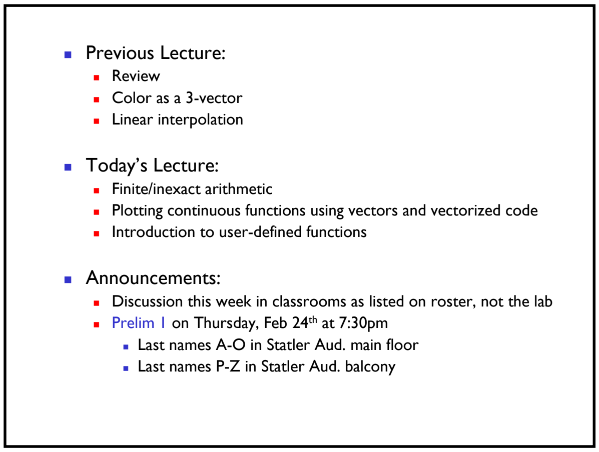
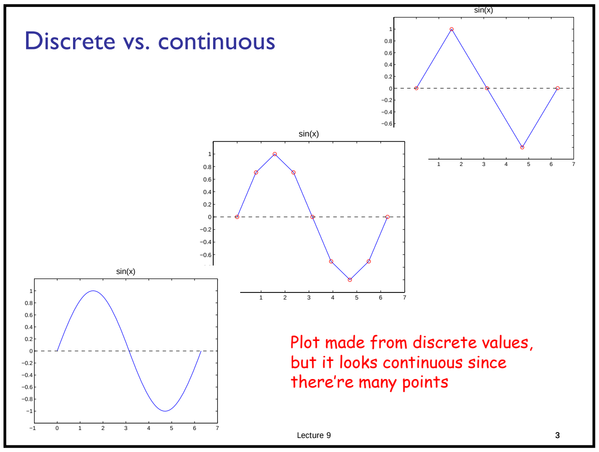
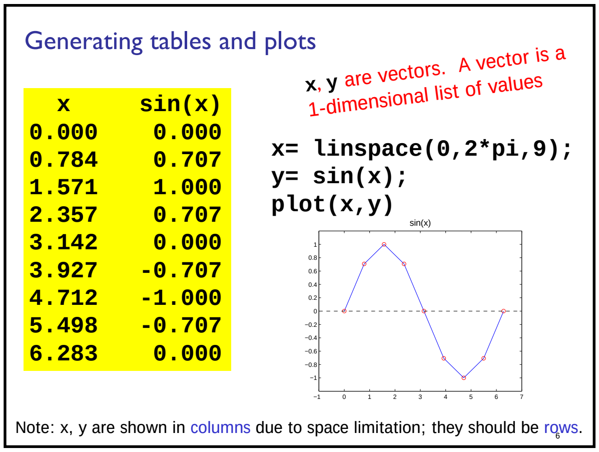


- Previous Lecture:
  - Review
  - Color as a 3-vector
  - Linear interpolation
- Today's Lecture:
  - Finite/inexact arithmetic
  - Plotting continuous functions using vectors and vectorized code
  - Introduction to user-defined functions
- Announcements:
  - Discussion this week in classrooms as listed on roster, not the lab
  - Prelim I on Thursday, Feb 24th at 7:30pm
    - Last names A-O in Statler Aud. main floor
    - Last names P-Z in Statler Aud. balcony



Discrete vs. continuous

Plot made from discrete values, but it looks continuous since there're many points



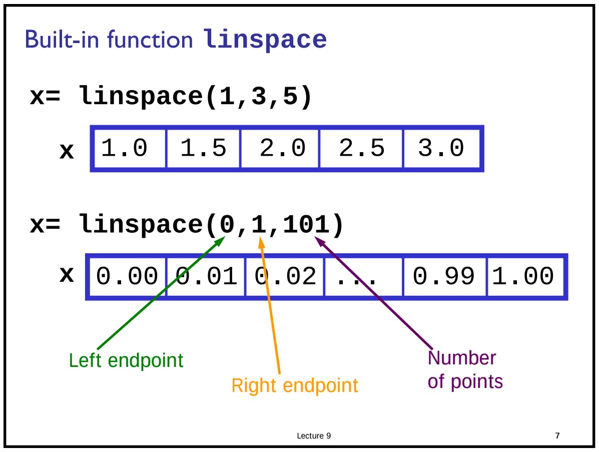
Generating tables and plots

x	sin(x)
0.000	0.000
0.784	0.707
1.571	1.000
2.357	0.707
3.142	0.000
3.927	-0.707
4.712	-1.000
5.498	-0.707
6.283	0.000

x, y are vectors. A vector is a 1-dimensional list of values

```
x= linspace(0,2*pi,9);
y= sin(x);
plot(x,y)
```

Note: x, y are shown in columns due to space limitation; they should be rows.



