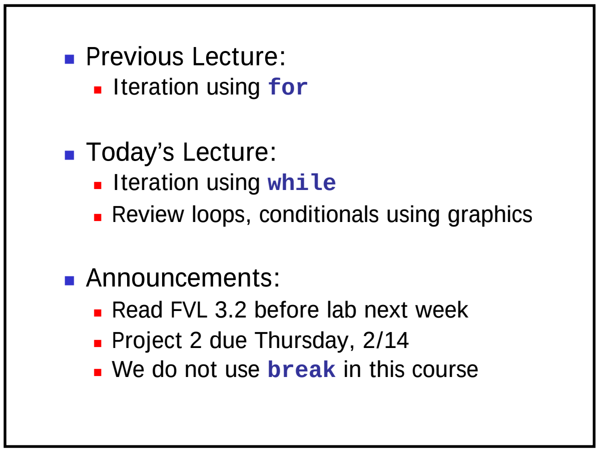
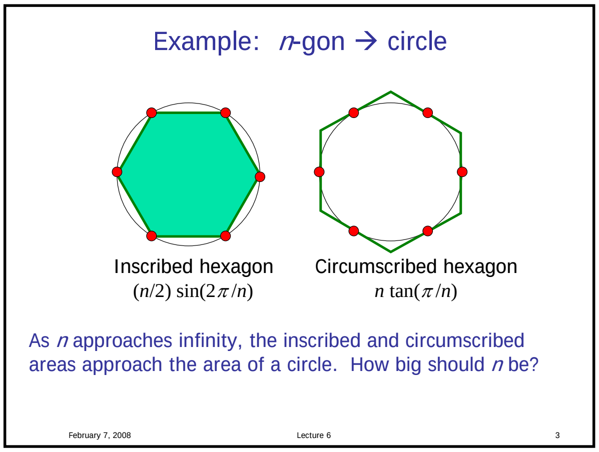


- Previous Lecture:
  - Iteration using **`for`**
- Today's Lecture:
  - Iteration using **`while`**
  - Review loops, conditionals using graphics
- Announcements:
  - Read FVL 3.2 before lab next week
  - Project 2 due Thursday, 2/14
  - We do not use **`break`** in this course

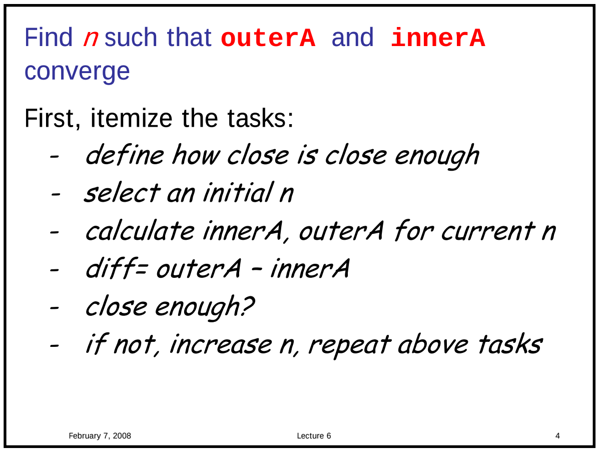


Example: *n*-gon → circle

Inscribed hexagon $(n/2)\sin(2\pi/n)$

Circumscribed hexagon $n\tan(\pi/n)$

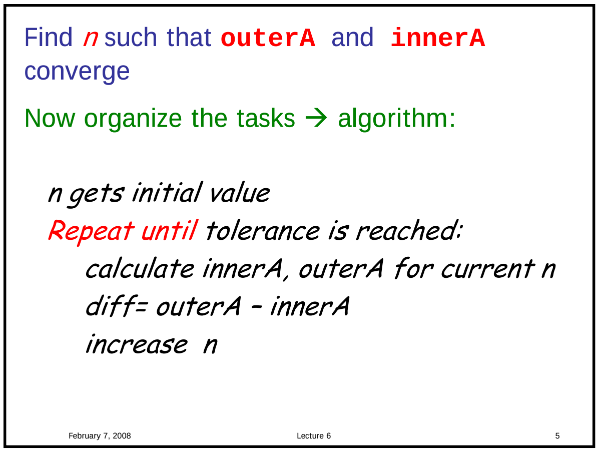
As *n* approaches infinity, the inscribed and circumscribed areas approach the area of a circle. How big should *n* be?



