SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB
Unit 6-Lecture 23
"for ... end" structure

BTech (CSBS) -Semester VII

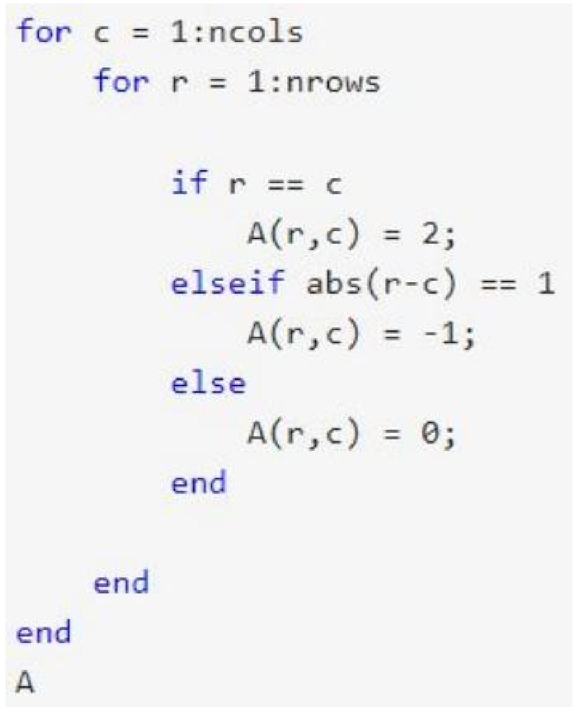
7 October 2022, 09:35AM

Control Flow and Operators

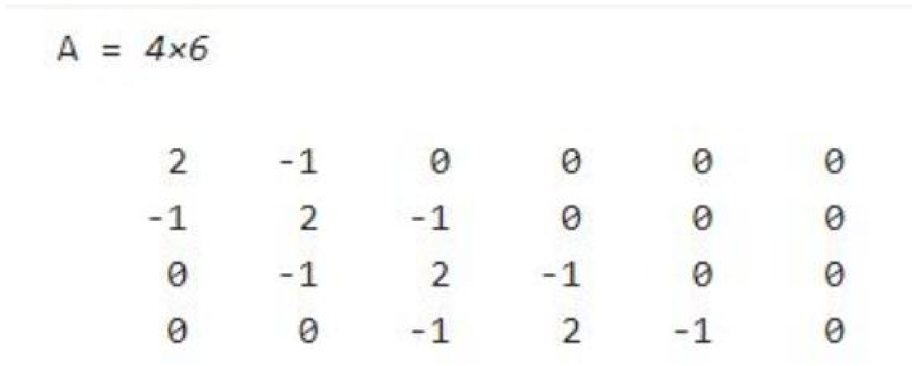
1) relational and logical operators
2) “if ... end" structure
3) “for ... end" loop
4) “while ... end" loop
5) other flow structures
6) operator precedence
7) saving output to a file

Question

Loop through the matrix and assign each element a new value. Assign 2 on the main diagonal, -1 on the adjescent diagonals and 0 everywhere else.
Given: nrows=4; nclos=6; A=ones(nrows,nclos);

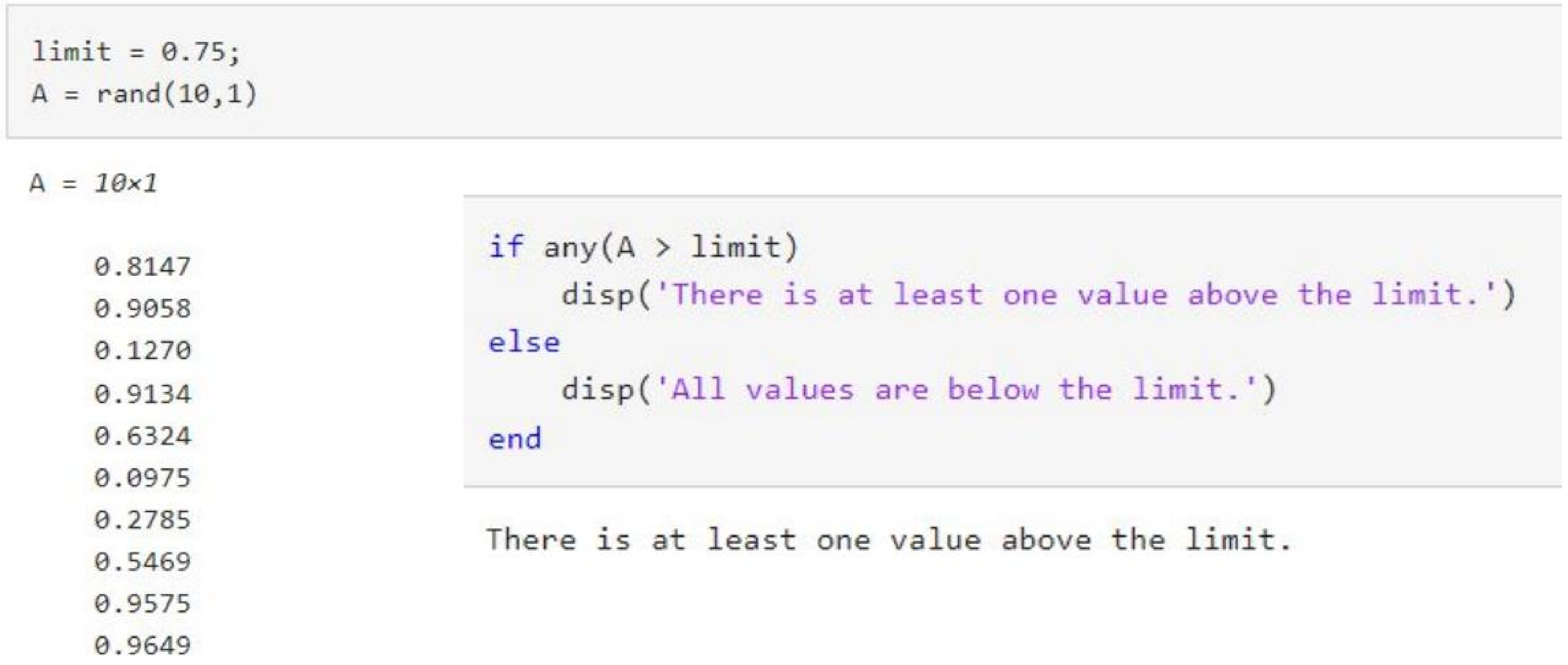


```
for c = 1:ncols
    for r = 1:nrows

        if r == c
            A(r,c) = 2;
        elseif abs(r-c) == 1
            A(r,c) = -1;
        else
            A(r,c) = 0;
        end

    end
end
A
```



```
A = 4×6

     2    -1     0     0     0     0
    -1     2    -1     0     0     0
     0    -1     2    -1     0     0
     0     0    -1     2    -1     0
```

example using “any”