Built-in function `linspace`

```
x= linspace(1,3,5)
```

x

1.0	1.5	2.0	2.5	3.0

```
x= linspace(0,1,101)
```

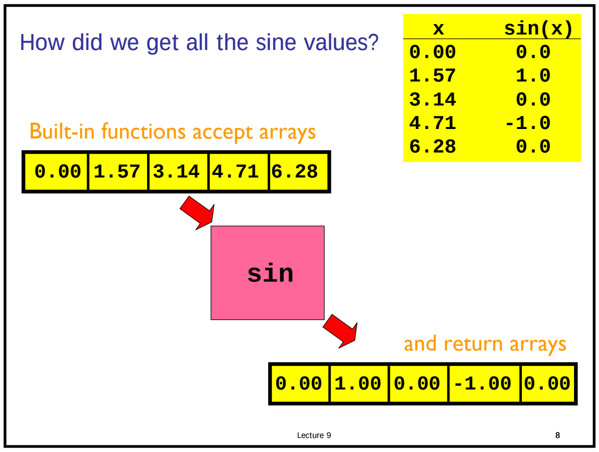
x

0.00	0.01	0.02	. . .	0.99	1.00

Left endpoint

Right endpoint

Number of points



How did we get all the sine values?

x	sin(x)
0.00	0.0
1.57	1.0
3.14	0.0
4.71	-1.0
6.28	0.0

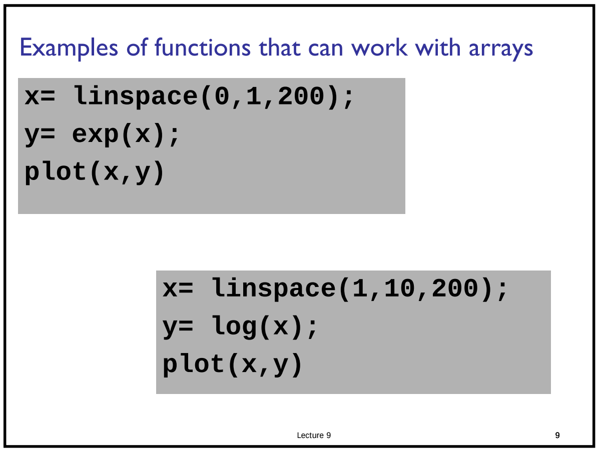
Built-in functions accept arrays

0.00	1.57	3.14	4.71	6.28

sin

and return arrays

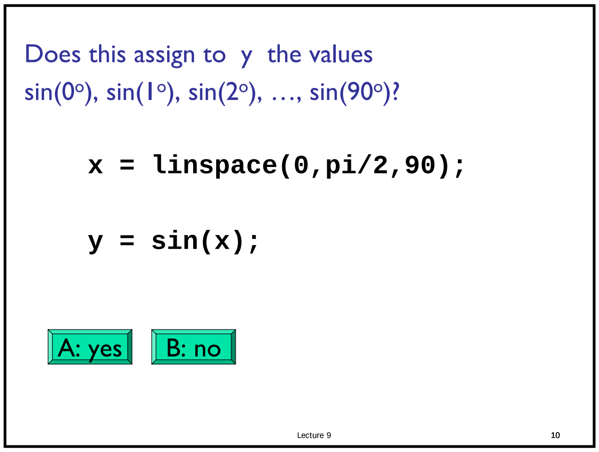
0.00	1.00	0.00	-1.00	0.00



Examples of functions that can work with arrays

```
x= linspace(0,1,200);
y= exp(x);
plot(x,y)
```

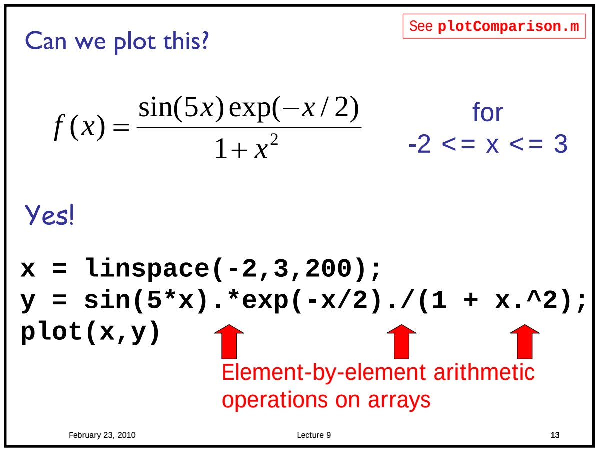
```
x= linspace(1,10,200);
y= log(x);
plot(x,y)
```