Find *n* such that `outerA` and `innerA` converge

First, itemize the tasks:

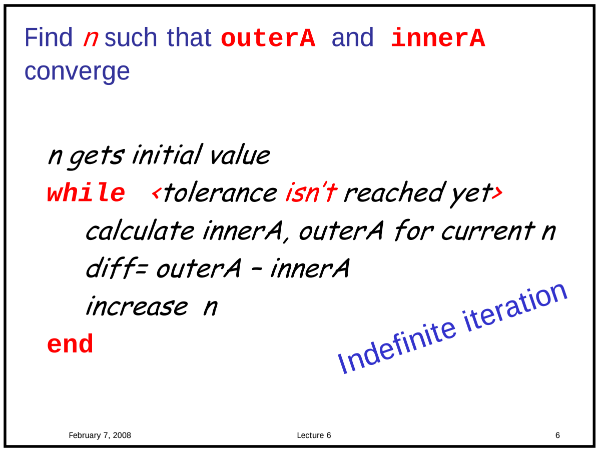
- *define how close is close enough*
- *select an initial n*
- *calculate innerA, outerA for current n*
- *diff= outerA – innerA*
- *close enough?*
- *if not, increase n, repeat above tasks*



Find *n* such that `outerA` and `innerA` converge

Now organize the tasks → algorithm:

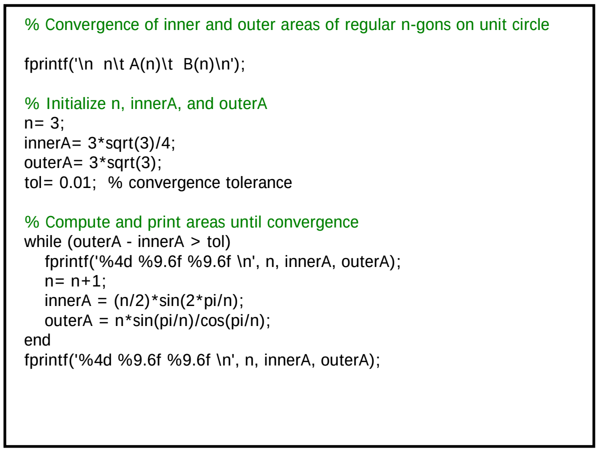
```
n gets initial value
Repeat until tolerance is reached:
    calculate innerA, outerA for current n
    diff= outerA – innerA
    increase  n
```



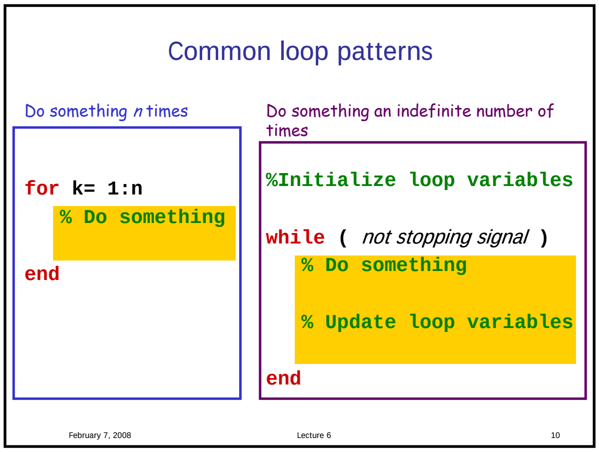
Find *n* such that `outerA` and `innerA` converge

```
n gets initial value
while  <tolerance isn't reached yet>
    calculate innerA, outerA for current n
    diff= outerA – innerA
    increase  n
end
```

Indefinite iteration



```
% Convergence of inner and outer areas of regular n-gons on unit circle

fprintf('\n  n\t A(n)\t  B(n)\n');

% Initialize n, innerA, and outerA
n= 3;
innerA= 3*sqrt(3)/4;
outerA= 3*sqrt(3);
tol= 0.01;  % convergence tolerance

% Compute and print areas until convergence
while (outerA - innerA > tol)
    fprintf('%4d %9.6f %9.6f \n', n, innerA, outerA);
    n= n+1;
    innerA = (n/2)*sin(2*pi/n);
    outerA = n*sin(pi/n)/cos(pi/n);
end
fprintf('%4d %9.6f %9.6f \n', n, innerA, outerA);
```