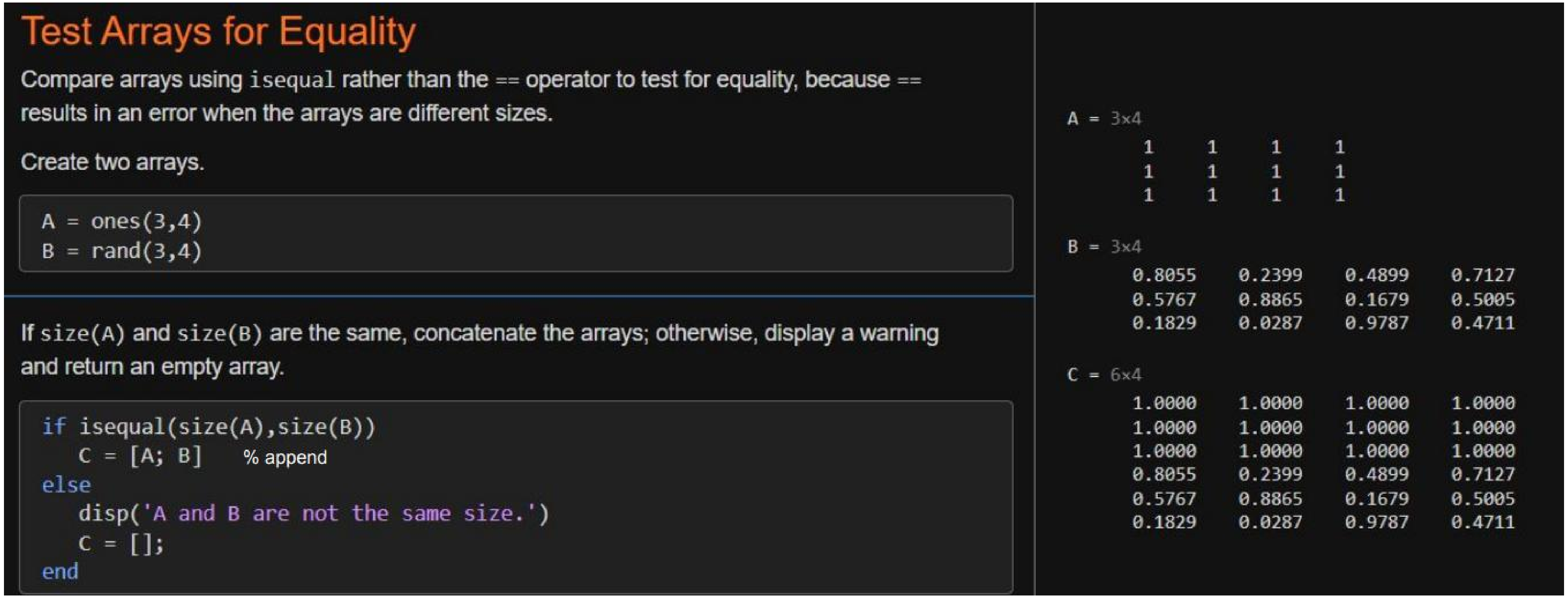
```
limit = 0.75;
A = rand(10,1)
```

```
A = 10×1

    0.8147
    0.9058
    0.1270
    0.9134
    0.6324
    0.0975
    0.2785
    0.5469
    0.9575
    0.9649
```

```
if any(A > limit)
    disp('There is at least one value above the limit.')
else
    disp('All values are below the limit.')
end
```

```
There is at least one value above the limit.
```

example using “isequal”



Test Arrays for Equality

Compare arrays using `isequal` rather than the == operator to test for equality, because == results in an error when the arrays are different sizes.

