- Previous Lecture:
  - Iteration using `for`
- Today's Lecture:
  - Detail on `for`-loop
  - Iteration using `while`
  - Review loops, conditionals using graphics
- Announcements:
  - Project 2 posted, due Thursday, 2/17
  - We do not use `break` in this course

`for` loop examples

```
for k= 2:0.5:3
     disp(k)
end
```

k takes on the values _____

Non-integer increment is OK

```
for k= 1:4
     disp(k)
end
```

k takes on the values _____

Default increment is 1

```
for k= 0:-2:-6
     disp(k)
end
```

k takes on the values _____

"Increment" may be negative

```
for k= 0:-2:-7
     disp(k)
end
```

k takes on the values _____

Colon expression specifies a *bound*

```
for k= 5:2:1
     disp(k)
end
```

`for` loop examples

```
for k= 2:0.5:3
     disp(k)
end
```

k takes on the values 2,2.5,3

Non-integer increment is OK

```
for k= 1:4
     disp(k)
end
```

k takes on the values 1,2,3,4

Default increment is 1

```
for k= 0:-2:-6
     disp(k)
end
```

k takes on the values 0,-2,-4,-6

"Increment" may be negative

```
for k= 0:-2:-7
     disp(k)
end
```

k takes on the values 0,-2,-4,-6

Colon expression specifies a *bound*

```
for k= 5:2:1
     disp(k)
end
```

The set of values for k is the empty set: the loop body won't execute

```
% What will be printed?
for k= 1:2:6
    fprintf('%d ', k)
end
```

A: 1 2 3 4 5 6

B: 1 3 5 6

C: 1 3 5

D: *error (incorrect bounds)*

```
% What will be printed?
for k= 10:-1:14
    fprintf('%d ', k)
end
fprintf('!')
```

A: *error (incorrect bounds)*

B: 10 *(then error)*

C: 10 !

D: 14 !

E: !

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

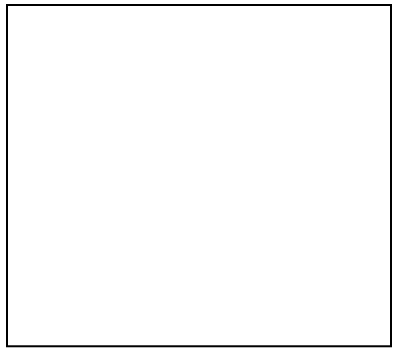
Something else …

C

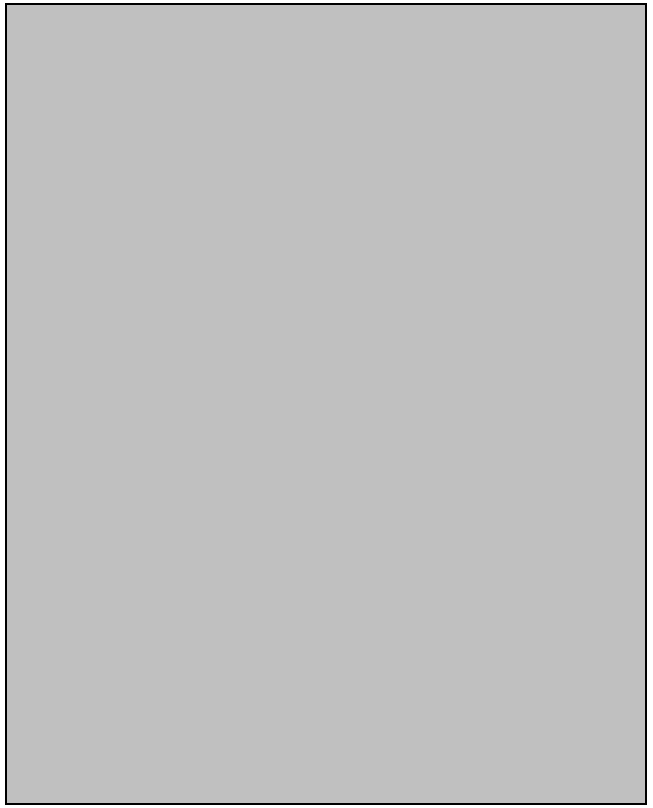
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```