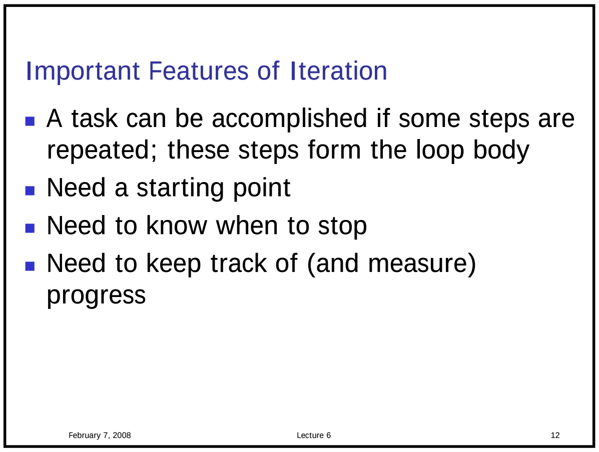
Common loop patterns

Do something *n* times

```
for k= 1:n
   % Do something
end
```

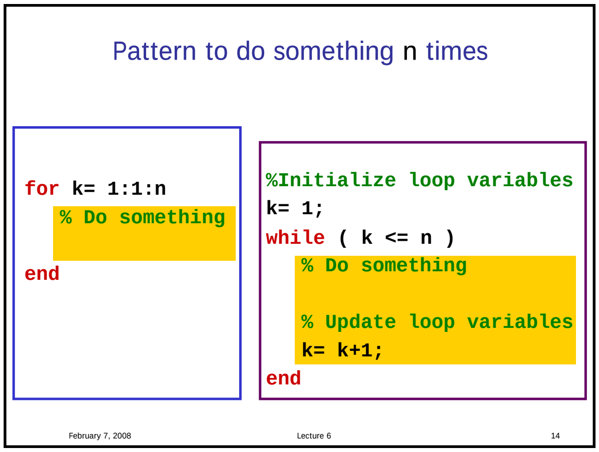
Do something an indefinite number of times

```
%Initialize loop variables

while ( not stopping signal )
   % Do something

   % Update loop variables
end
```



Important Features of Iteration

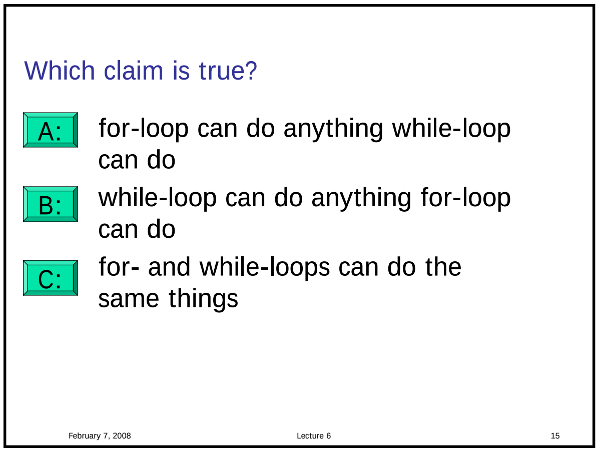
- A task can be accomplished if some steps are repeated; these steps form the loop body
- Need a starting point
- Need to know when to stop
- Need to keep track of (and measure) progress



Pattern to do something n times

```
for k= 1:1:n
   % Do something
end
```

```
%Initialize loop variables
k= 1;
while ( k <= n )
   % Do something

   % Update loop variables
   k= k+1;
end
```

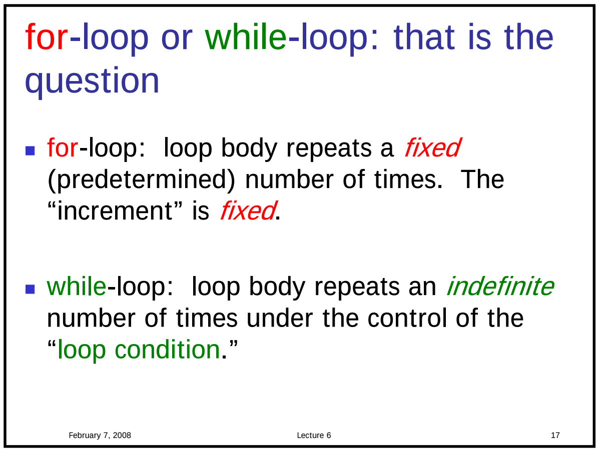


Which claim is true?