Create two arrays.

```
A = ones(3,4)
B = rand(3,4)
```

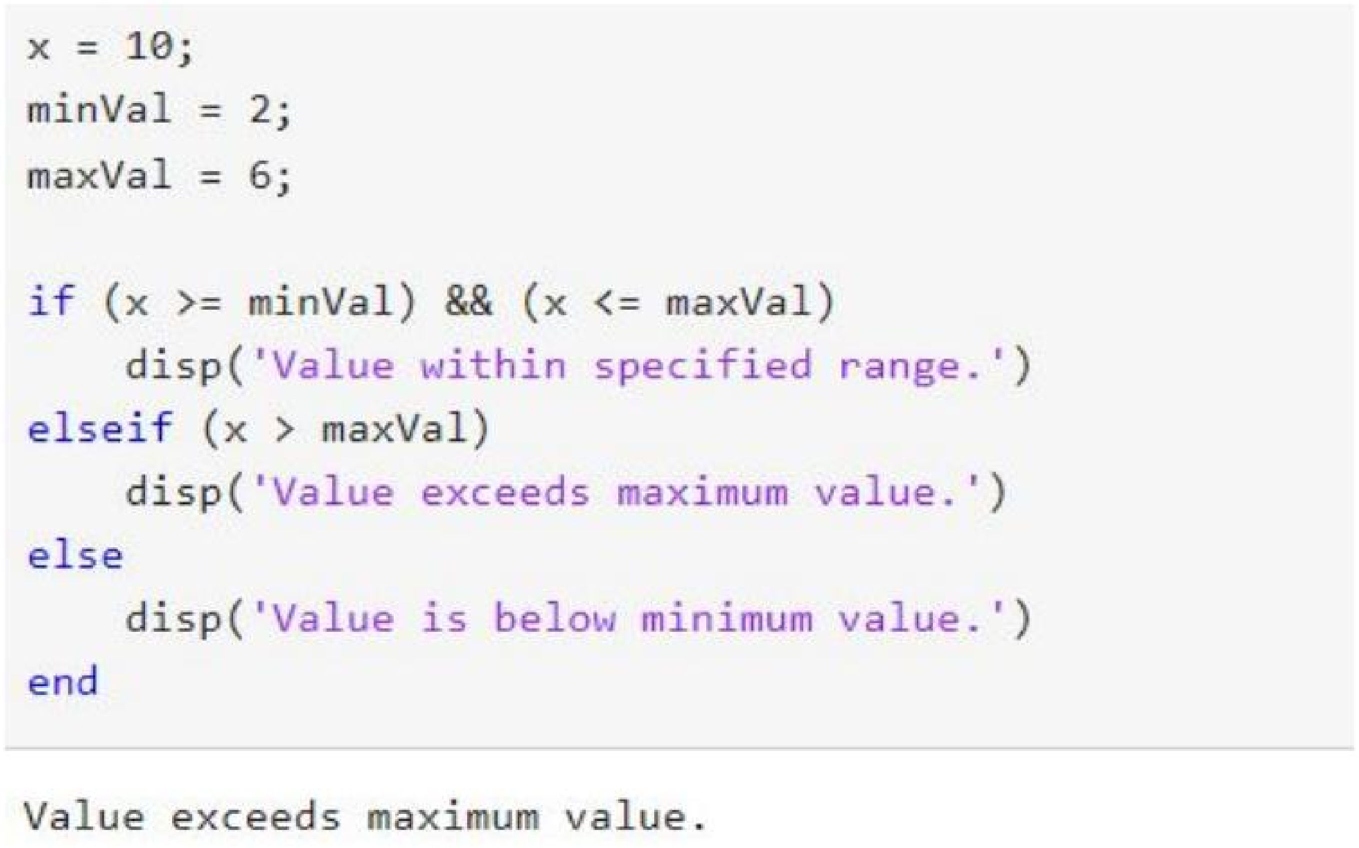
If `size(A)` and `size(B)` are the same, concatenate the arrays; otherwise, display a warning and return an empty array.

```
if isequal(size(A),size(B))
   C = [A; B]     % append
else
   disp('A and B are not the same size.')
   C = [];
end
```

```
A = 3x4
      1      1      1      1
      1      1      1      1
      1      1      1      1

B = 3x4
    0.8055    0.2399    0.4899    0.7127
    0.5767    0.8865    0.1679    0.5005
    0.1829    0.0287    0.9787    0.4711

C = 6x4
    1.0000    1.0000    1.0000    1.0000
    1.0000    1.0000    1.0000    1.0000
    1.0000    1.0000    1.0000    1.0000
    0.8055    0.2399    0.4899    0.7127
    0.5767    0.8865    0.1679    0.5005
    0.1829    0.0287    0.9787    0.4711
```

Evaluate multiple condition in an expression

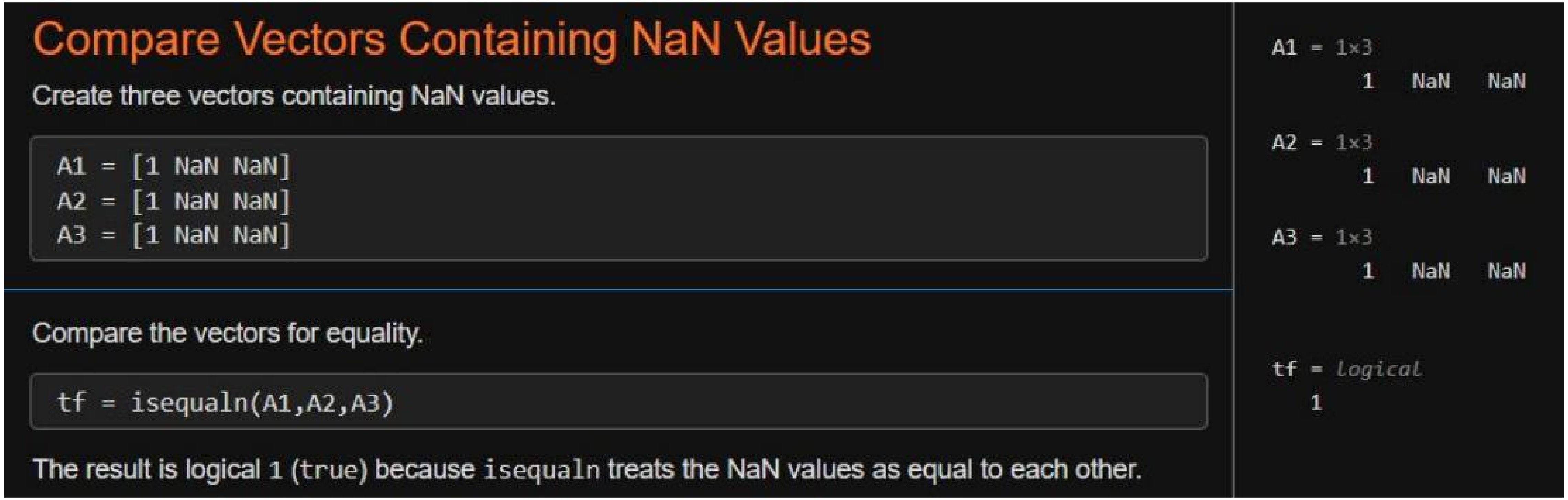
Q: Determine if a value falls within a specified range.



```
x = 10;
minVal = 2;
maxVal = 6;

if (x >= minVal) && (x <= maxVal)
    disp('Value within specified range.')
elseif (x > maxVal)
    disp('Value exceeds maximum value.')
else
    disp('Value is below minimum value.')
end
```

```
Value exceeds maximum value.
```

Evaluate multiple condition in an expression



Compare Vectors Containing NaN Values

Create three vectors containing NaN values.

```
A1 = [1 NaN NaN]
A2 = [1 NaN NaN]
A3 = [1 NaN NaN]
```

```
A1 = 1×3
     1   NaN   NaN

A2 = 1×3
     1   NaN   NaN

A3 = 1×3
     1   NaN   NaN
```

