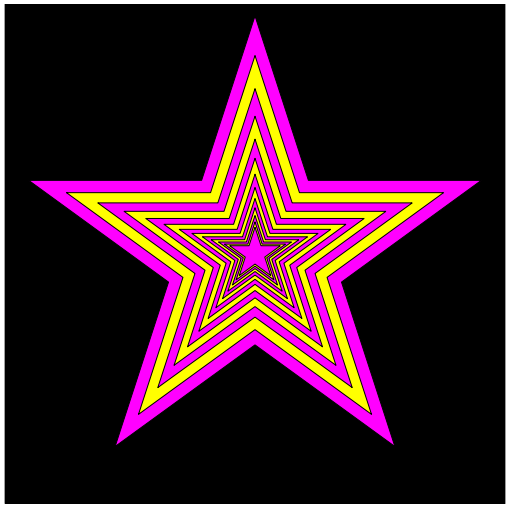
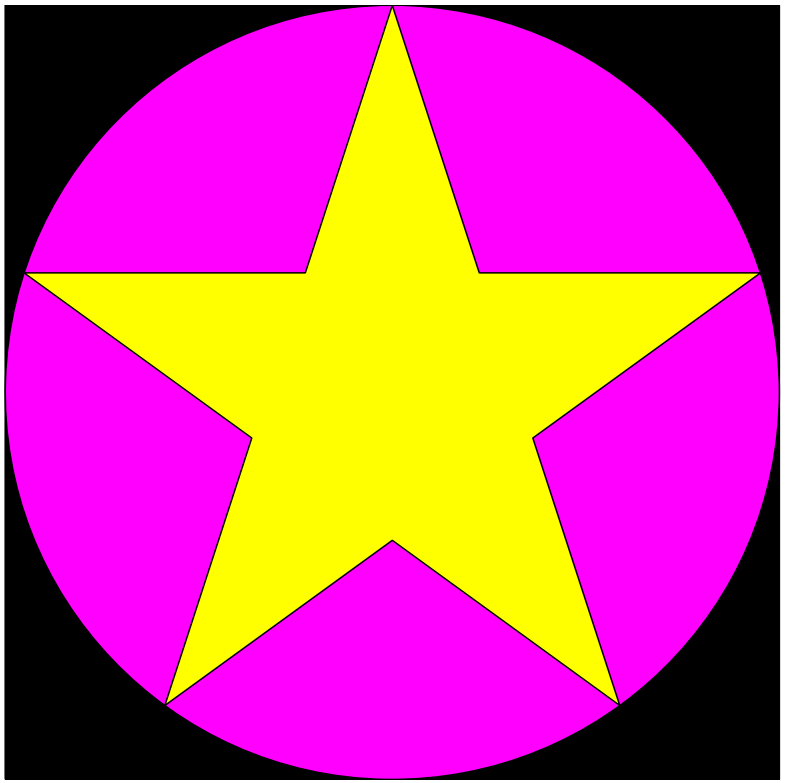
- Previous Lecture:
  - Iteration using `while`
- Today's Lecture:
  - Developing algorithms
  - Nested loops
- Announcements:
  - Project 2 due 9/24 (Thurs) at 11pm
  - CS major Q&A sessions, both at Upson 315
    - Thurs, 9/17, 3-4pm
    - Mon, 9/21, 10:30-11:30am
  - Engineering Majors Fair on Nov. 2

Example: Nested Stars



Review loops/conditionals using user-defined graphics function

```
DrawRect(...)
DrawDisk(...)
DrawStar(...)
```