Does this assign to y the values sin(0°), sin(1°), sin(2°), …, sin(90°)?

```
x = linspace(0,pi/2,90);

y = sin(x);
```

A: yes B: no



Can we plot this?

See `plotComparison.m`

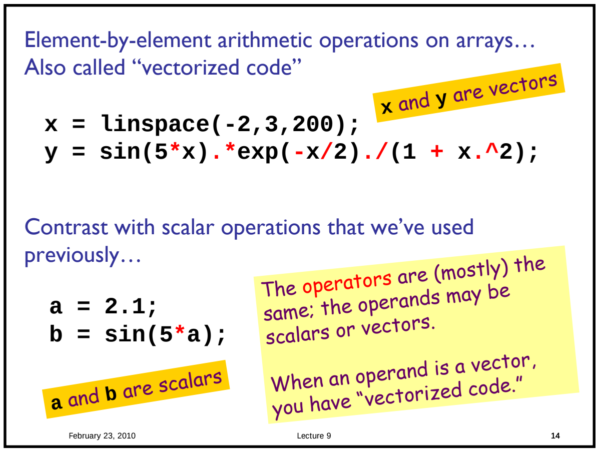
$$f(x)=\frac{\sin(5x)\exp(-x/2)}{1+x^2}$$

for -2 <= x <= 3

Yes!

```
x = linspace(-2,3,200);
y = sin(5*x).*exp(-x/2)./(1 + x.^2);
plot(x,y)
```

Element-by-element arithmetic operations on arrays



Element-by-element arithmetic operations on arrays… Also called "vectorized code"

```
x = linspace(-2,3,200);
y = sin(5*x).*exp(-x/2)./(1 + x.^2);
```

x and **y** are vectors

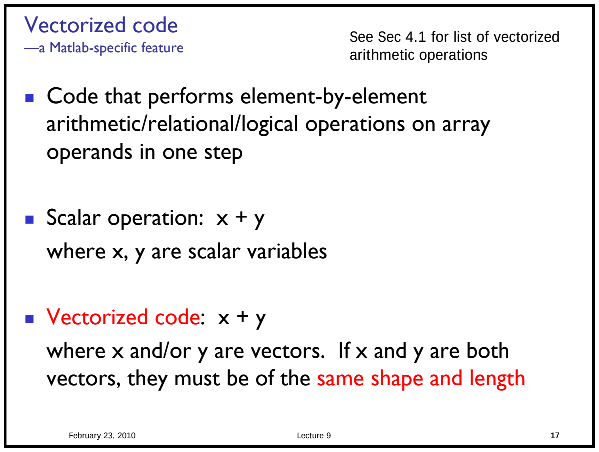
Contrast with scalar operations that we've used previously…

```
a = 2.1;
b = sin(5*a);
```

a and **b** are scalars

The operators are (mostly) the same; the operands may be scalars or vectors.

When an operand is a vector, you have "vectorized code."



Vectorized code

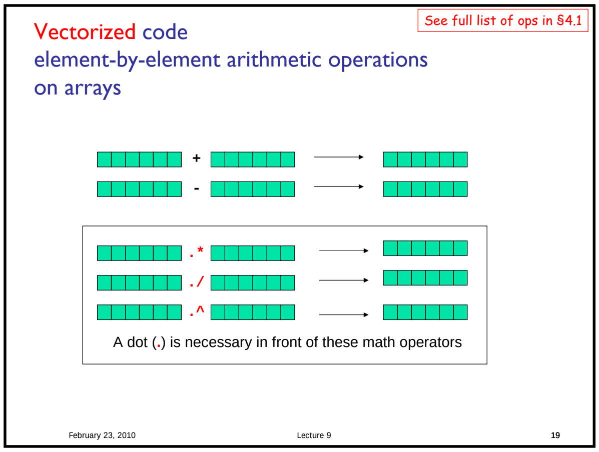
—a Matlab-specific feature

See Sec 4.1 for list of vectorized arithmetic operations

- Code that performs element-by-element arithmetic/relational/logical operations on array operands in one step
- Scalar operation: x + y

  where x, y are scalar variables
- Vectorized code: x + y

  where x and/or y are vectors. If x and y are both vectors, they must be of the same shape and length



