Lecture 8: User-defined functions

- Previous lecture:
  - Nested loops
  - Developing algorithms and code

- Today:
  - Review nested loops
  - User-defined functions, part I

- Announcements:
  - Regrade requests
  - Project 2 due Mon, Mar 8
  - Office hours after lecture
  - Next Tue/Wed: Wellness Days (no lecture or lab)
Example: Times Table

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

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
</tr>
</tbody>
</table>
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 $\pi$

- $\pi = 3.141592653589793...$
- Can be closely approximated by fractions, e.g., $\pi \approx 22/7$
- Rational number: a quotient of two integers
- Approximate $\pi$ as $p/q$ where $p$ and $q$ are positive integers $\leq M$
- Start with a straightforward solution:
  - Get $M$ from user
  - Calculate quotient $p/q$ for all combinations of $p$ and $q$
  - Pick best quotient $\rightarrow$ 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
    % At this q, check all possible numerators
    for p = 1:M

        end
    end
end
Pattern: Best in set

Algorithm: Finding the best in a set

Init bestSoFar (value & quality)
Loop over set
  if current is better than bestSoFar
    bestSoFar ← current
  end
end
bestSoFar is best overall
% 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
    % At this q, check all possible numerators
    for p = 1:M
        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
    % At this q, check all possible numerators
    for p = 1:M
        if abs(p/q - pi) < err_pq  % better p/q found
            err_pq = abs(p/q - pi);
pBest= p;
qBest= q;
        end
    end
end

myPi = pBest/qBest;
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

    % At this q, check all possible numerators
    p0 = 1; e0 = abs(p0/q - pi); % best p & error so far
    for p = 1:M
        if abs(p/q - pi) < e0 % new best numerator found
            p0 = p; e0 = abs(p/q - pi);
        end
    end

    % Is best quotient for this q the best overall?
    if e0 < err_pq
        pBest = p0; qBest = q; err_pq = e0;
    end
end

myPi = pBest/qBest;
Analyzing cost

- See Eg3_1 and FasterEg3_1 in the book

```matlab
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
Stepping back... what do we know about scripts?

- Bundle complicated logic conveniently under one name
  - To find best rational approx. to π, just run piFrac

- Inputs and outputs interact with humans
  - input(), fprintf()

- Share variables in common workspace
  - Danger: inheriting bad initialization from previous computation

What if...

- Inputs and outputs interacted with other code?
  - Interaction with humans considered a “side effect”

- Behavior not affected by other computations?
**Built-in functions**

- We’ve used many Matlab built-in functions, e.g., `rand()`, `abs()`, `floor()`, `rem()`
- Example: `abs(x-0.5)`
- Observations:
  - `abs()` is set up to be able to work with any valid data
  - `abs()` *doesn’t prompt us for input; it expects that we provide data* that it’ll then work on
  - `abs()` *returns* a value that we can use in our program

```matlab
yDistance = abs(y2-y1);
while abs(myPi-pi) > .0001
    ...
```
User-defined functions

We can write our own functions to perform a specific task:

▪ **Example**: draw a disk with specified radius, color, and center coordinates
  ◦ Inputs: center, radius, color
  ◦ Outputs: none
  ◦ Side effects: Shows disk to user

▪ **Example**: generate a random floating-point number in a specified interval
  ◦ Inputs: interval lower bound, interval upper bound
  ◦ Outputs: random number

▪ **Example**: convert polar coordinates to x-y (Cartesian) coordinates
  ◦ Inputs: \( r \)-coordinate, \( \theta \)-coordinate
  ◦ Outputs: \( x \)-coordinate, \( y \)-coordinate

  *Multiple outputs are allowed!*
Functions step-by-step

1. Identify candidates
   ◦ Look for opportunities to reuse logic or improve clarity

2. Design interface
   ◦ Name, inputs, outputs, side effects

3. Implement function
   ◦ “Write code”

4. Test
   ◦ Try it out (and try to break it)

5. Use
Draw a bulls eye figure with randomly placed dots

- Dots are randomly placed within concentric rings
- User decides how many rings, how many dots per ring
Draw a bulls eye figure with randomly placed dots

- What are the main tasks?
- Accommodate variable number of rings—loop

- For each ring
  - Need many dots (another loop)
  - For each dot
    - Generate random position
    - Choose color
    - Draw it
Convert from polar to Cartesian coordinates

Polar coordinates

Cartesian coordinates

$r$, $\theta$, $x$, $y$
c = input('How many concentric rings? ');
d = input('How many dots per ring? ');

% Put dots btwn circles with radii rRing and (rRing-1)
for rRing= 1:c
    % Draw d dots
    for count= 1:d

        % Generate random dot location (polar coord.)
        theta= _______
        r= _______

        % Convert from polar to Cartesian
        x= _______
        y= _______

        % Use plot to draw dot
        end
    end
end

A common task! Create a function polar2xy to do this. polar2xy likely will be useful in other problems as well.

Outline
- For each ring
  - For each dot
    - Generate random position
    - Choose color
    - Draw dot
% Generate random dot location (polar)
theta= _____  % degrees
r= ________

% Convert from polar to Cartesian
rads= theta*pi/180;  % radian
x= r*cos(rads);
y= r*sin(rads);
function [x, y] = polar2xy(r,theta)
% Convert polar coordinates (r,theta) to
% Cartesian coordinates (x,y).
% theta is in degrees.

rads = theta*pi/180;  % radian
x = r*cos(rads);
y = r*sin(rads);
function [x, y] = polar2xy(r,theta)
% Convert polar coordinates (r,theta) to
% Cartesian coordinates (x,y).
% theta is in degrees.

rads = theta*pi/180; % radian
x = r*cos(rads);
y = r*sin(rads);

Think of polar2xy as a factory

Caller provides input to the “factory”

rads = theta*pi/180; % radian
x = r*cos(rads);
y = r*sin(rads);

“Factory” produces output for caller to use
function [x, y] = polar2xy(r,theta)
% Convert polar coordinates (r,theta) to
% Cartesian coordinates (x,y).
% theta is in degrees.

rads = theta*pi/180;  % radian
x = r*cos(rads);
y = r*sin(rads);
Functions step-by-step

1. **Identify candidates**
   - Look for opportunities to reuse logic or improve clarity

2. **Design interface**
   - Name, inputs, outputs, side effects

3. **Implement function**
   - “Write code”

4. **Test**
   - Try it out (and try to break it)

5. **Use**
Time for testing

- Good test cases:
  - \( r = 0 \)
  - \( \theta = 0, \pi/2, \pi, 3\pi/2 \)
  - \( \theta = -\pi, 3\pi \)

- How to test
  - \([x, y] = \text{polar2xy}(r, \theta)\)
  - Command window
  - Test script
  - DON'T try to “run” the function file
c = input('How many concentric rings? ');
d = input('How many dots per ring? ');

% Put dots btwn circles with radii rRing and (rRing-1)
for rRing= 1:c
    % Draw d dots
    for count= 1:d

        % Generate random dot location (polar coord.)
        theta= _______
r= _______

        % Convert from polar to Cartesian
        x= _______
y= _______

        % Use plot to draw dot
    end
end

A common task! Create a function polar2xy to do this. polar2xy likely will be useful in other problems as well.
c = input('How many concentric rings? ');
d = input('How many dots per ring? ');

% Put dots btw circles with radii rRing and (rRing-1)
for rRing= 1:c
    % Draw d dots
    for count= 1:d
        % Generate random dot location (polar coord.)
        theta = _______
        r = _______

        % Convert from polar to Cartesian
        x = _______
        y = _______

        % Use plot to draw dot
    end
end