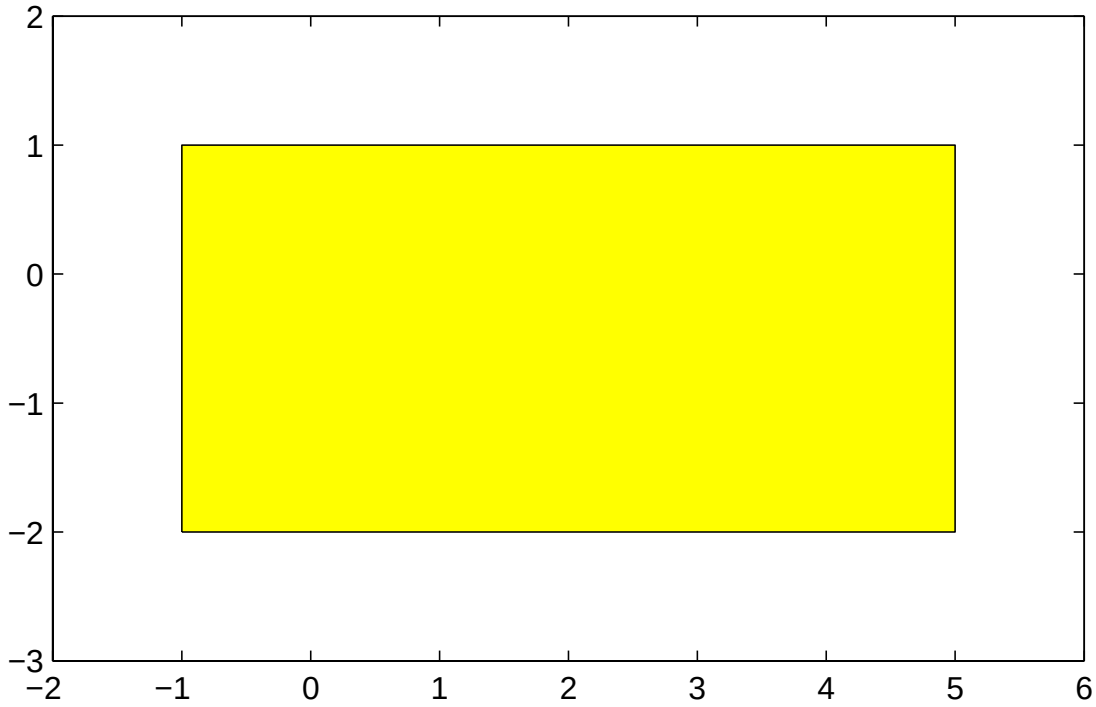
x and y coordinates
of lower left corner

