Lecture 7: Nested loops

▪ Previous lecture:
  ◦ (Indefinite) iteration using while

▪ Today:
  ◦ Nested loops
  ◦ Developing algorithms

▪ Announcements:
  ◦ Project 1 grades released tomorrow
  ◦ Project 2 due Mon, Mar 8; part B released after lecture

“I had to learn how to study differently – by practicing every day rather than cramming before. I wish that we could have been told earlier in the year to practice like 30 minutes per day...” – FA19 student
tol = input('Enter the error tolerance: ');

n = 3;  % Number of Polygon Edges
A_n = (n/2)*sin(2*pi/n);  % Inscribed Area
B_n = n*tan(pi/n);  % Circumscribed Area
ErrorBound = B_n - A_n;  % The error bound

while (ErrorBound > tol)
    n = n+1;  A_n = (n/2)*sin(2*pi/n);  B_n = n*tan(pi/n);
    ErrorBound = B_n - A_n;
end

% Display the final approximation
tol = input('Enter the error tolerance: ');

tolMin = 1e-12;
while tol < tolMin
    tol = input(sprintf('Enter a tolerance >= %.0e: ', tolMin));
end

n = 3;           % Number of Polygon Edges
A_n = (n/2)*sin(2*pi/n);  % Inscribed Area
B_n = n*tan(pi/n);       % Circumscribed Area
ErrorBound = B_n - A_n;  % The error bound

while (ErrorBound > tol)
    n = n+1;    A_n = (n/2)*sin(2*pi/n);    B_n = n*tan(pi/n);
    ErrorBound = B_n - A_n;
end

% Display the final approximation
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.”
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

\[
\% \text{Initialize loop variables} \\
\text{while} (\text{not stopping signal}) \\
\quad \% \text{ Do something} \\
\quad \% \text{ Update loop variables} \\
\text{end}
\]
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

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

\[
\text{while } \left( \text{not stopping signal} \right) \\
\text{\hspace{1em}} \% \text{ Do something} \\
\text{\hspace{1em}} \% \text{ Update loop variables} \\
\text{end}
\]
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 VS. while

\[ \text{N} = \_\_\_; \ \text{L} = \_\_\_; \ \text{hits} = 0; \]

\begin{verbatim}
for k = 1:1:N  
    % Throw kth dart  
    x = rand() * L - L/2;  
    y = rand() * L - L/2;  
    % Count if in circle  
    if sqrt(x^2 + y^2) <= L/2  
        hits = hits + 1;  
    end
end

\text{myPi} = 4*\text{hits}/\text{N};
\end{verbatim}

\[ \text{N} = \_\_\_; \ \text{L} = \_\_\_; \ \text{hits} = 0; \]

\begin{verbatim}
k = 1;  
while k <= N  
    % Throw kth dart  
    x = rand() * L - L/2;  
    y = rand() * L - L/2;  
    % Count if in circle  
    if sqrt(x^2 + y^2) <= L/2  
        hits = hits + 1;  
    end  
    k = k + 1;
end

\text{myPi} = 4*\text{hits}/\text{N};
\end{verbatim}
Can we cheat?

\[ N = 10; \]
\[ \text{for } k = 1:N \]
\[ \% \text{ Do some work.} \]
\[ \text{if (stopping condition)} \]
\[ N = k; \]
\[ \text{end} \]
\[ \text{end} \]

Save off range
Pick next k from saved range
N changes in workspace; nothing else happens
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
Review loops/conditionals using user-defined graphics function

Draw a black square;
then draw a magenta disk;
then draw a yellow star.
Refinement tip: Survey your tools
DrawRect(-1,-2,6,3,'y')
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
<table>
<thead>
<tr>
<th>Color</th>
<th>Code</th>
<th>Color Bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>White</td>
<td>'w'</td>
<td><img src="image" alt="White Color" /></td>
</tr>
<tr>
<td>Black</td>
<td>'k'</td>
<td><img src="image" alt="Black Color" /></td>
</tr>
<tr>
<td>Red</td>
<td>'r'</td>
<td><img src="image" alt="Red Color" /></td>
</tr>
<tr>
<td>Blue</td>
<td>'b'</td>
<td><img src="image" alt="Blue Color" /></td>
</tr>
<tr>
<td>Green</td>
<td>'g'</td>
<td><img src="image" alt="Green Color" /></td>
</tr>
<tr>
<td>Yellow</td>
<td>'y'</td>
<td><img src="image" alt="Yellow Color" /></td>
</tr>
<tr>
<td>Magenta</td>
<td>'m'</td>
<td><img src="image" alt="Magenta Color" /></td>
</tr>
<tr>
<td>Cyan</td>
<td>'c'</td>
<td><img src="image" alt="Cyan Color" /></td>
</tr>
</tbody>
</table>
Draw a black square; then draw a magenta disk; then draw a yellow star.

