ME 350 Lab Exercise 3 `for` loops, `fprintf`, `if` constructs
Fall 2017

Practice Reading `for` Loops

For each of the following code snippets, fill out the table to the right with the values displayed when the code snippet is executed. After you have evaluated the code manually (i.e., without using MATLAB), create an m-file with these code snippets to check your work.

1. Loop 1:

```
x = 10.0;
for i = 1:4
  x = x/2.0;
  disp([i x]);
end
disp('At end of the loop, x =')
disp(x)
```

i	x
1	
2	
3	
4	

At the end of the loop, `x` =

2. Loop 2:

```
x = 10.0;
for i = 1:4
  fprintf('%3d   %8.5f\n',i,x)
  x = x/2.0;
end
fprintf('At end of loop, x = %8.5f',x);
```

i	x
1	
2	
3	
4	

At the end of the loop, `x` =

3. Loop 3:

```
m = 3;
n = 3;
fprintf('\n  i   j   i+j\n')
for i = 1:m
  for j = 1:n
    fprintf('%3d %3d %4d\n',i,j,i+j);
  end
end
```

i	j	i+j
1		
1		
1		

4. Loop 4: Use the space to the right to show the output of running the `demoDouble` function with the default input parameters. Work out the pattern "by hand" before you enter the code in MATLAB or download it from the Lab web page.

```
function demoDouble(m,n)

if nargin<1, m=4; end
if nargin<2, n=5;  end

fprintf('\nBegin double loop:\n\n     ')
fprintf('%3d',1:n)
fprintf('\n    +')
for j=1:n
  fprintf('---');
end
fprintf('\n')
for i = 1:m
  fprintf('%3d |',i)
  for j = 1:n
    fprintf('%3d',i+j);
  end
  fprintf('\n')
end
end
```

A Practical Example

`for` loops are most often used when each element in a vector or matrix is to be processed.

Syntax

```
for index = expression
    block of statements
end
```

Example: Sum of elements in a vector

```
x = rand(1,50);         %  create a row vector of random values, as an example
sumx = 0;               %  initialize the sum
for k = 1:length(x)     %  length(x) returns the number of elements in x
   sumx = sumx + x(k);
end
```

1. Write an m-file called `mySum` that accepts an input vector, `x`, and returns the sum of the elements in `x`. This amounts to putting a function wrapper around the preceding code block.

2. Write an m-file called `testSum` that compares the output of your `mySum` function with the built-in `sum` function. Use random inputs with `x = rand(100,1)` as test vectors.

Focus on the `for` Statement

The following three examples are nearly identical. The only difference is in the "`for`" expression. The body of the loop has the same code to print values of `theta` in degrees and radians, and print the value of the sine of `theta`.

The goal of these three exercises is to get you to think about how the code in the `for` statement controls the execution. *The code in Loop 7 is the recommended way* to perform these calculations.

Finally, note that the "`...`" in the `fprintf` statement allows us to break the long line into two lines that better fit onto the page. The `...` is legitimate MATLAB syntax. If you type this code into MATLAB you don't *need* to use the `...` to break the `fprintf` statement in two separate lines.

1. Loop 5:

```
fprintf('   theta\n  (rad)   (deg)  sin(theta)\n')

for theta = [0 pi/4.0 pi/2.0 3*pi/4.0 pi]

    fprintf('  %6.4f %5.0f     %6.4f\n',...
      theta,theta*180.0/pi,sin(theta));
end
```

2. Loop 6:

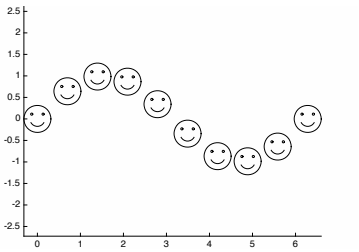
```
fprintf('   theta\n  (rad)   (deg)  sin(theta)\n')

theta = 0:pi/4.0:pi;

for i = length(theta)

    fprintf('  %6.4f %5.0f     %6.4f\n',...
       theta(i),theta(i)*180.0/pi,sin(theta(i)));
end
```