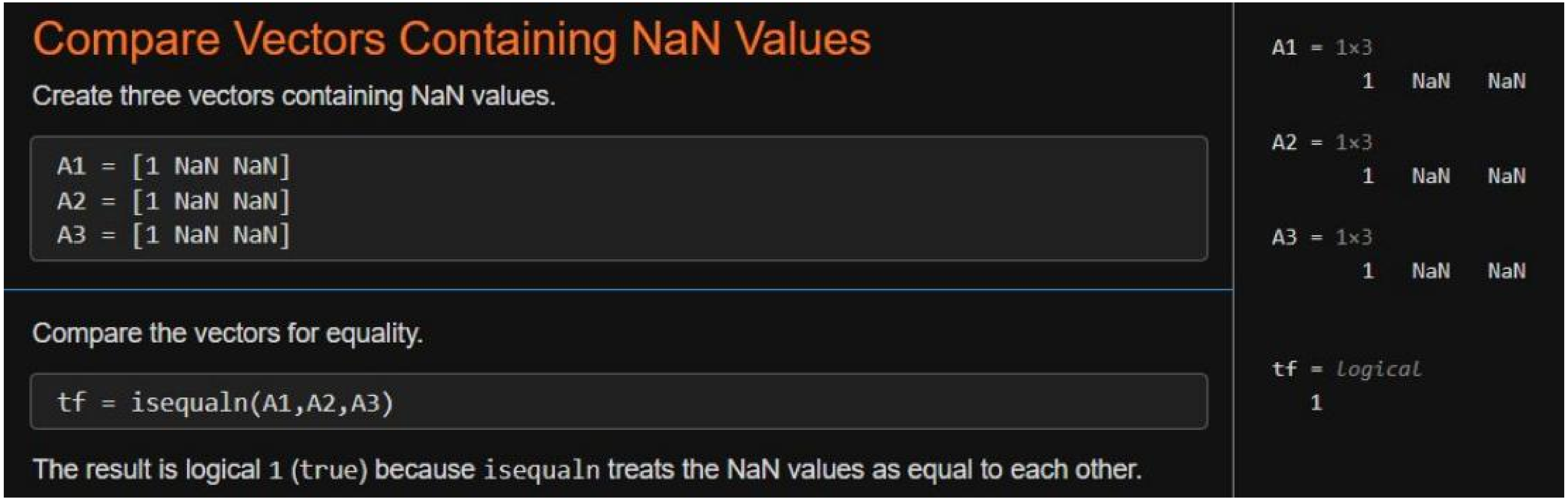
Compare the vectors for equality.

```
tf = isequaln(A1,A2,A3)
```

```
tf = logical
   1
```

The result is logical `1` (`true`) because `isequaln` treats the NaN values as equal to each other.

Evaluate multiple condition in an expression



Compare Vectors Containing NaN Values

Create three vectors containing NaN values.

```
A1 = [1 NaN NaN]
A2 = [1 NaN NaN]
A3 = [1 NaN NaN]
```

```
A1 = 1×3
     1    NaN    NaN

A2 = 1×3
     1    NaN    NaN

A3 = 1×3
     1    NaN    NaN
```

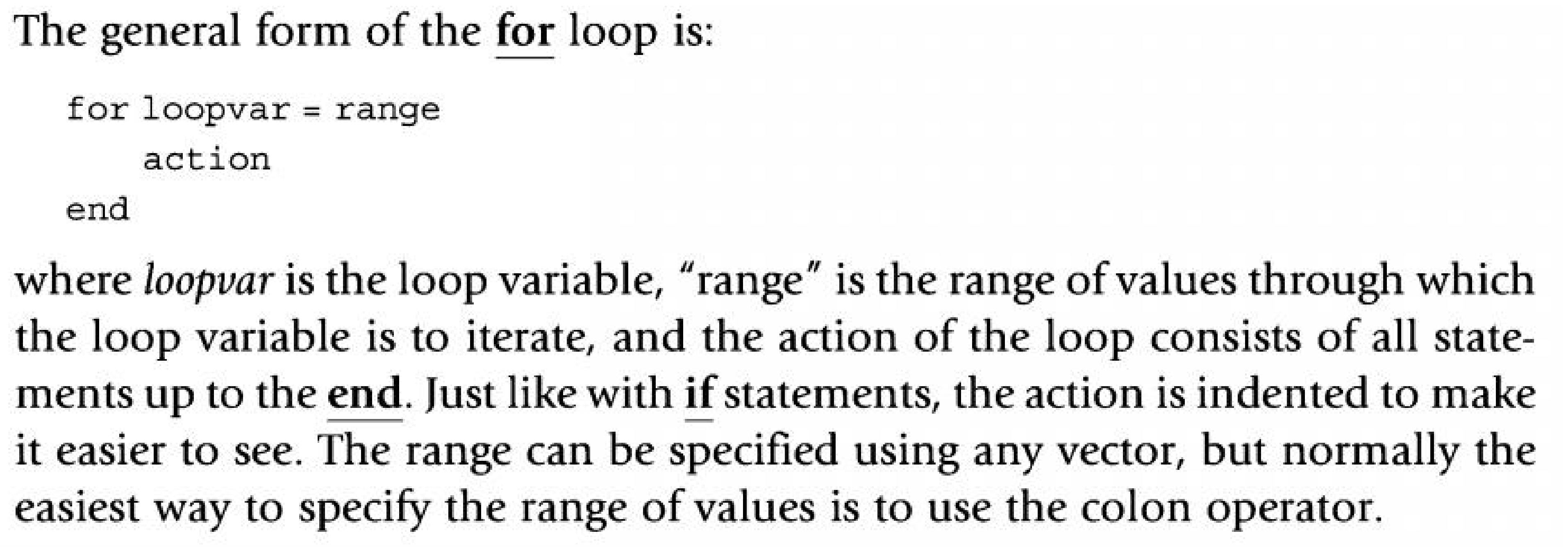
Compare the vectors for equality.

```
tf = isequaln(A1,A2,A3)
```

```
tf = logical
   1
```

The result is logical `1` (`true`) because `isequaln` treats the NaN values as equal to each other.