With this loop header, **k** "promises" to be these values, one at a time

k



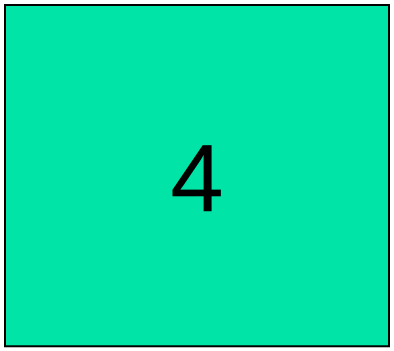
Output in Command Window



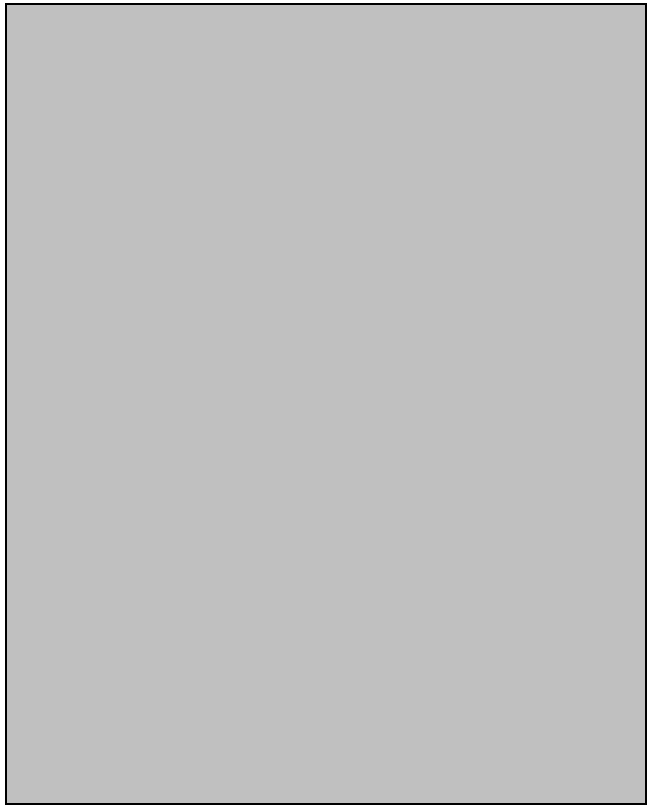
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```

