- Previous Lecture:
  - Iteration using `for`

- Today’s Lecture:
  - Iteration using `while`
  - Review loop & conditionals using graphics

- Announcements:
  - Project 2 Part A posted, due Thursday, 2/14. Part B will be posted later and is also due on Thursday.
  - We do not use `break` in this course
  - Register your iClicker this week! See registration link on course website.
Syntax of the **for** loop

```plaintext
for <var> = <start value>:<incr>:<end bound>

    statements to be executed repeatedly

end
```

Loop header specifies all the values that the index variable will take on, one for each pass of the loop.

E.g., \( k = 3:1:7 \) means \( k \) will take on the values 3, 4, 5, 6, 7, one at a time.
Pattern for doing something $n$ times

\begin{verbatim}
n = _____
for k = 1:n
    \% code to do
    \% that something
end
\end{verbatim}

Definite iteration
% 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?

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

4
9
or
4
4
or
Something else …

A
B
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
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.

Output in Command Window:

\[
\begin{array}{c}
4 \\
5 \\
6 \\
\end{array}
\]
for k = 4:6
    disp(k)
    k = 9;
    disp(k)
end

Output in Command Window

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

Output in Command Window

\( k = 9 \)
for $k = 4:6$

disp(k)

$k = 9$

disp(k)

end

Output in Command Window

$\begin{array}{c}
4 \\
9
\end{array}$
for k = 4:6
    disp(k)
k = 9;
disp(k)
end

Output in Command Window

4 5 6

4
9

k 5
for k = 4:6
    disp(k)
    k = 9;
    disp(k)
end

Output in Command Window

4 5 6

k

5

4

9

5
for k = 4:6
disp(k)
k= 9;
disp(k)
end

Output in Command Window

k  9

4  5  6

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

Output in Command Window

k

4 5 6

4
9
5
9
6
for k = 4:6
    disp(k)
k= 9;
disp(k)  
end

Output in Command Window

4 5 6

k

9

4
5
9
6
9
for k = 4:6
    disp(k)
k = 9;
disp(k)
end

Output in Command Window

4 9
5
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:

4 5 6

for-loop header is executed only once!
(Loop body is may be executed multiple times)
Example: \( n \)-gon \( \rightarrow \) circle

\[ \text{Inscribed hexagon} \quad (n/2) \sin(2\pi/n) \]

\[ \text{Circumscribed hexagon} \quad n \tan(\pi/n) \]

As \( n \) approaches infinity, the inscribed and circumscribed areas approach the area of a circle.

When will \( \text{|OuterA} - \text{InnerA}| \leq 0.00001 \)?
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 $\rightarrow$ 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 \( \text{outerA} \) and \( \text{innerA} \) converge

Now organize the tasks \( \Rightarrow \) algorithm:

\( n \) gets initial value
\( \text{innerA}, \text{outerA} \) get initial values

Repeat until difference is small:
  \( \text{increase } n \)
  \( \text{calculate } \text{innerA}, \text{outerA} \text{ for current } n \)
  \( \text{diff} = \text{outerA} - \text{innerA} \)
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.

\[
\text{while } (B_n - A_n > \text{delta } \&\& \ n < n\text{Max})
\]

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

\begin{verbatim}
for k = 1:1:n
    % Do something
end
\end{verbatim}

Do something an indefinite number of times

\begin{verbatim}
% Initialize loop variables
while ( not stopping signal )
    % Do something
    % Update loop variables
end
\end{verbatim}
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

\[
\text{for } k = 1:1:n \\
\quad \text{% Do something} \\
\text{end}
\]

Do something an indefinite number of times

\[
\text{while ( not stopping signal )} \\
\quad \text{% Do something} \\
\quad \text{% Update loop variables} \\
\text{end}
\]
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
Pattern to do something \( n \) times

\[
\text{for } k = 1:1:n \\
\quad \text{\% Do something} \\
\text{end}
\]

\[
\text{\% Initialize loop variables} \\
k = 1; \\
\text{\% Do something} \\
\text{\% Update loop variables} \\
k = k + 1; \\
\text{end}
\]

\[
\text{\textbf{while} ( k <= n )} \\
\quad \text{\% Do something} \\
\text{\textbf{end}}
\]
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.”
Review loops/conditionals using user-defined graphics function

Draw a black square;
then draw a magenta disk;
then draw a yellow star.
DrawRect(-1,-2,6,3,'y')

x and y coordinates of lower left corner
width
height
color
DrawDisk(1, 3, 4, 'r')

- x and y coordinates of the center
- radius
- color
DrawStar(1,3,4,'g')

- x and y coordinates of the center
- "radius"
- color
## Color Options

<table>
<thead>
<tr>
<th>Color</th>
<th>Code</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>White</td>
<td>‘w’</td>
<td><img src="image" alt="White" /></td>
</tr>
<tr>
<td>Black</td>
<td>‘k’</td>
<td><img src="image" alt="Black" /></td>
</tr>
<tr>
<td>Red</td>
<td>‘r’</td>
<td><img src="image" alt="Red" /></td>
</tr>
<tr>
<td>Blue</td>
<td>‘b’</td>
<td><img src="image" alt="Blue" /></td>
</tr>
<tr>
<td>Green</td>
<td>‘g’</td>
<td><img src="image" alt="Green" /></td>
</tr>
<tr>
<td>Yellow</td>
<td>‘y’</td>
<td><img src="image" alt="Yellow" /></td>
</tr>
<tr>
<td>Magenta</td>
<td>‘m’</td>
<td><img src="image" alt="Magenta" /></td>
</tr>
<tr>
<td>Cyan</td>
<td>‘c’</td>
<td><img src="image" alt="Cyan" /></td>
</tr>
</tbody>
</table>
A simple 3-line script

\texttt{DrawRect(...)}
\texttt{DrawDisk(...)}
\texttt{DrawStar(...)}
% 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

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

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

hold off
```
Example: Nested Stars