The “for” loop

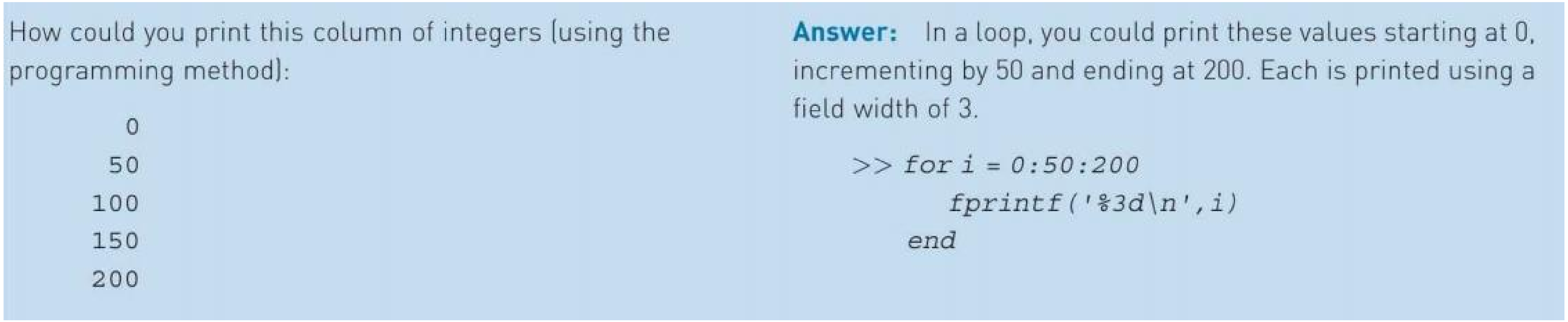


The general form of the **for** loop is:

```
for loopvar = range
    action
end
```

where *loopvar* is the loop variable, “range” is the range of values through which the loop variable is to iterate, and the action of the loop consists of all statements up to the **end**. Just like with **if** statements, the action is indented to make it easier to see. The range can be specified using any vector, but normally the easiest way to specify the range of values is to use the colon operator.

The “for” loop: Question



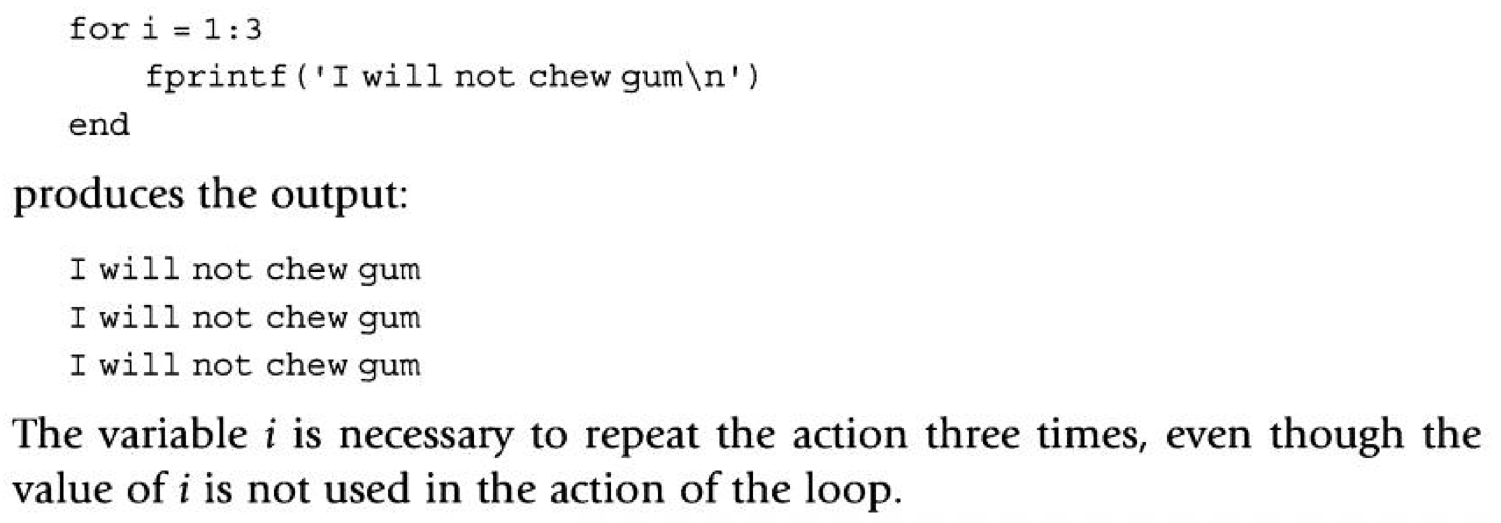
How could you print this column of integers (using the programming method):

```
  0
 50
100
150
200
```

Answer: In a loop, you could print these values starting at 0, incrementing by 50 and ending at 200. Each is printed using a field width of 3.

```
>> for i = 0:50:200
       fprintf('%3d\n',i)
   end
```

The “for” loop that don’t use iterator

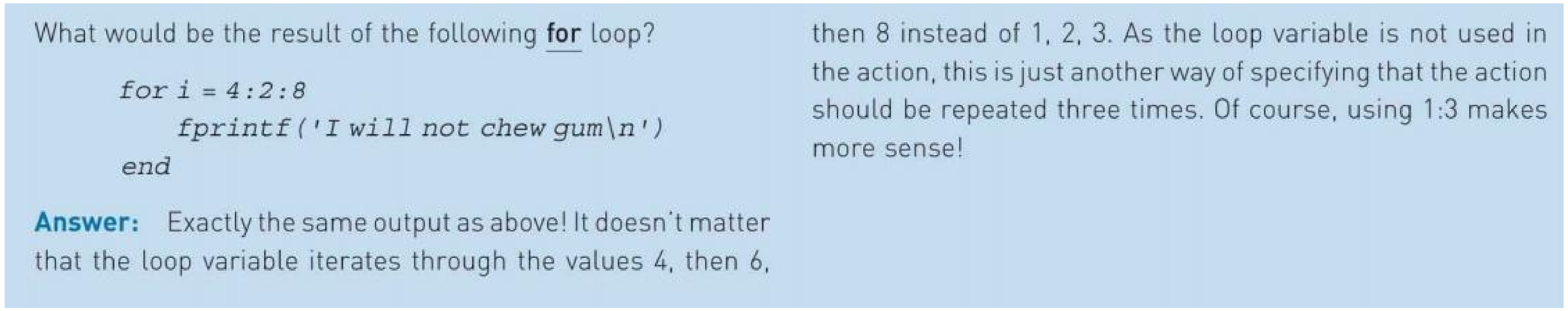


```
for i = 1:3
    fprintf('I will not chew gum\n')
end
```

produces the output:

```
I will not chew gum
I will not chew gum
I will not chew gum
```

The variable *i* is necessary to repeat the action three times, even though the value of *i* is not used in the action of the loop.