A: for-loop can do anything while-loop can do

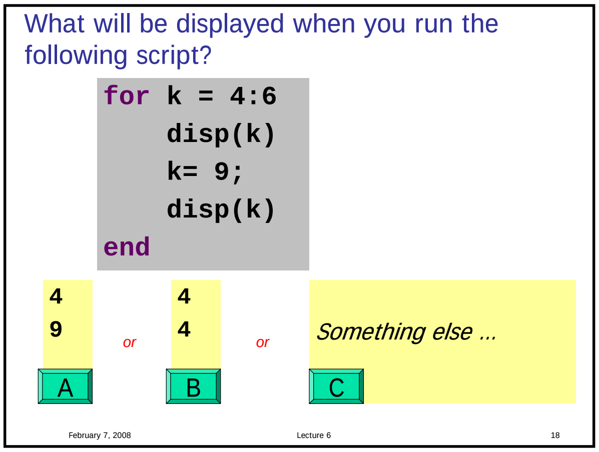
B: while-loop can do anything for-loop can do

C: for- and while-loops can do the same things



for-loop or while-loop: that is the question

- for-loop: loop body repeats a *fixed* (predetermined) number of times. The “increment” is *fixed*.
- while-loop: loop body repeats an *indefinite* number of times under the control of the “loop condition.”



What will be displayed when you run the following script?

```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```

4
9

A

or

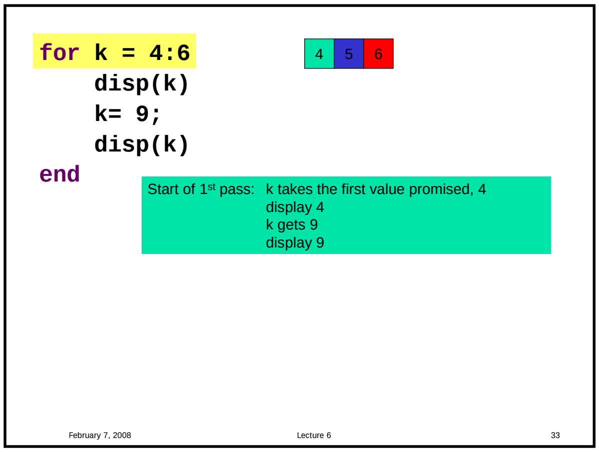
4
4

B

or

Something else …

C

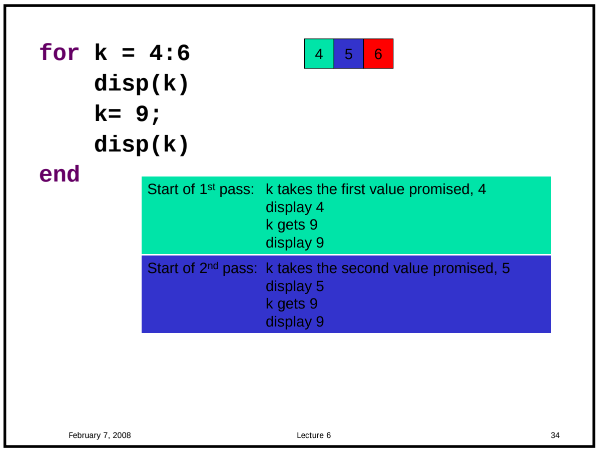


```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```

4	5	6

Start of 1st pass: k takes the first value promised, 4
display 4
k gets 9
display 9

February 7, 2008   Lecture 6   33



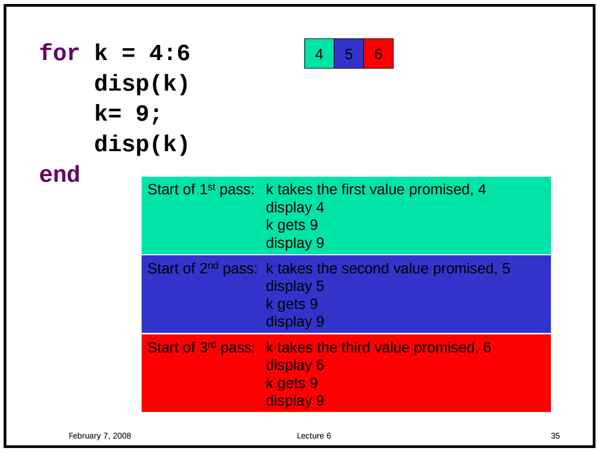
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```

