- Previous Lecture:
  - Nested loops
  - Developing algorithms and code

- Today's Lecture:
  - Review nested loops
  - User-defined functions

- Announcements:
  - Discussion this week in Upson B7 lab. Read *Insight* §3.2 before discussion.
  - Project 2 due Thursday at 11pm

Rational approximation of π

- π = 3.141592653589793…
- Can be closely approximated by fractions, e.g., π ≈ 22/7
- Rational number: a quotient of two integers
- Approximate π as p/q where p and q are positive integers ≤M
- Start with a straight forward solution:
  - Get M from user
  - Calculate quotient p/q for all combinations of p and q
  - Pick best quotient → smallest error

Lecture 7   3

```
% Rational approximation of pi

M = input('Enter M: ');



% Check all possible denominators
for q = 1:M
    % At this q, check all possible numerators
    for p = 1:M




    end
end
```

Analyze the program for efficiency

- See Eg3_1 and FasterEg3_1 in the book

```
for a = 1:n
    disp('alpha')
    for b = 1:m
        disp('beta')
    end
end
```

How many times are "alpha" and "beta" displayed?

A: n, m

B: m, n

C: n, n+m

D: n, n*m
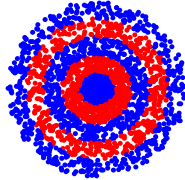
E: m*n, m

Lecture 7   12

Built-in functions

- We've used many Matlab built-in functions, e.g., `rand`, `abs`, `floor`, `rem`
- Example: `abs(x-.5)`
- Observations:
  - `abs` is set up to be able to work with any valid data
  - `abs` *doesn't prompt us for input; it expects that we provide data* that it'll then work on

Lecture 8   14

User-defined functions

- We can write our own functions to perform a specific task
  - Example: draw a disk with specified radius, color, and center coordinates
  - Example: generate a random floating point number in a specified interval
  - Example: convert polar coordinates to x-y (Cartesian) coordinates
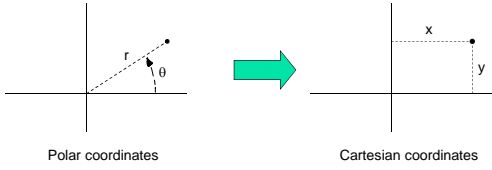
Lecture 8   15

Draw a bulls eye figure with randomly placed dots

- What are the main tasks?
- Accommodate variable number of rings—loop

- For each ring
  - Need many dots
  - For each dot
    - Generate random position
    - Choose color
    - Draw it

Lecture 8   18

Convert from polar to Cartesian coordinates

Polar coordinates   Cartesian coordinates

Lecture 8   19

```
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
```

A common task! Create a function **polar2xy** to do this. **polar2xy** likely will be useful in other problems as well.

Lecture 8   20

```
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
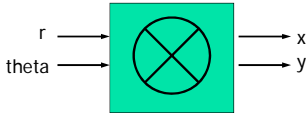
r ⟶   ⟶ x
theta ⟶   ⟶ y

```
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

```
r= input('Enter radius: ');
theta= input('Enter angle in degrees: ');
```

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

(Part of) a script file

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

    % Use plot to draw dot
  end
end
```

Lecture 8   29

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

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

Input parameter
list enclosed in
( )

Output
parameter list
enclosed in [ ]

Lecture 8   30

Function header is the "contract" for how the function will be used (called)

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 8   31

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 8   34

---

dotsInRings.m

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

Lecture 8   35

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*')
…

36

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 <u>what the function does concisely</u>

Lecture 8   38

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 |

Accessing your functions

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

MyDirectory

```
dotsInCircles.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
- ➡ Software management

Lecture 8   42

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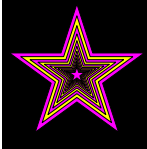
Each task becomes a function that can be implemented and tested independently

Lecture 8   43

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 8   44

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 8   45

To <u>implement</u> 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 8   46

Software Management

<u>Today:</u>

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 8   48

Software Management

<u>During this year</u> :

You write software that makes extensive use of

EPerimeter(a,b)

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

Lecture 8   49

Software Management

<u>Next year</u>:

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 8   50

Script vs. Function

- 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

- 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

Lecture 8   52

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;
```

A: -3   B: 3   C: error

Lecture 8   55