Vectorized code

element-by-element arithmetic operations on arrays

See full list of ops in §4.1

🟩🟩🟩🟩🟩🟩 + 🟩🟩🟩🟩🟩🟩 → 🟩🟩🟩🟩🟩🟩

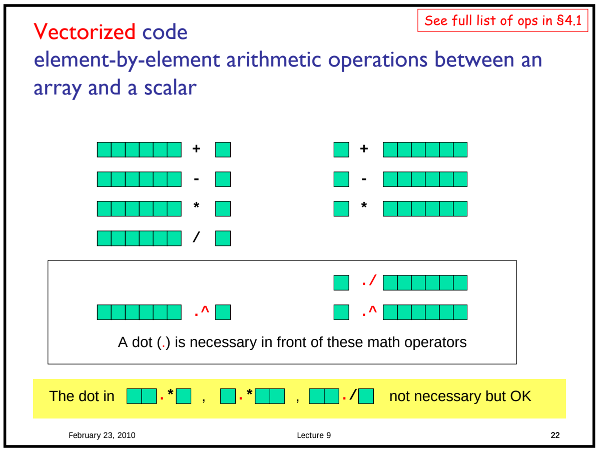
🟩🟩🟩🟩🟩🟩 - 🟩🟩🟩🟩🟩🟩 → 🟩🟩🟩🟩🟩🟩

🟩🟩🟩🟩🟩🟩 .* 🟩🟩🟩🟩🟩🟩 → 🟩🟩🟩🟩🟩🟩

🟩🟩🟩🟩🟩🟩 ./ 🟩🟩🟩🟩🟩🟩 → 🟩🟩🟩🟩🟩🟩

🟩🟩🟩🟩🟩🟩 .^ 🟩🟩🟩🟩🟩🟩 → 🟩🟩🟩🟩🟩🟩

A dot (.) is necessary in front of these math operators



Vectorized code

element-by-element arithmetic operations between an array and a scalar

See full list of ops in §4.1

🟩🟩🟩🟩🟩🟩 + 🟩 🟩 + 🟩🟩🟩🟩🟩🟩

🟩🟩🟩🟩🟩🟩 - 🟩 🟩 - 🟩🟩🟩🟩🟩🟩

🟩🟩🟩🟩🟩🟩 * 🟩 🟩 * 🟩🟩🟩🟩🟩🟩

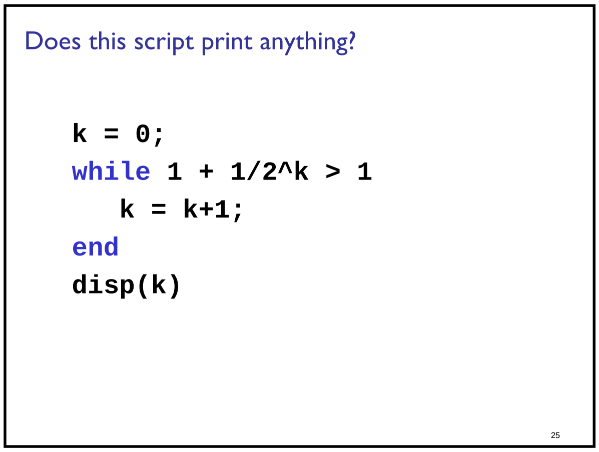
🟩🟩🟩🟩🟩🟩 / 🟩

🟩 ./ 🟩🟩🟩🟩🟩🟩

🟩🟩🟩🟩🟩🟩 .^ 🟩 🟩 .^ 🟩🟩🟩🟩🟩🟩

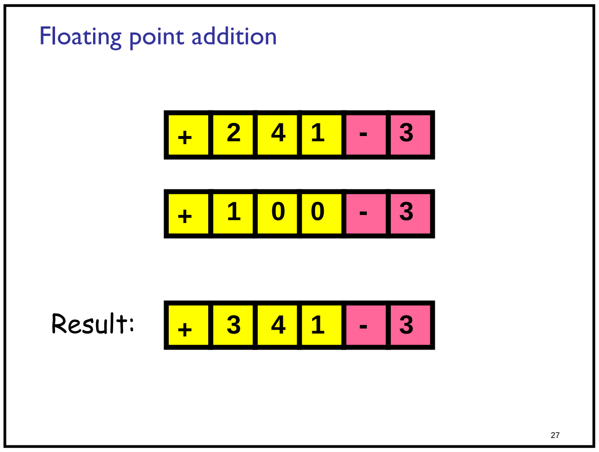
A dot (.) is necessary in front of these math operators