With this loop header, **k** "promises" to be these values, one at a time

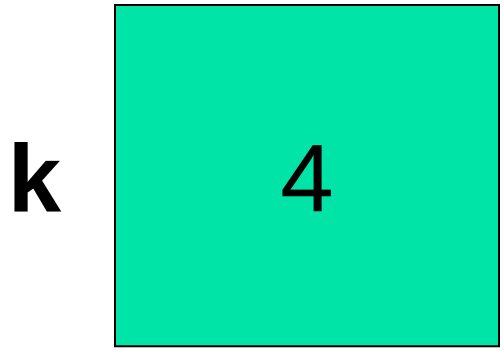
k



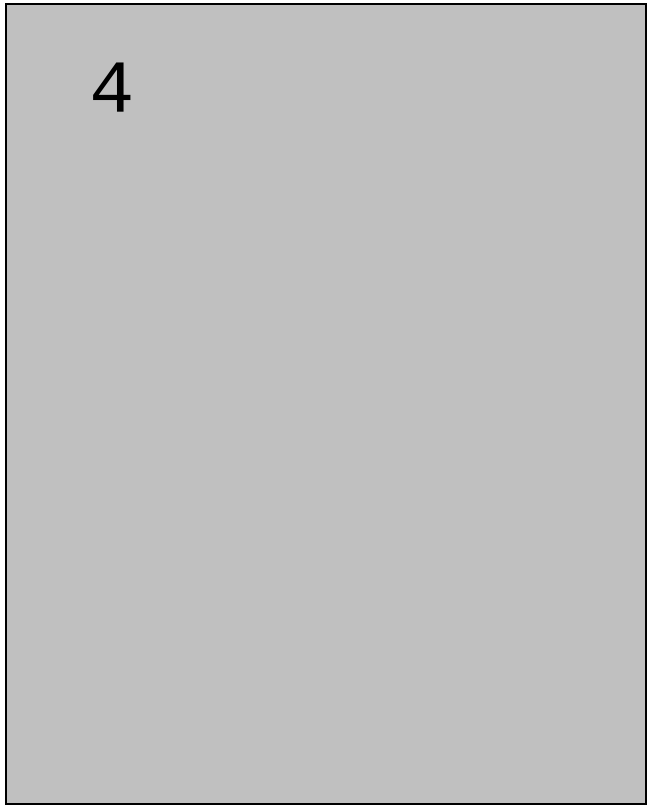
Output in Command Window



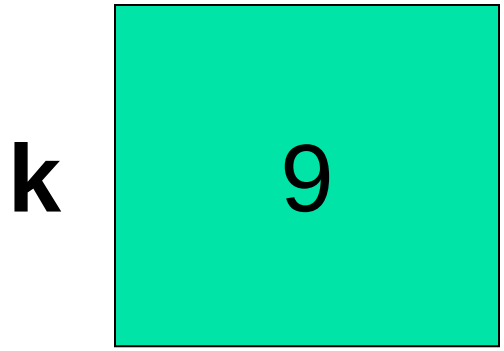
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```



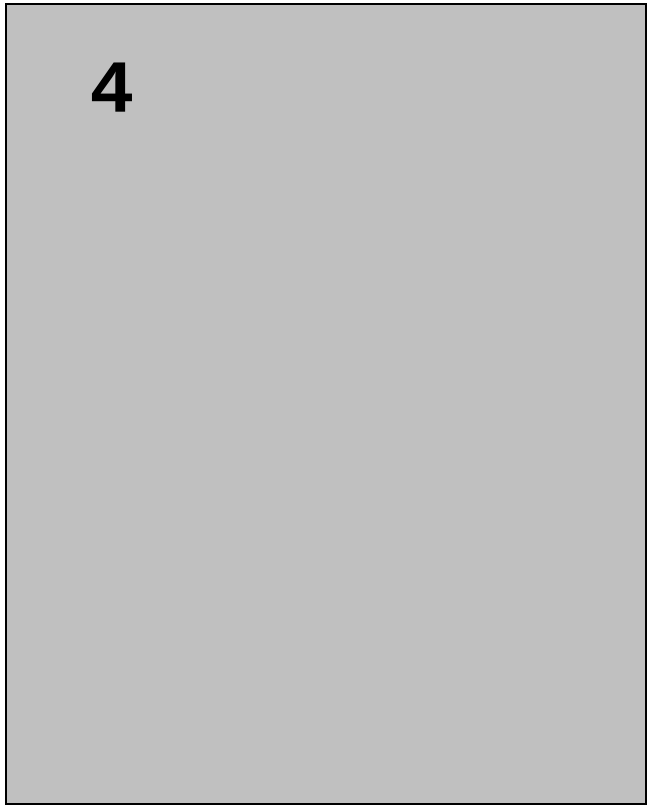
Output in Command Window



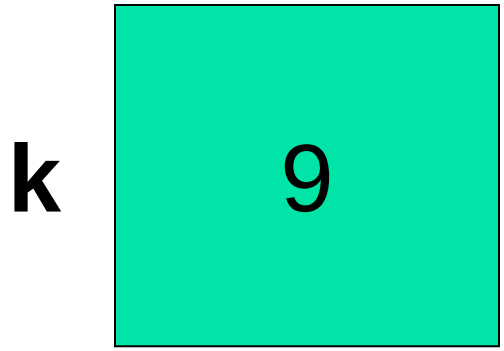
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```



Output in Command Window



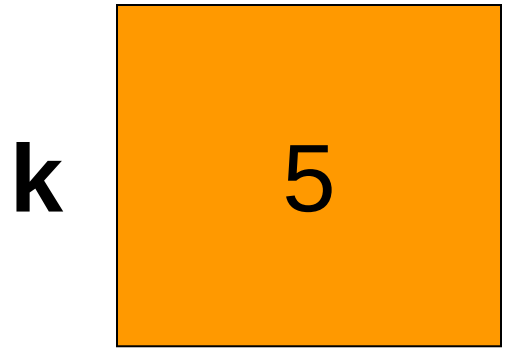
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```