What would be the result of the following **for** loop?

```
for i = 4:2:8
    fprintf('I will not chew gum\n')
end
```

Answer: Exactly the same output as above! It doesn’t matter that the loop variable iterates through the values 4, then 6, then 8 instead of 1, 2, 3. As the loop variable is not used in the action, this is just another way of specifying that the action should be repeated three times. Of course, using 1:3 makes more sense!

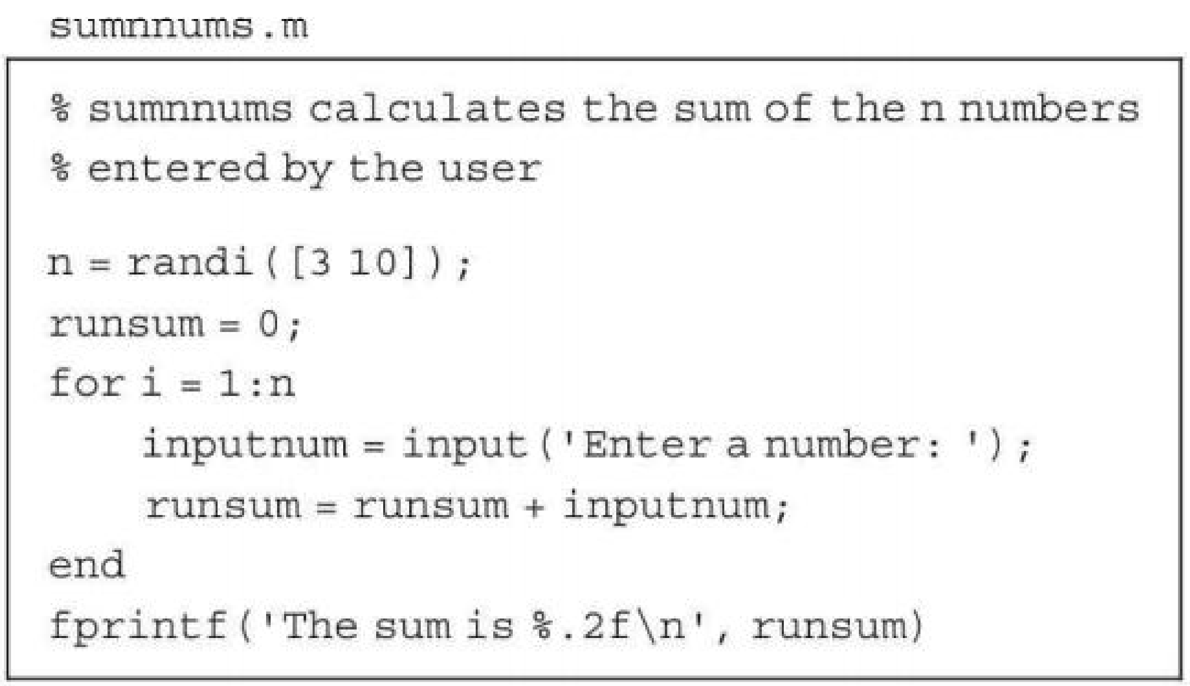
Input for “for” loop

forecho.m

```
% This script loops to repeat the action of
% prompting the user for a number and echo-printing it

for iv = 1:3
    inputnum = input('Enter a number: ');
    fprintf('You entered %.1f\n',inputnum)
end
```

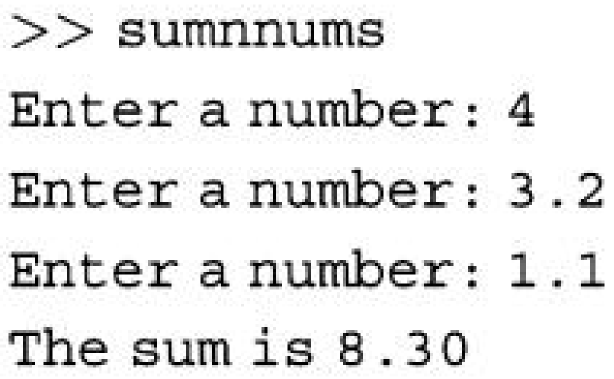
```
>> forecho
Enter a number: 33
You entered 33.0
Enter a number: 1.1
You entered 1.1
Enter a number: 55
You entered 55.0
```

Find sum and product



sumnnums.m

```
% sumnnums calculates the sum of the n numbers
% entered by the user

n = randi([3 10]);
runsum = 0;
for i = 1:n
    inputnum = input('Enter a number: ');
    runsum = runsum + inputnum;
end
fprintf('The sum is %.2f\n', runsum)
```



```
>> sumnnums
Enter a number: 4
Enter a number: 3.2
Enter a number: 1.1
The sum is 8.30
```