The dot in 🟩🟩.*🟩 , 🟩.*🟩🟩 , 🟩🟩./🟩 not necessary but OK



Does this script print anything?

```
k = 0;
while 1 + 1/2^k > 1
   k = k+1;
end
disp(k)
```



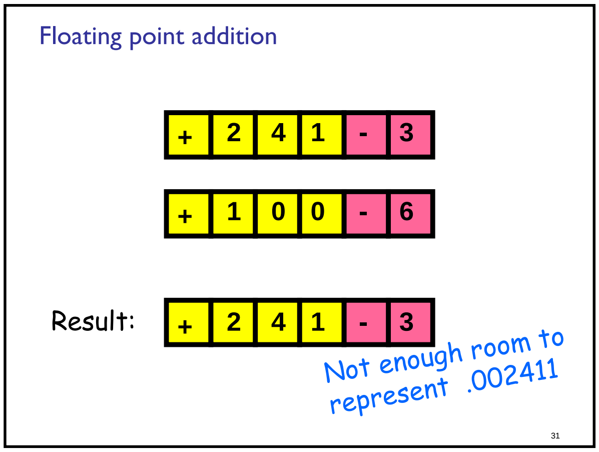
Floating point addition

+	2	4	1	-	3

+	1	0	0	-	3

Result:

+	3	4	1	-	3



Floating point addition

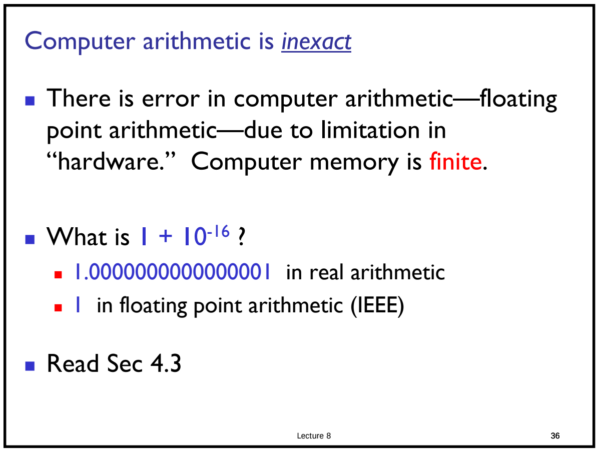
+	2	4	1	-	3

+	1	0	0	-	6

Result:

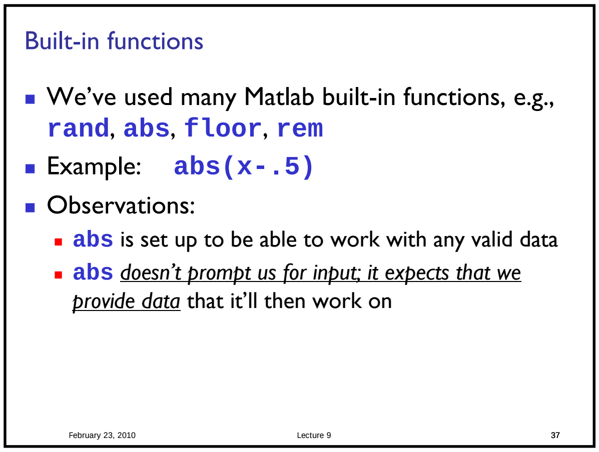
+	2	4	1	-	3

Not enough room to represent .002411



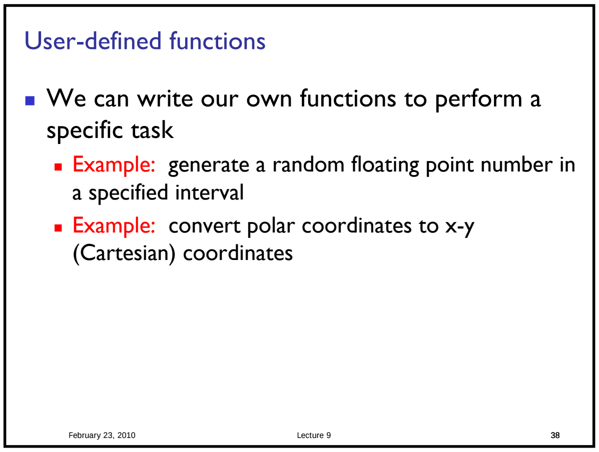
Computer arithmetic is *inexact*

- There is error in computer arithmetic—floating point arithmetic—due to limitation in "hardware." Computer memory is finite.
- What is 1 + 10^{-16} ?
  - 1.0000000000000001 in real arithmetic
  - 1 in floating point arithmetic (IEEE)
- Read Sec 4.3