DrawRect( , , , , )
DrawDisk( , , , )
DrawStar( , , , )
Draw a black square; then draw a magenta disk; then draw a yellow star.

DrawRect(0,0,2,2,'k')
DrawDisk(1,1,1,'m')
DrawStar(1,1,1,'y')
% 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
Example: Nested Stars

Outline
I. Draw box
II. Repeat until too small
   a) Draw positive Star of current color
   b) Decrease size
   c) Change color
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 square
- When to stop?
  - when r is small
x = 0; y = 0;  % figure centered at (0,0)
r = 1;       % radius of outermost star
s = 2*r + 0.1;  % side length of square
DrawRect(x-s/2, y-s/2, s, s, 'k')

% Draw nested stars, smallest r at least 0.1
x = 0; y = 0; % figure centered at (0,0)
r = 1; % radius of outermost star
s = 2*r + 0.1; % side length of square
DrawRect(x-s/2, y-s/2, s, s, 'k')

% Draw nested stars, smallest r at least 0.1

while r >= 0.1
    % Draw a star with radius r

    % Reduce r

end
x = 0; y = 0;  % figure centered at (0,0)
r = 1;  % radius of outermost star
s = 2*r + 0.1;  % side length of square
DrawRect(x-s/2, y-s/2, s, s, 'k')

% Draw nested stars, smallest r at least 0.1

while r >= 0.1
    % Draw a star with radius r

    % Reduce r
    r = r/1.2;
end
x = 0; y = 0;  \% figure centered at (0,0)
r = 1; \% radius of outermost star
s = 2*r + 0.1; \% side length of square

DrawRect(x-s/2, y-s/2, s, s, 'k')

\% Draw nested stars, smallest r at least 0.1

\textbf{while} \ r >= 0.1
\textbf{do}
\hspace{1cm} \% Draw a star with radius r
\hspace{1cm} \textbf{if}
\hspace{2cm} \% magenta
\hspace{3cm} \% magenta
\hspace{1cm} \textbf{else}
\hspace{2cm} \% yellow
\hspace{3cm} \% yellow
\hspace{1cm} \textbf{end}
\hspace{1cm} \% Reduce r
\hspace{1cm} r = r/1.2;
\hspace{1cm} \textbf{end}
x = 0; y = 0; % figure centered at (0,0)
r = 1; % radius of outermost star
s = 2*r + 0.1; % side length of square
DrawRect(x-s/2, y-s/2, s, s, 'k')

% Draw nested stars, smallest r at least 0.1

while r >= 0.1
    % Draw a star with radius r
    if rem(k,2) == 1 % odd k
        DrawStar(x, y, r, 'm') % magenta
    else
        DrawStar(x, y, r, 'y') % yellow
    end
    % Reduce r
    r = r/1.2;
k = k + 1;
end
Knowing how to draw, ... how difficult is it to draw... ?
Pattern for doing something $n$ times

\[ n = \_\_\_ \]

\[ \text{for } k = 1 : n \]

\[ \text{% code to do} \]
\[ \text{% that something} \]

\[ \text{end} \]
x = 0; y = 0;    % figure centered at (0,0)
r = 1;          % radius of outermost star
s = 2*r + 0.1;  % side length of square
DrawRect(x-s/2, y-s/2, s, s, 'k')

% Draw nested stars, smallest r at least 0.1

while r >= 0.1
    % Draw a star with radius r
    if rem(k,2) == 1  % odd k
        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= 0; y= 0;  % figure centered at (0,0)
rem(k,2) == 1 % odd k
r= 1;  % radius of outermost star
r= r/1.2;
end

% radius of outermost star
s= 2*r + 0.1;  % side length of square
k= k + 1;
end

% side length of square
DrawRect(x-s/2, y-s/2, s, s, 'k')

% figure centered at (0,0)
while r >= 0.1
% Draw a star with radius r
end

if rem(k,2) == 1 % odd k
% Draw nested stars, smallest r at least 0.1
DrawStar(x, y, r, 'm') % magenta
else
% Reduce r
DrawStar(x, y, r, 'y') % yellow
end

% Reduce r
r= r/1.2;
k= k + 1;
end

end
Example: Times Table

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

<table>
<thead>
<tr>
<th>Row headings</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column headings</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>21</td>
</tr>
</tbody>
</table>
Developing the algorithm for the times table

- 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 is specified and the increment is regular
  - 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? ');