4	5	6

Start of 1st pass: k takes the first value promised, 4
display 4
k gets 9
display 9

Start of 2nd pass: k takes the second value promised, 5
display 5
k gets 9
display 9

February 7, 2008   Lecture 6   34



```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```

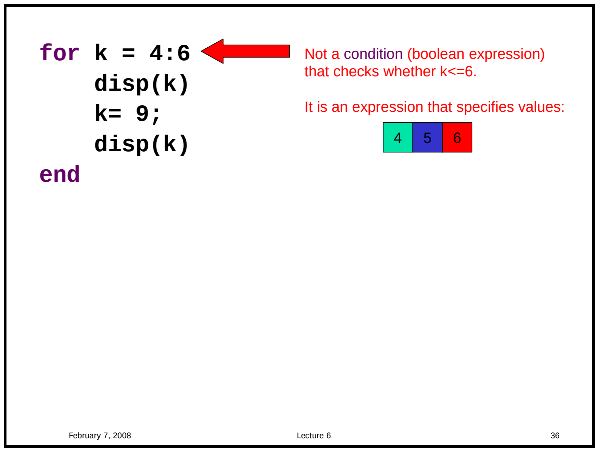
4	5	6

Start of 1st pass: k takes the first value promised, 4
display 4
k gets 9
display 9

Start of 2nd pass: k takes the second value promised, 5
display 5
k gets 9
display 9

Start of 3rd pass: k takes the third value promised, 6
display 6
k gets 9
display 9

February 7, 2008   Lecture 6   35



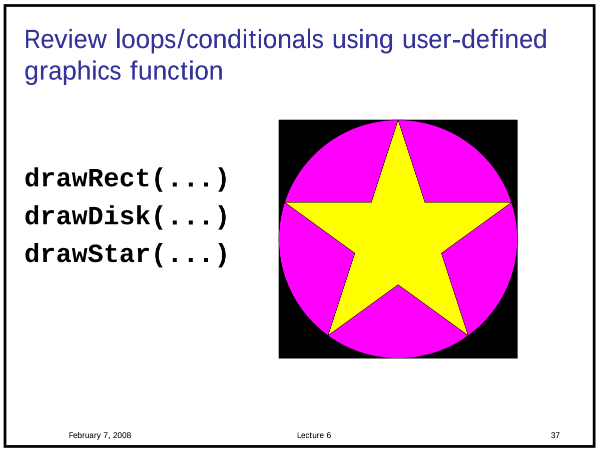
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```

Not a condition (boolean expression) that checks whether k<=6.

It is an expression that specifies values:

4	5	6

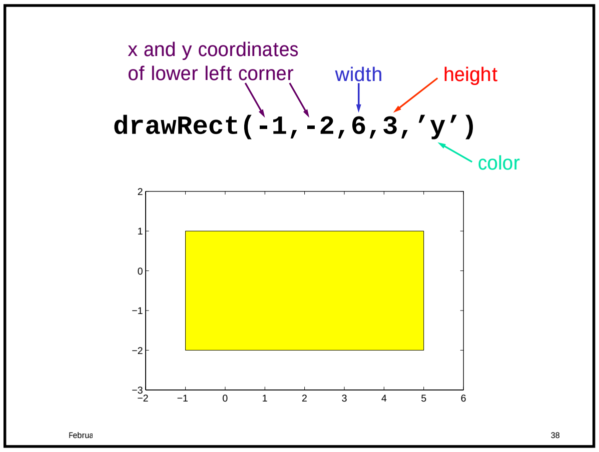
February 7, 2008   Lecture 6   36



Review loops/conditionals using user-defined graphics function

```
drawRect(...)
drawDisk(...)
drawStar(...)
```

February 7, 2008   Lecture 6   37



x and y coordinates of lower left corner

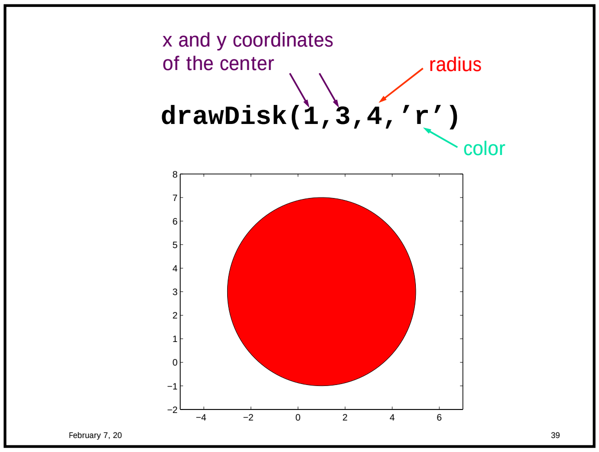
width

height