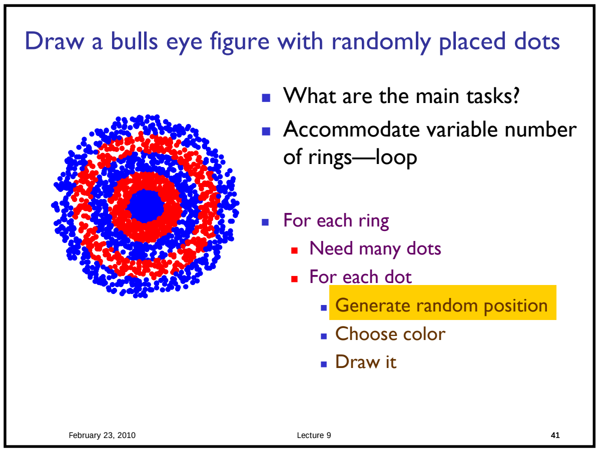
Built-in functions

- 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



User-defined functions

- We can write our own functions to perform a specific task
  - Example: generate a random floating point number in a specified interval
  - Example: convert polar coordinates to x-y (Cartesian) coordinates

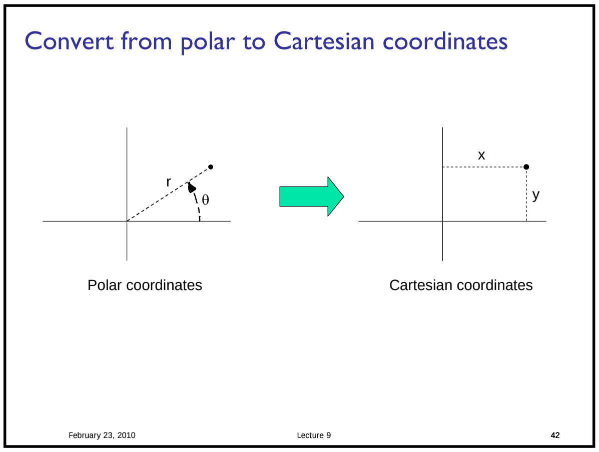


Draw a bulls eye figure with randomly placed dots

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

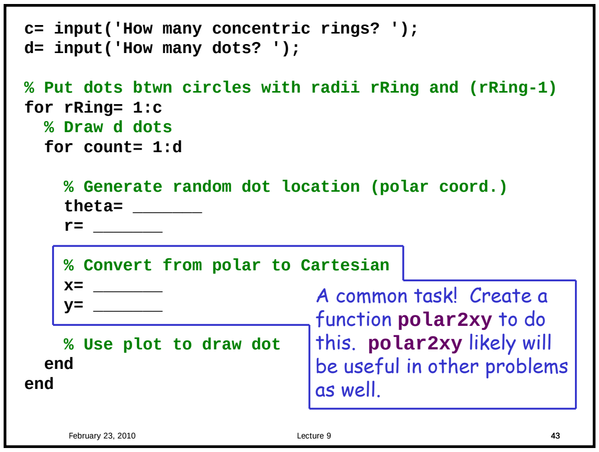
- For each ring
  - Need many dots
  - For each dot
    - Generate random position
    - Choose color
    - Draw it

February 23, 2010 Lecture 9 41



Convert from polar to Cartesian coordinates

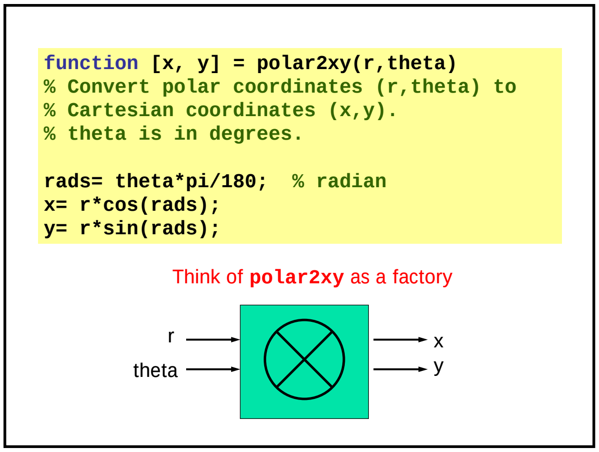
February 23, 2010 Lecture 9 42



```
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
```