Preallocating vector

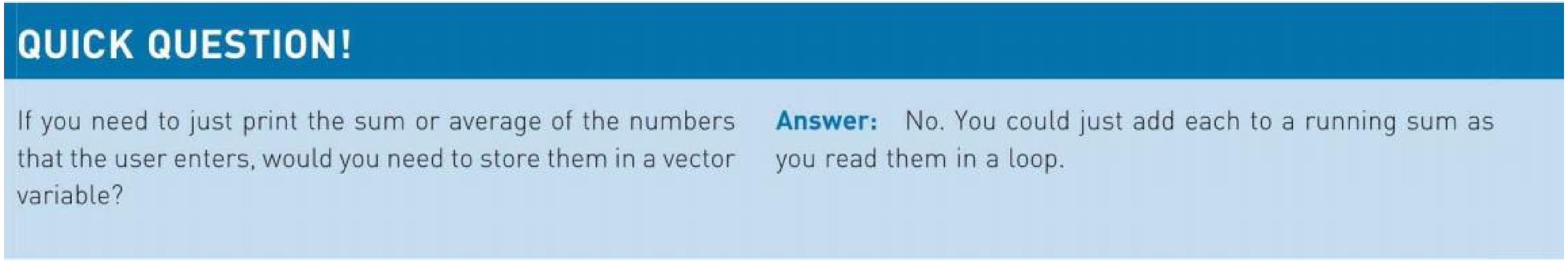
forgenvec.m

```
% forgenvec creates a vector of length n
% It prompts the user and puts n numbers into a vector

n = randi([4   8]);
numvec = zeros(1,n);
for iv = 1:n
    inputnum = input('Enter a number: ');
    numvec(iv) = inputnum;
end
fprintf('The vector is: \n')
disp(numvec)
```

```
>> forgenvec
Enter a number: 44
Enter a number: 2.3
Enter a number: 11
The vector is:
   44.0000    2.3000   11.0000
```

It is very important to notice that the loop variable iv is used as the index into the vector.

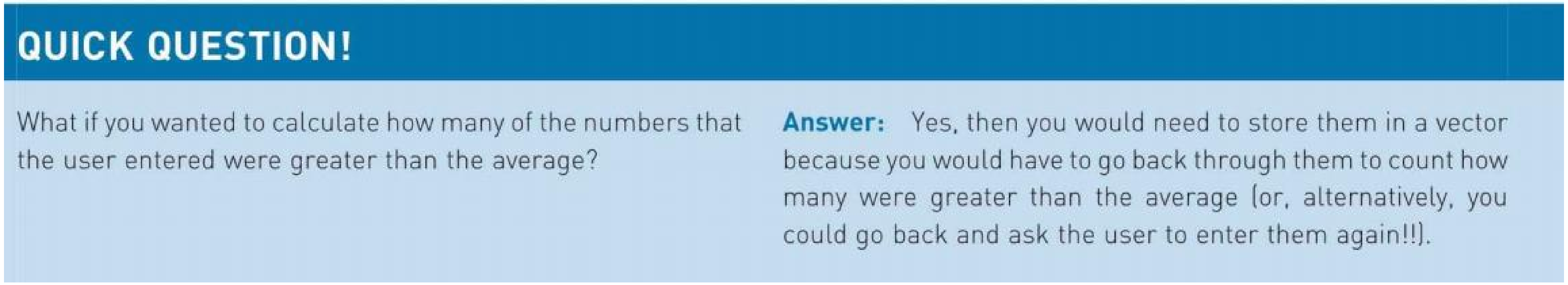
Question



QUICK QUESTION!

If you need to just print the sum or average of the numbers that the user enters, would you need to store them in a vector variable?

Answer: No. You could just add each to a running sum as you read them in a loop.



QUICK QUESTION!

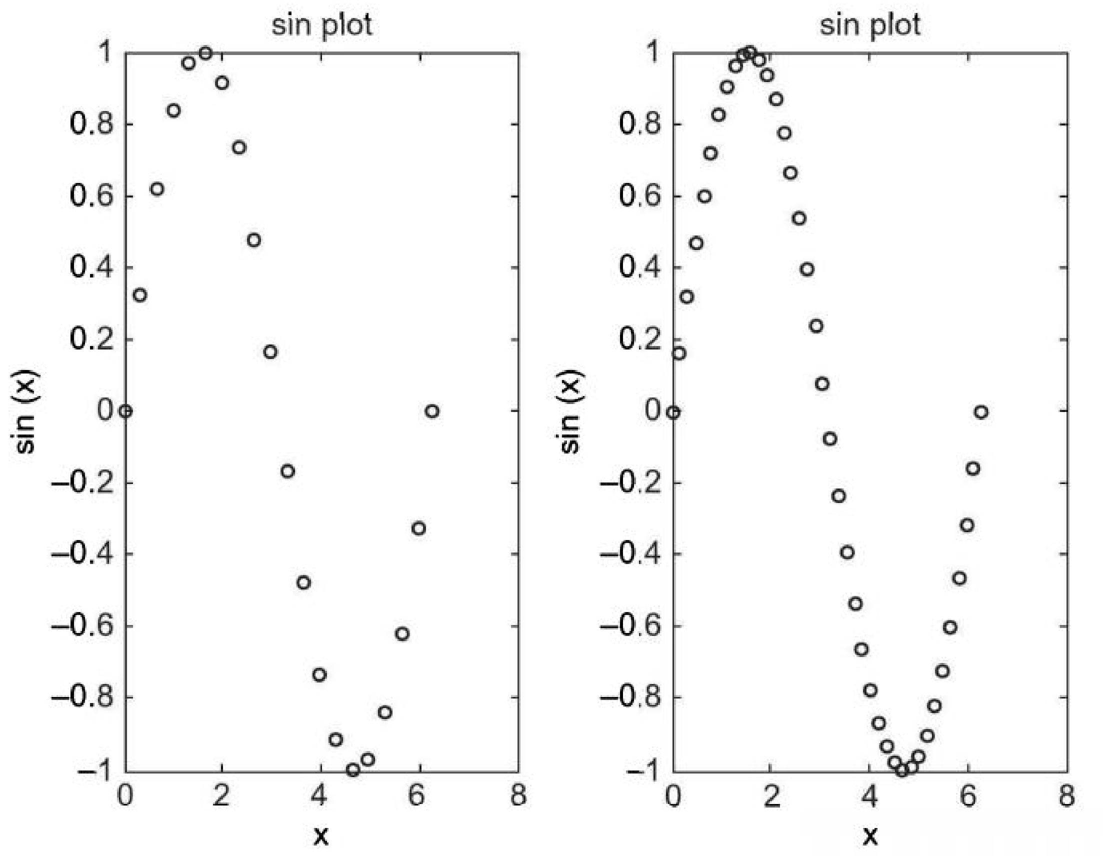
What if you wanted to calculate how many of the numbers that the user entered were greater than the average?

Answer: Yes, then you would need to store them in a vector because you would have to go back through them to count how many were greater than the average (or, alternatively, you could go back and ask the user to enter them again!!).