width

height

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

color

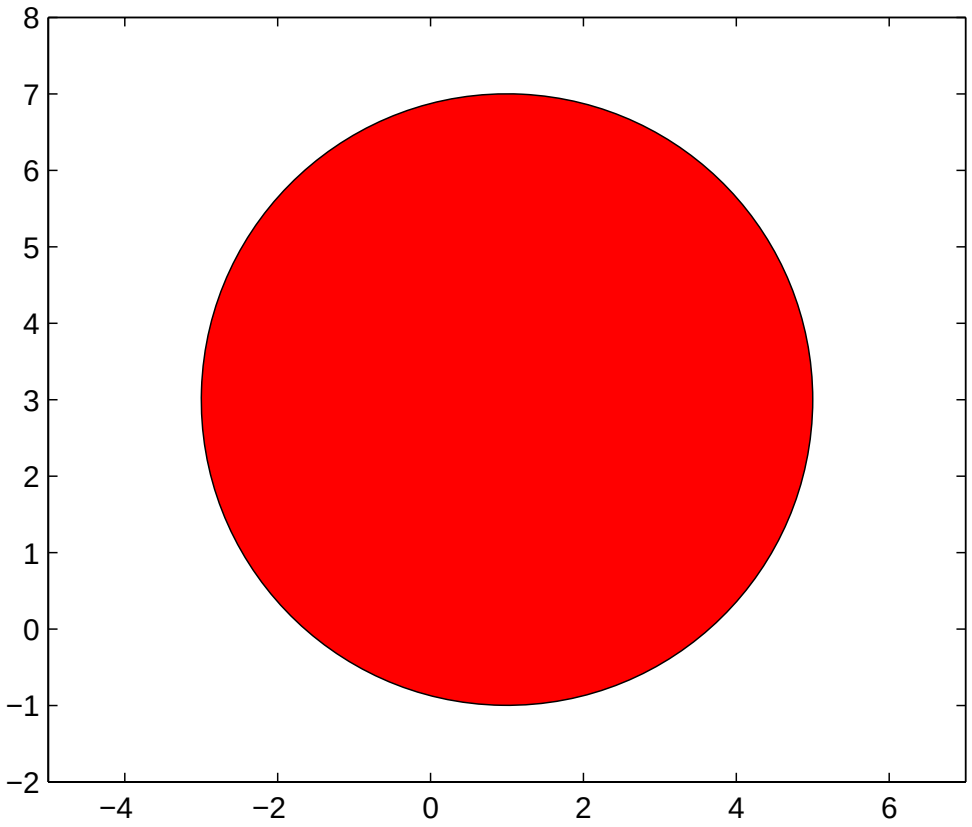


x and y coordinates
of the center

radius

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

color

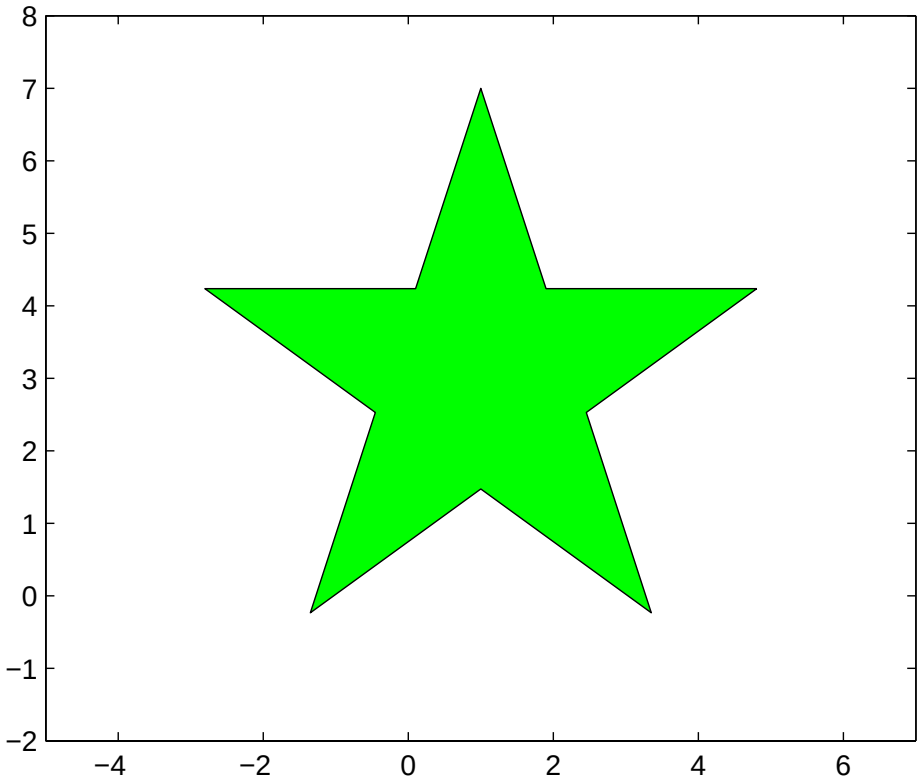


x and y coordinates
of the center

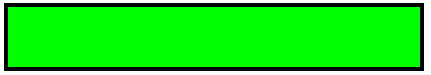
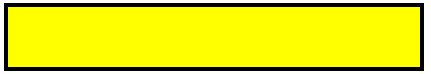
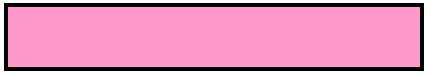
"radius"

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

color



Color Options

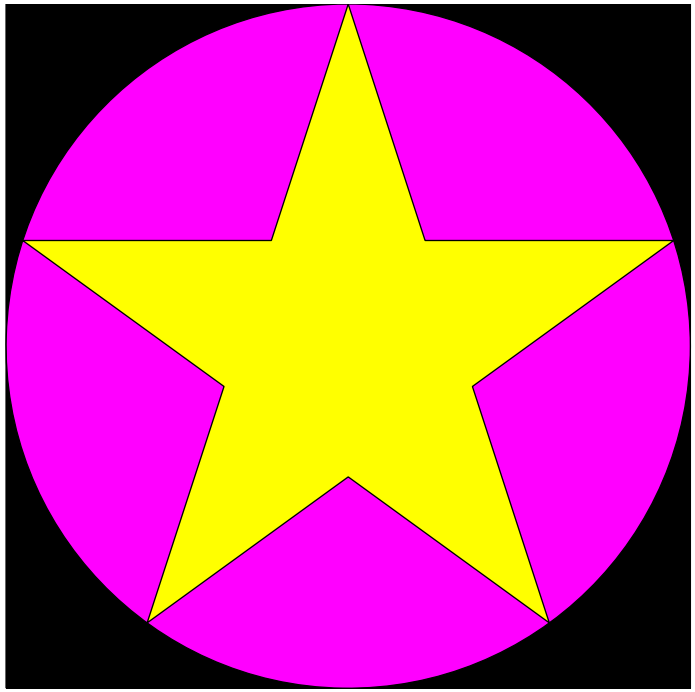
White	`'w'`	
Black	`'k'`	
Red	`'r'`	
Blue	`'b'`	
Green	`'g'`	
Yellow	`'y'`	
Magenta	`'m'`	
Cyan	`'c'`	