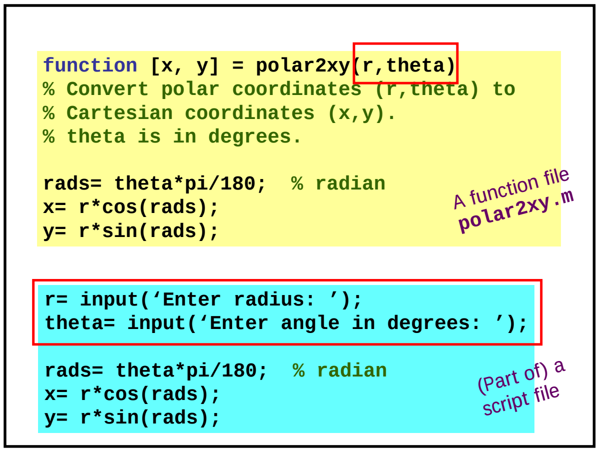
A common task! Create a function **polar2xy** to do this. **polar2xy** likely will be useful in other problems as well.

February 23, 2010 Lecture 9 43



```
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);
```

Think of **polar2xy** as a factory

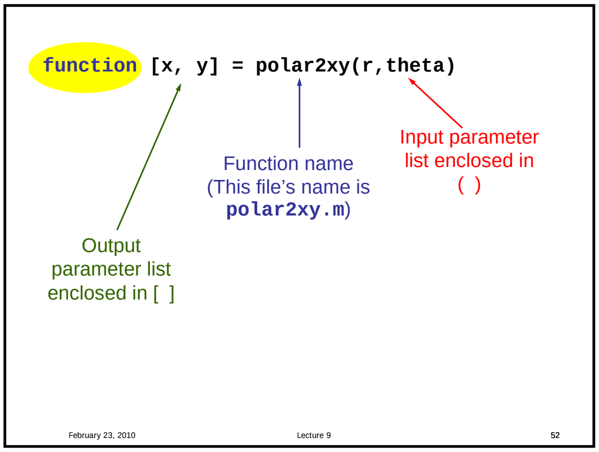


```
function [x, y] = polar2xy(r,theta)
% Convert polar coordinates (r,theta) to
% Cartesian coordinates (x,y).
% theta is in degrees.

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



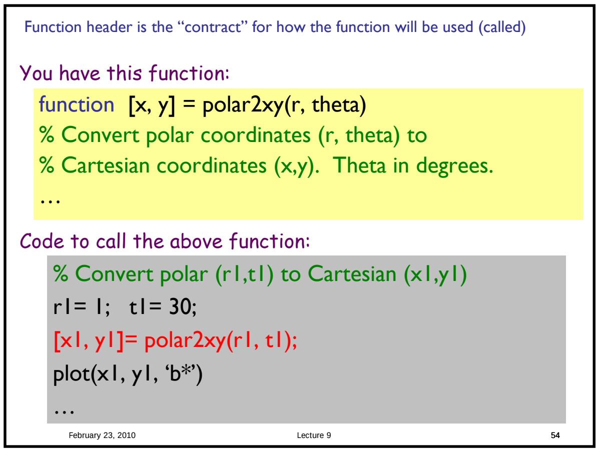
```
function [x, y] = polar2xy(r,theta)
```

Output parameter list enclosed in [ ]

Function name (This file’s name is **polar2xy.m**)

Input parameter list enclosed in ( )

February 23, 2010 Lecture 9 52



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

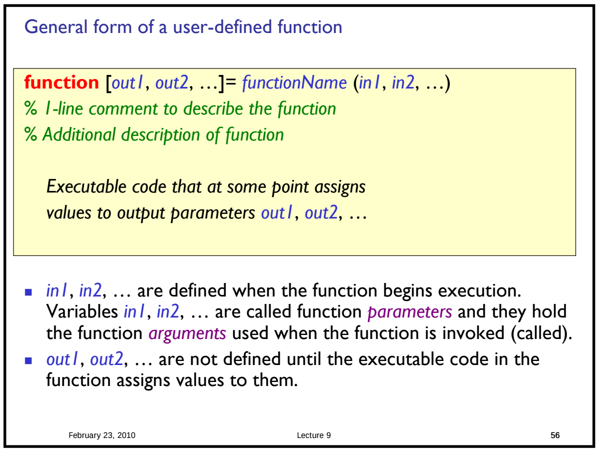
You have this function:

```
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*’)
…
```

February 23, 2010 Lecture 9 54

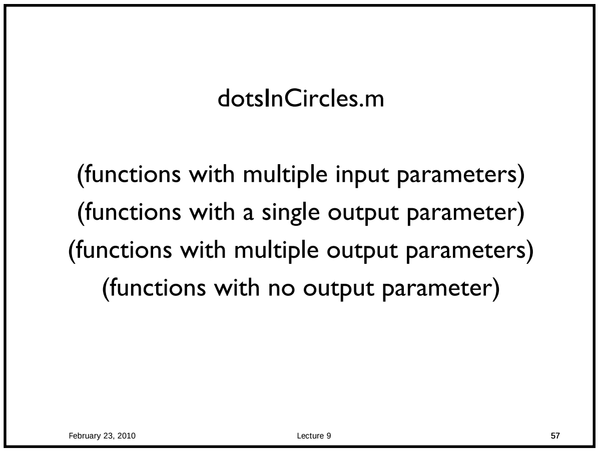


General form of a user-defined function

```
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.

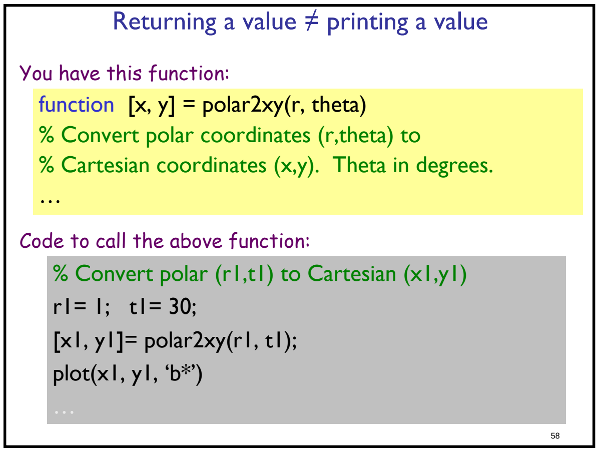
February 23, 2010 Lecture 9 56



dotsInCircles.m

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

February 23, 2010 Lecture 9 57



Returning a value ≠ printing a value

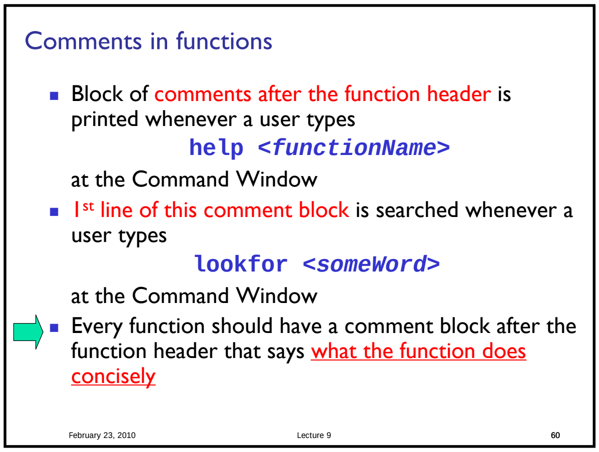
You have this function:

```
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*’)
…
```

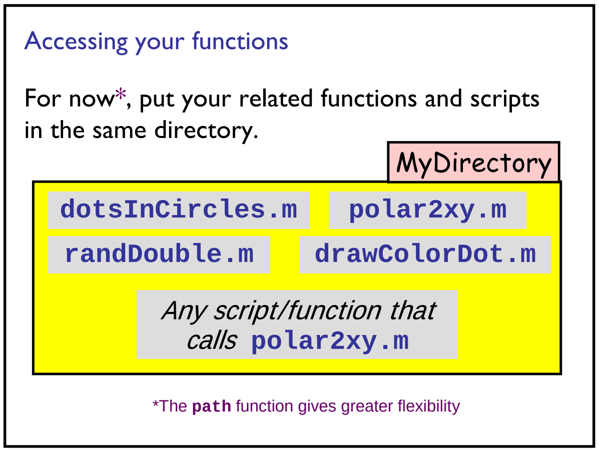
58



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

February 23, 2010 Lecture 9 60



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