Output in Command Window

```
4
9
```

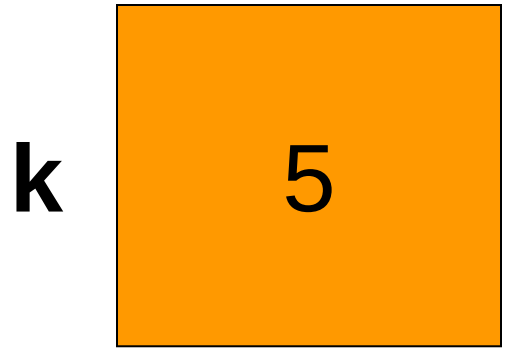
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```



Output in Command Window

```
4
9
```

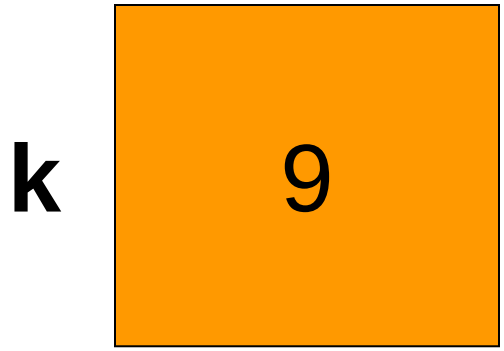
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```



Output in Command Window

```
4
9
5
```

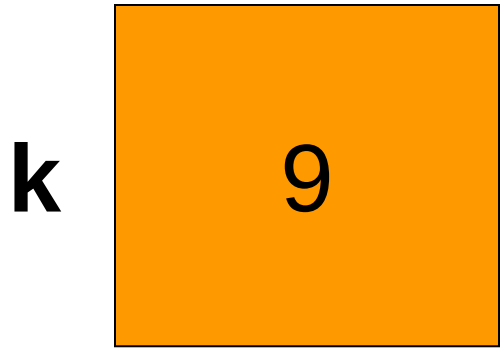
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```



Output in Command Window

```
4
9
5
```

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

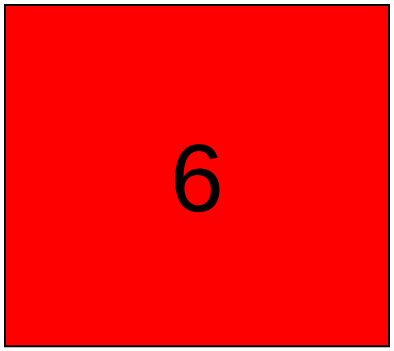


Output in Command Window

```
4
9
5
9
```

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

k

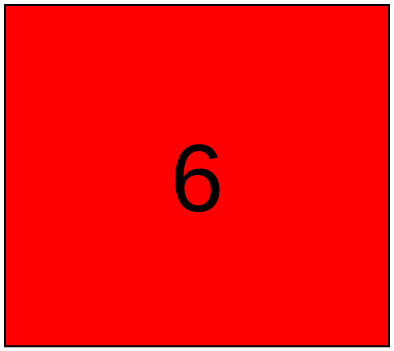


Output in Command Window

```
4
9
5
9
```

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

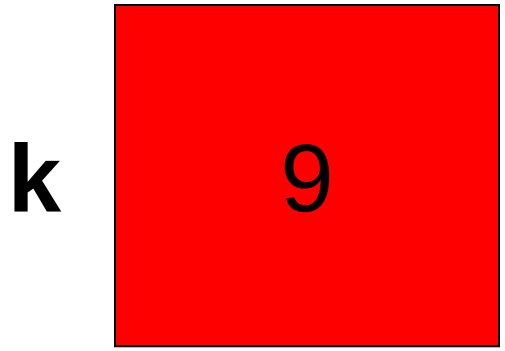
k



Output in Command Window

```
4
9
5
9
6
```

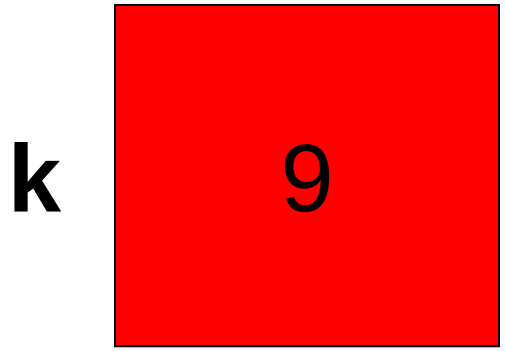
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```