A Simple 3-line Script to …

draw a black square;

then draw a magenta disk;

then draw a yellow star.



```
% 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
```

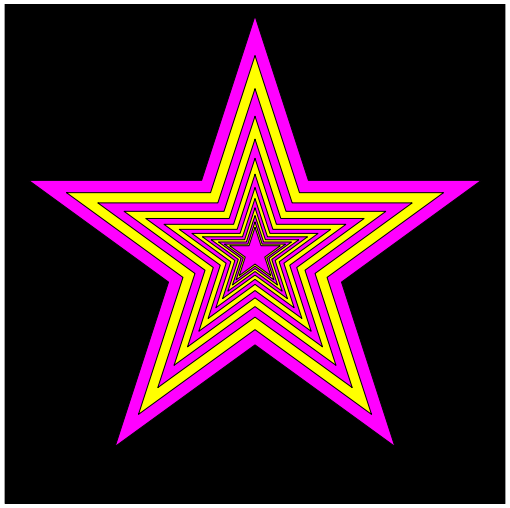
A general graphics framework

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

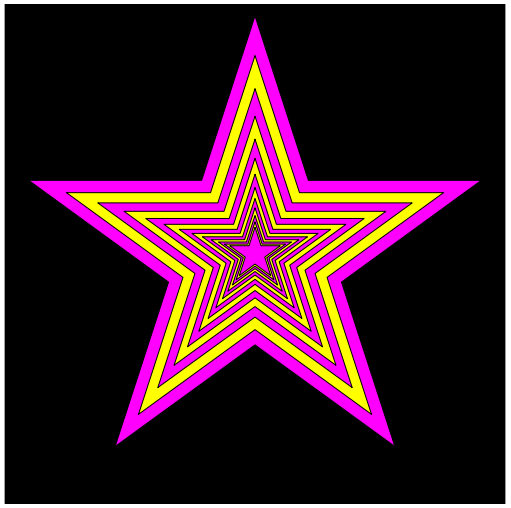
Code fragment to draw the objects (rectangle, disk, star)

```
hold off
```

Example: Nested Stars



Example: Nested Stars

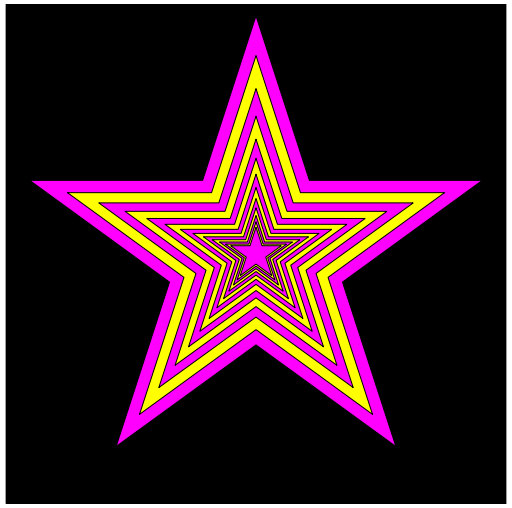


Draw a black square
-Bigger than the biggest star
(at least 2 times radius of star)
- Center at (0,0)

