Lecture 10

55