Output in Command Window

```
4
9
5
9
6
```

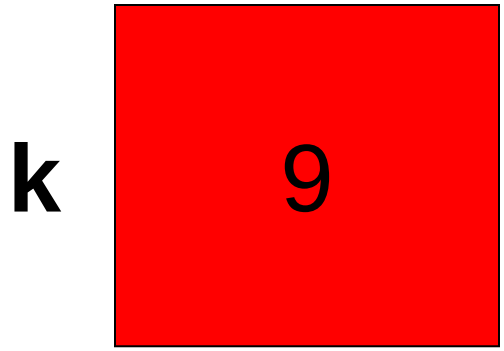
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```



Output in Command Window

```
4
9
5
9
6
9
```

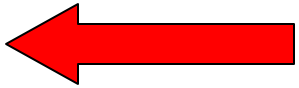
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```



Output in Command Window

```
4
9
5
9
6
9
```

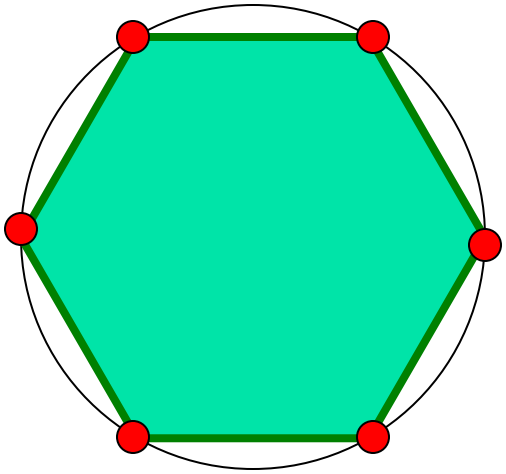
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```



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

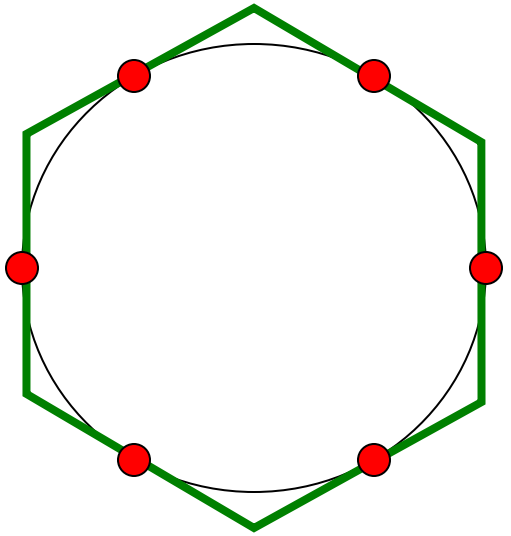
It is an expression that specifies values:

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.

When will |OuterA – InnerA| <= .000001?

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 difference is small:

increase n

calculate innerA, outerA for current n

diff= outerA – innerA

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

Now organize the tasks → algorithm:

n gets initial value
innerA, outerA get initial values
Repeat until difference is small:
    increase n
    calculate innerA, outerA for current n
    diff= outerA – innerA

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

n gets initial value
calculate innerA, outerA for current n
while *<difference is not small enough>*
    increase n
    calculate innerA, outerA for current n
    diff= outerA – innerA
end

Indefinite iteration

areaCircle.m

Guard against infinite loop

Use a loop guard that guarantees termination of the loop. Or just limit the number of iterations.

```
while (B_n-A_n >delta && n<nMax)
```

See **Eg2_2.m**

Another use of the while-loop: user interaction

- Example: Allow a user to repeatedly calculate the inscribed and circumscribed areas of n-gons on a unit circle.
- Need to define a "stopping signal"

`areaIndef.m`

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

Common loop patterns

Do something *n* times

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

end
```

Do something an indefinite number of times

```
%Initialize loop variables

while ( not stopping signal )
    % Do something

    % Update loop variables

end
```

In Matlab, which claim is true? (without **`break`**)

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

Common loop patterns

Do something *n* times

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

end
```

Do something an indefinite number of times

```
%Initialize loop variables

while ( not stopping signal )
    % Do something

    % Update loop variables

end
```

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
```

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

- for-loop: loop body repeats a *fixed* (predetermined) number of times.

- while-loop: loop body repeats an *indefinite* number of times under the control of the "loop guard."