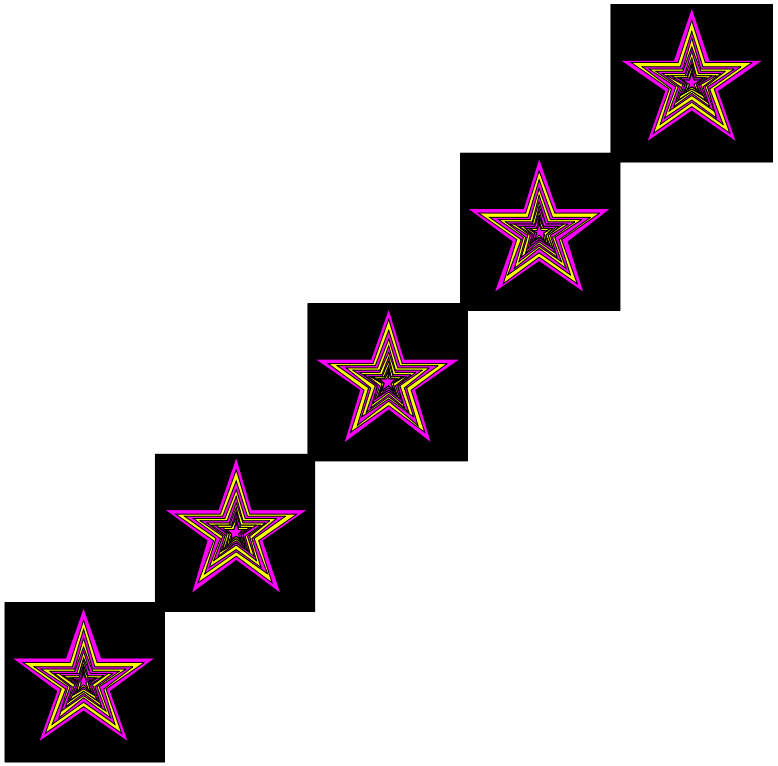
Draw a sequence of stars
-Stars alternate in color
-Stars get smaller
    - radius r=1 to start
-1st star smaller than the sqr
-When to stop?
    - when r is small

nestedStars.m

Knowing how to draw



How difficult is it to draw



Pattern for doing something *n* times

```
n= _____
for k= 1:n
    % code to do
    % that something
end
```

```
x= 0; y= 0; % figure centered at (0,0)

s= 2.1;   % side length of square
DrawRect(x-s/2,y-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(x,y,r,'m')  %magenta
   else
      DrawStar(x,y,r,'y')  %yellow
   end
   % reduce r
   r= r/1.2;
   k= k + 1;
end
```

```
for c = 0:2:8
    x= c; y= c; % figure centered at (c,c)
    s= 2.1;   % side length of square
    DrawRect(x-s/2,y-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(x,y,r,'m')  %magenta
       else
          DrawStar(x,y,r,'y')  %yellow
       end
       % reduce r
       r= r/1.2;
       k= k + 1;
    end
end
```

Pattern for doing something *n* times

```
n= _____
for k= 1:n
    % code to do
    % that something
end
```

Example: Are they prime?

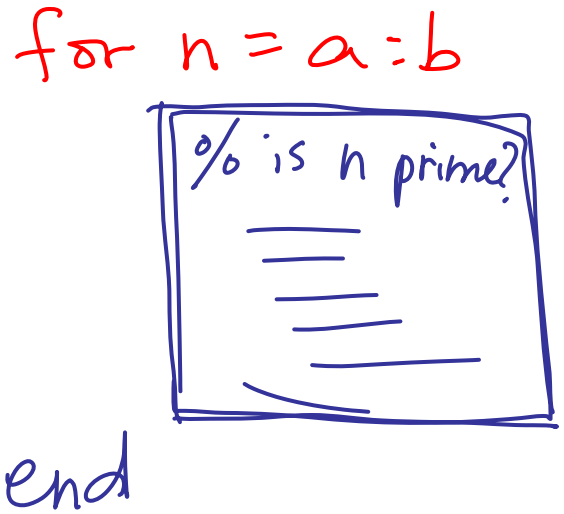
- Given integers a and b, write a program that lists all the prime numbers in the range [a, b].

- Assume a>1, b>1 and a<b.

Example: Are they prime?

- Given integers a and b, write a program that lists all the prime numbers in the range [a, b].

- Assume a>1, b>1 and a<b.



Example: Are they prime?
Subproblem: Is it prime?

- Write a program fragment to determine whether a given integer n is prime, n>1.

- Reminder: rem(x,y) returns the remainder of x divided by y.

Start:

divisor = 2

Repeat:

rem(n, divisor)
divisor = divisor + 1

End:

rem(n, divisor) == 0

divisor ← n ?

```
divisor = 2;

while (rem(n,divisor) ~= 0)

    divisor = divisor + 1;

end

if (divisor == n)

    disp ('prime')

else

    disp ('composite')

end
```