```
drawRect(-1,-2,6,3,'y')
```

color

Februa   38



x and y coordinates of the center

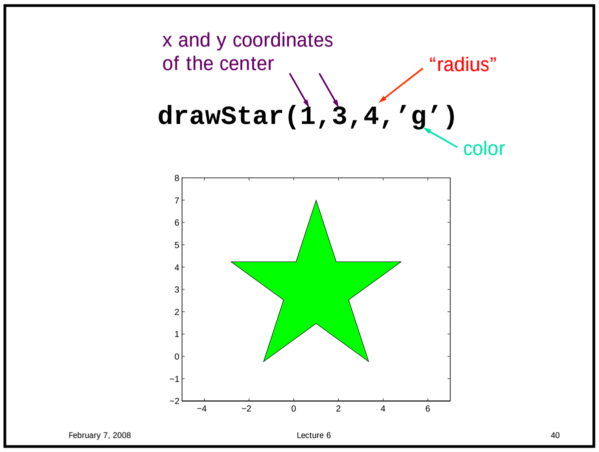
radius

`drawDisk(1,3,4,'r')`

color

February 7, 20

39



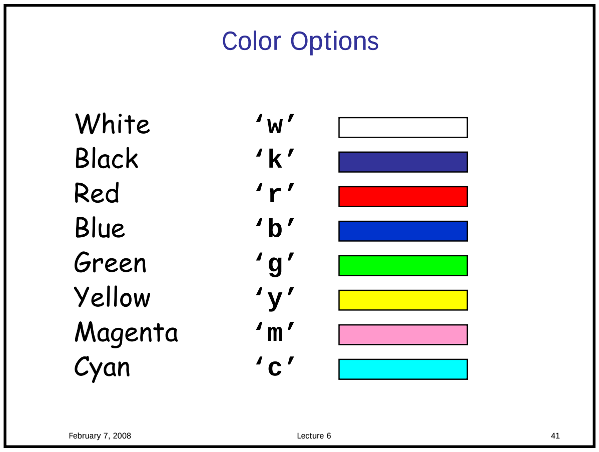
x and y coordinates of the center

"radius"

`drawStar(1,3,4,'g')`

color

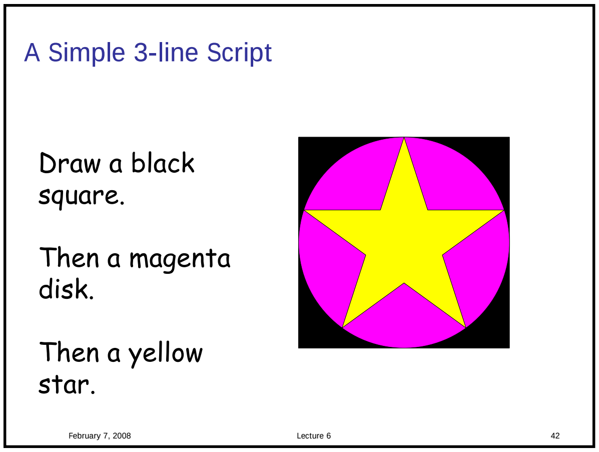
February 7, 2008 Lecture 6 40



Color Options

White	'w'	□
Black	'k'	■
Red	'r'	🟥
Blue	'b'	🟦
Green	'g'	🟩
Yellow	'y'	🟨
Magenta	'm'	🟪
Cyan	'c'	🟦

February 7, 2008 Lecture 6 41



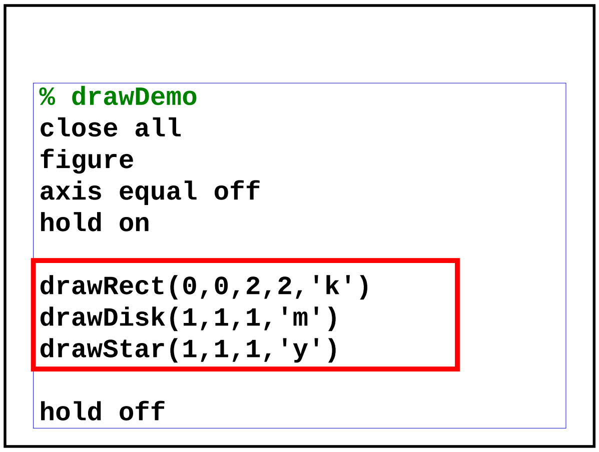
A Simple 3-line Script

Draw a black square.

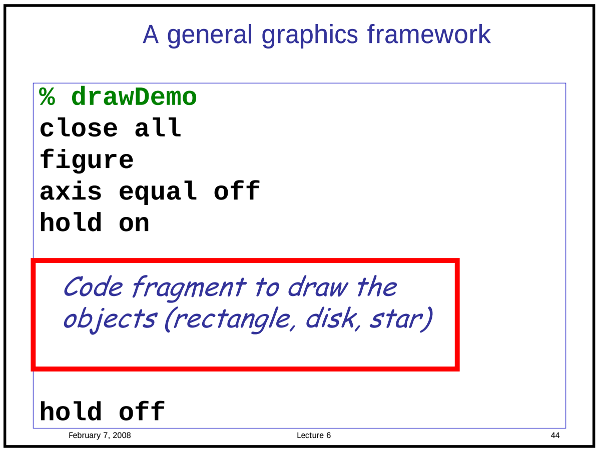
Then a magenta disk.

Then a yellow star.

February 7, 2008 Lecture 6 42



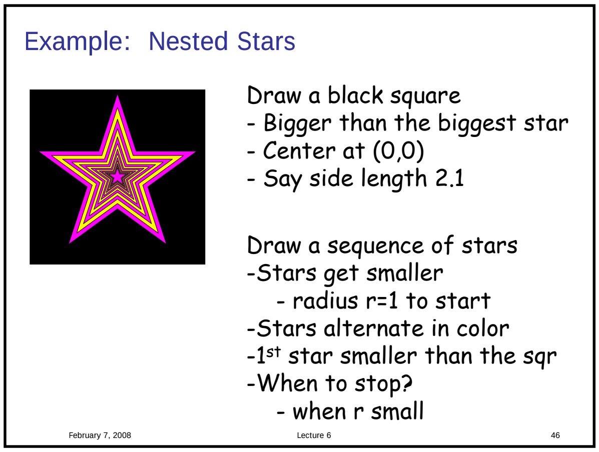
```
% drawDemo
close all
figure
axis equal off
hold on