3. Loop 7:

```
fprintf('   theta\n  (rad)   (deg)  sin(theta)\n')

theta = 0:pi/4.0:pi;

for i = 1:length(theta)

    fprintf('  %6.4f %5.0f     %6.4f\n',...
       theta(i),theta(i)*180.0/pi,sin(theta(i)));
end
```

Putting `for` Loops to Work

1. Download the `drawSmile.m` m-file from the web page for the lab exercise. *Without altering the code in* `drawSmile.m`, write another m-file, say `sineSmile.m` that draws smiley faces centered at 10 points along a sine curve in the range $[0, 2\pi]$. The result should look like the plot to the right.



2. The infinite series for $\sin(x)$ is

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + \frac{(-1)^{k-1}x^{2k-1}}{(2k-1)!} + \ldots \qquad (1)$$

Download the `nTermSine1` function from the lab web page. `nTermSine1` is a straightforward implementation of this series in Equation (1). Get acquainted with the `nTermSine1` function by running these commands

(a) `s = nTermSine1(pi/4)`

(b) `s = nTermSine1(pi/4) - sin(pi/4)`

(c) `s = nTermSine1(pi/4,3) - sin(pi/4)`

(d) `s = nTermSine1(pi/4,10,true)`

3. Create a new m-file called `testSineSeries` to perform the following tasks.

(a) Generate a vector of x values in the range $0 \leq x \leq \pi$.

(b) Run `nTermSine1` to create a vector of 5-term approximations to $\sin(x)$ over the range of x values.
Hint

```
theta = linspace( ... );          %  Range of angles
sinApprox = zeros(size(theta));   %  Pre-allocate for efficiency

for i = 1:length(theta)

   sinApprox(i) = ...

end

err = ...              %  absolute error
plot(theta,err,'.')
```

DO NOT change any code in `nTermSine1.m`, and **DO NOT** copy any code from `nTermSine1.m` into your `testSineSeries.m` file.

(c) Store the approximate values from the sine series in `sinApprox` and compare these with the value returned by the built-in `sin` function. In the preceding code snippet, `err` is the absolute error.

(d) What happens to the error if the range of θ values used in the test is extended to $0 \leq \theta \leq 2\pi$?

Note that the loop is *required* because unlike the built-in `sin` function that can accept a vector of values of input, the first input value to `nTermSin1` *must be* a scalar value.

```
function s = nTermSine1(x,n,verbose)
% nTermSine1  Evaluate the n-term series approximation to sin(x)
%             Simplest approach: evaluate each term from scratch
%
% Synopsis:  s = nTermSine(x)
%            s = nTermSine(x,n)
%            s = nTermSine(x,n,verbose)
%
% Input:  x = argument of sine, i.e. compute sin(x)
%         n = (optional) number of terms to evaluate.  Default: n = 5
%         verbose = (optional) flag to control printing of intermediate
%                   results.  Default: verbose = false, no printing
%
% Output:  s = n-term approximation to sin(x)

if nargin<2, n=5;            end
if nargin<3, verbose=false;  end

% -- Initialize the series and print deader if desired
term = x;
s = term;
sgn = 1;
if verbose
  fprintf('\n    i   sign    k        term             s\n');
  fprintf(' %4d  %4d  %4d  %18.13f  %8.5f\n',1,sgn,1,term,s);
end

% -- Loop for terms 2 to n
for i=2:n
  sgn = -sgn;          %  switch sign of term
  k = 2*i - 1;
  term = sgn*(x^k)/factorial(k);
  s = s + term;
  if verbose, fprintf(' %4d  %4d  %4d  %18.13f  %8.5f\n',i,sgn,k,term,s); end
end
```

Listing 1: The `nTermSine1` function is a straightforward implementation of the the series approximation to $\sin(x)$.