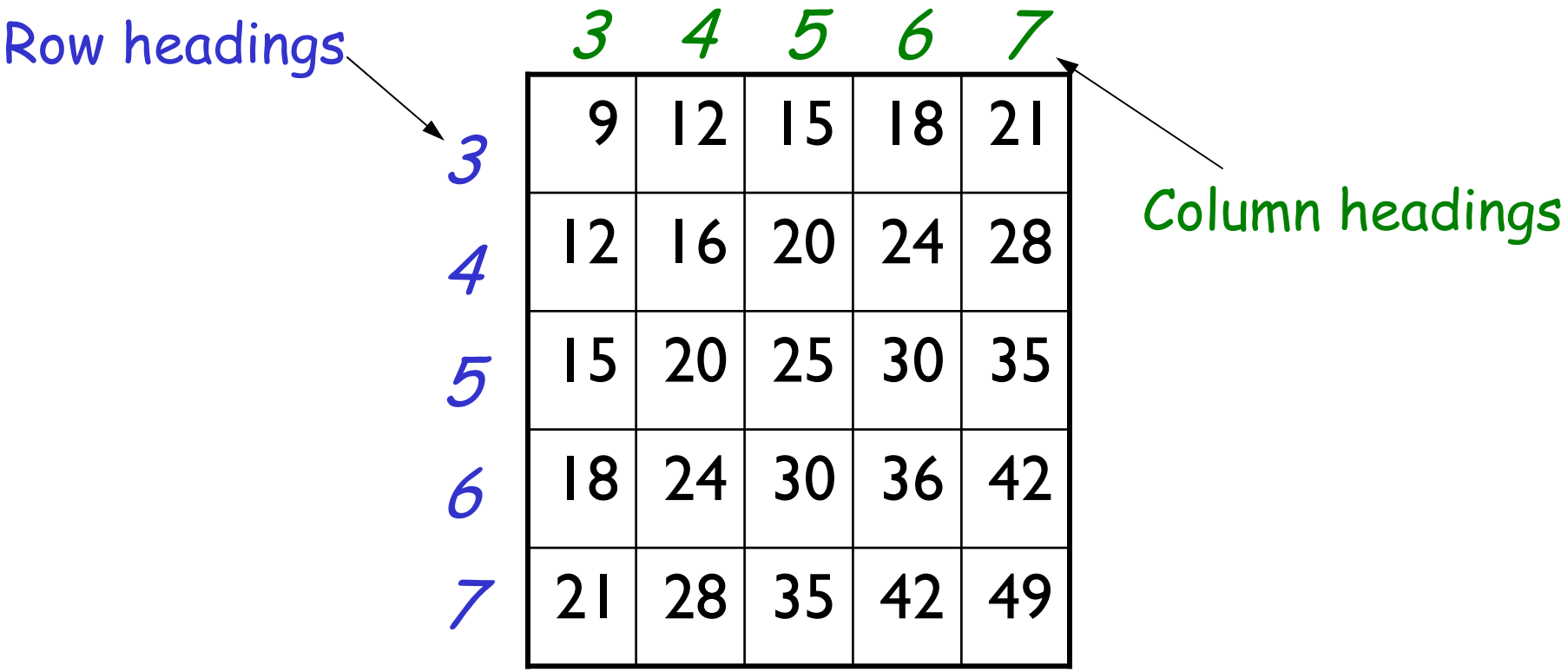
```
%Given n, display whether it is prime
divisor= 2;
while ( rem(n,divisor)~=0 )
     divisor= divisor + 1;
end
if (divisor==n)
     fprintf('%d is prime\n', n)
else
     fprintf('%d is composite\n', n)
end
```

```
for  n = a:b
    %Given n, display whether it is prime
    divisor= 2;
    while ( rem(n,divisor)~=0 )
        divisor= divisor + 1;
    end
    if (divisor==n)
        fprintf('%d is prime\n', n)
    else
        fprintf('%d is composite\n', n)
    end

end
```

Example: Times Table

Write a script to print a times table for a specified range.



Developing the algorithm for the times table

	3	4	5	6	7
3	9	12	15	18	21
4	12	16	20	24	28
5	15	20	25	30	35
6	18	24	30	36	42
7	21	28	35	42	49

Developing the algorithm for the times table

	3	4	5	6	7
3	9	12	15	18	21
4	12	16	20	24	28
5	15	20	25	30	35
6	18	24	30	36	42
7	21	28	35	42	49

- Look for patterns
  - Each entry is row# × col#
  - Row#, col# increase regularly
- ⇒ Loop!!!
- What kind of loop?
  - for-loop—since the range of the headings will be specified and increment regularly
  - for each row#, get the products with all the col#s. Then go to next row# and get products with all col#s, …
  - ⇒ Nested loops!
- Details: what will be the print format? Don't forget to start new lines. Also need initial input to specify the range.

```
disp('Show the times table for specified range')
lo= input('What is the lower bound? ');
hi= input('What is the upper bound? ');
```