drawRect(0,0,2,2,'k')
drawDisk(1,1,1,'m')
drawStar(1,1,1,'y')

hold off
```



A general graphics framework

```
% drawDemo
close all
figure
axis equal off
hold on

  Code fragment to draw the
  objects (rectangle, disk, star)

hold off
```

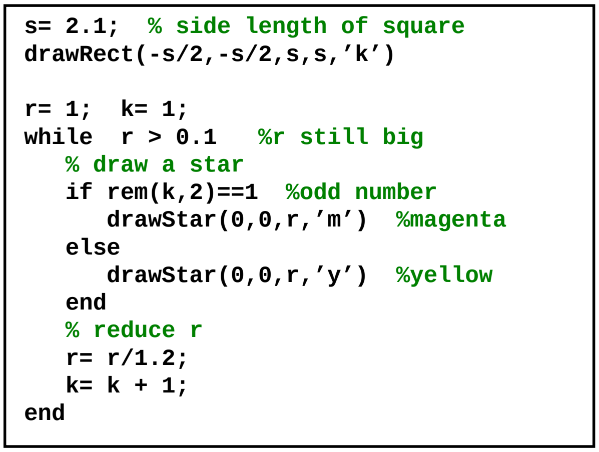
February 7, 2008 Lecture 6 44



Example: Nested Stars

Draw a black square
- Bigger than the biggest star
- Center at (0,0)
- Say side length 2.1

Draw a sequence of stars
-Stars get smaller
   - radius r=1 to start
-Stars alternate in color
-1st star smaller than the sqr
-When to stop?
   - when r small



```
s= 2.1;  % side length of square
drawRect(-s/2,-s/2,s,s,’k’)

r= 1;  k= 1;
while  r > 0.1   %r still big
   % draw a star
   if rem(k,2)==1  %odd number
      drawStar(0,0,r,’m’)  %magenta
   else
      drawStar(0,0,r,’y’)  %yellow
   end
   % reduce r
   r= r/1.2;
   k= k + 1;
end
```