Rational approximation of π

- π = 3.141592653589793…
- Can be closely approximated by fractions,

  e.g., $\pi \approx 22/7$
- 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

```
% Rational approximation of pi

M = input(‘Enter M: ’);



% Check all possible denominators
```

```
% Rational approximation of pi

M = input('Enter M: ');



% Check all possible denominators
for q = 1:M

    For current q find best numerator p…
    Check all possible numerators

end
```

```
% Rational approximation of pi

M = input('Enter M: ');



% Check all possible denominators
for q = 1:M
  % Find best numerator for this q

  for p = 1:M  % Check all possible p



  end



end
```

```
% Rational approximation of pi

M = input('Enter M: ');
% Best q, p, and error so far
qBest=1;  pBest=1;
err_pq = abs(pBest/qBest - pi);
% Check all possible denominators
for q = 1:M
  % Find best numerator for this q

  for p = 1:M  % Check all possible p



  end



end
myPi = pBest/qBest;
```

```
% Rational approximation of pi

M = input('Enter M: ');
% Best q, p, and error so far
qBest=1;  pBest=1;
err_pq = abs(pBest/qBest - pi);
% Check all possible denominators
for q = 1:M
  % Find best numerator for this q

  for p = 1:M  % Check all possible p



  end



end
myPi = pBest/qBest;
```

```
% Rational approximation of pi

M = input('Enter M: ');
% Best q, p, and error so far
qBest=1;  pBest=1;
err_pq = abs(pBest/qBest - pi);
% Check all possible denominators
for q = 1:M
  % Find best numerator for this q
  p0=1;  e0=abs(p0/q - pi);  % best p & error so far
  for p = 1:M  % Check all possible p



  end



end
myPi = pBest/qBest;
```

```
% Rational approximation of pi

M = input('Enter M: ');
% Best q, p, and error so far
qBest=1;  pBest=1;
err_pq = abs(pBest/qBest - pi);
% Check all possible denominators
for q = 1:M
  % Find best numerator for this q
  p0=1;  e0=abs(p0/q - pi);  % best p & error so far
  for p = 1:M  % Check all possible p
    if abs(p/q - pi) < e0  % new best numerator found
      p0=p;  e0 = abs(p0/q - pi);
    end
  end
```

Now we have the best p for this q.
Is the quotient at this q best among all previous q's?

```
end
myPi = pBest/qBest;
```

```
% Rational approximation of pi

M = input('Enter M: ');
% Best q, p, and error so far
qBest=1;  pBest=1;
err_pq = abs(pBest/qBest - pi);
% Check all possible denominators
for q = 1:M
  % Find best numerator for this q
  p0=1;  e0=abs(p0/q - pi);  % best p & error so far
  for p = 1:M  % Check all possible p
    if abs(p/q - pi) < e0  % new best numerator found
      p0=p;  e0 = abs(p0/q - pi);
    end
  end
```

Now we have the best p for this q.
Is the quotient at this q best among all previous q's?

```
end
myPi = pBest/qBest;
```

```
% Rational approximation of pi

M = input('Enter M: ');
% Best q, p, and error so far
qBest=1;  pBest=1;
err_pq = abs(pBest/qBest - pi);
% Check all possible denominators
for q = 1:M
  % Find best numerator for this q
  p0=1;  e0=abs(p0/q - pi);  % best p & error so far
  for p = 1:M  % Check all possible p
    if abs(p/q - pi) < e0  % new best numerator found
      p0=p;  e0 = abs(p0/q - pi);
    end
  end
  % Is best quotient for this q is best over all?
  if e0 < err_pq
    pBest=p0;  qBest=q; err_pq=e0;
  end
end
myPi = pBest/qBest;
```

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

The savvy programmer…

- Learns useful programming patterns and use them where appropriate
- Seeks inspiration by working through test data "by hand"
  - Asks, "What am I doing?" at each step
  - Sets up a variable for each piece of information maintained when working the problem by hand
- Decomposes the problem into manageable subtasks
  - Refines the solution iteratively, solving simpler subproblems first
- Remembers to check the problem's boundary conditions
- Validates the solution (program) by trying it on test data