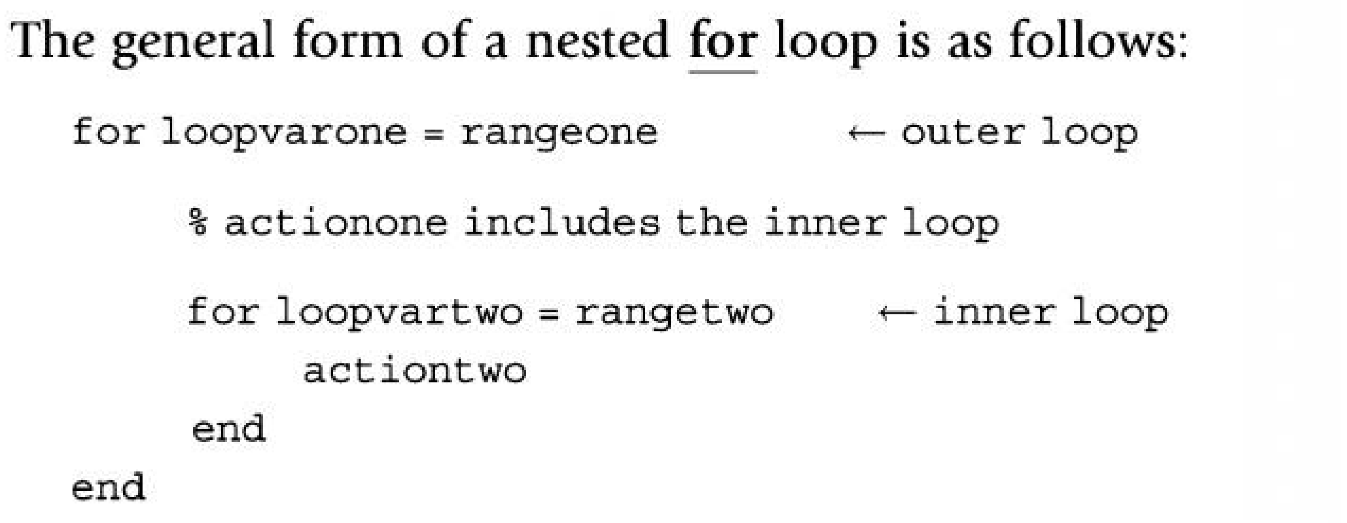
“for” loop-subplot

subplotex.m

```
% Demonstrates subplot using a for loop
for i = 1:2
    x = linspace(0,2*pi,20*i);
    y = sin(x);
    subplot(1,2,i)
    plot(x,y,'ko')
    xlabel('x')
    ylabel('sin(x)')
    title('sin plot')
end
```



Nested “for” loop



The general form of a nested **for** loop is as follows:

```
for loopvarone = rangeone            ← outer loop

    % actionone includes the inner loop

    for loopvartwo = rangetwo       ← inner loop
        actiontwo
    end
end
```

Nested "for" loop

- For every row of output:
  Print the required number of stars
  Move the cursor down to the next line (print '\n')

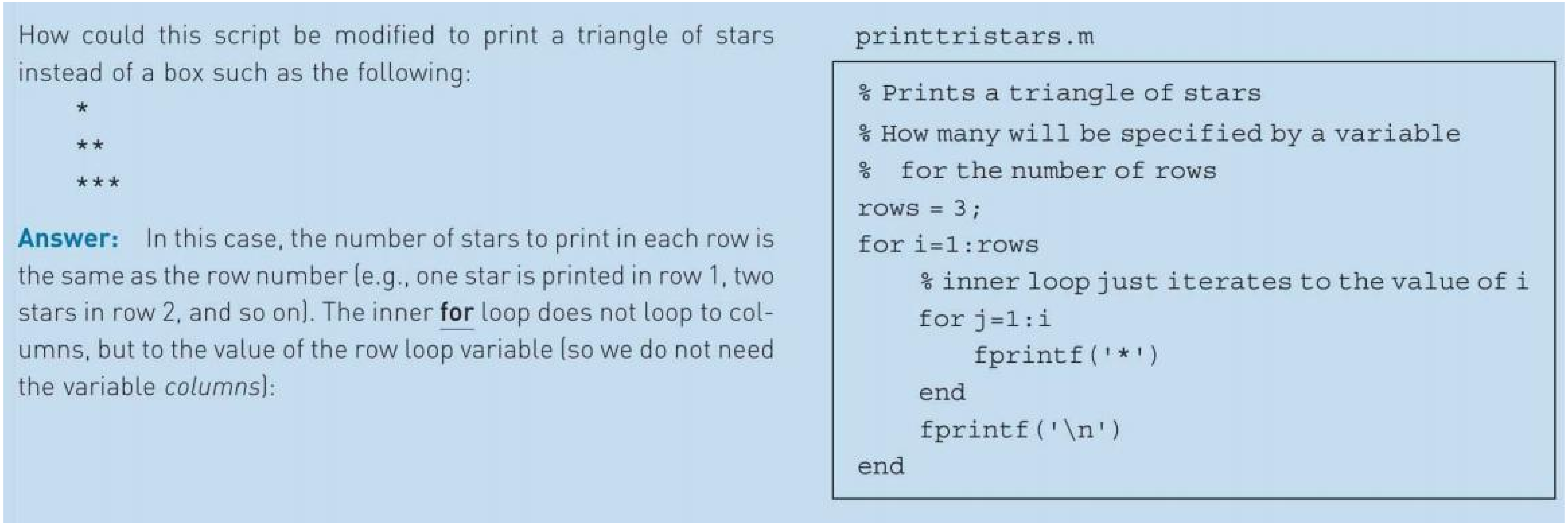
printstars.m

```
% Prints a box of stars
% How many will be specified by two variables
%  for the number of rows and columns

rows = 3;
columns = 5;
% loop over the rows
for i=1:rows
    %  for every row loop to print *'s and then one \n
    for j=1:columns
        fprintf('*')
    end
    fprintf('\n')
end
```

Executing the script displays the output:

```
>> printstars
*****
*****
*****
```

Nested “for” loop



How could this script be modified to print a triangle of stars instead of a box such as the following:

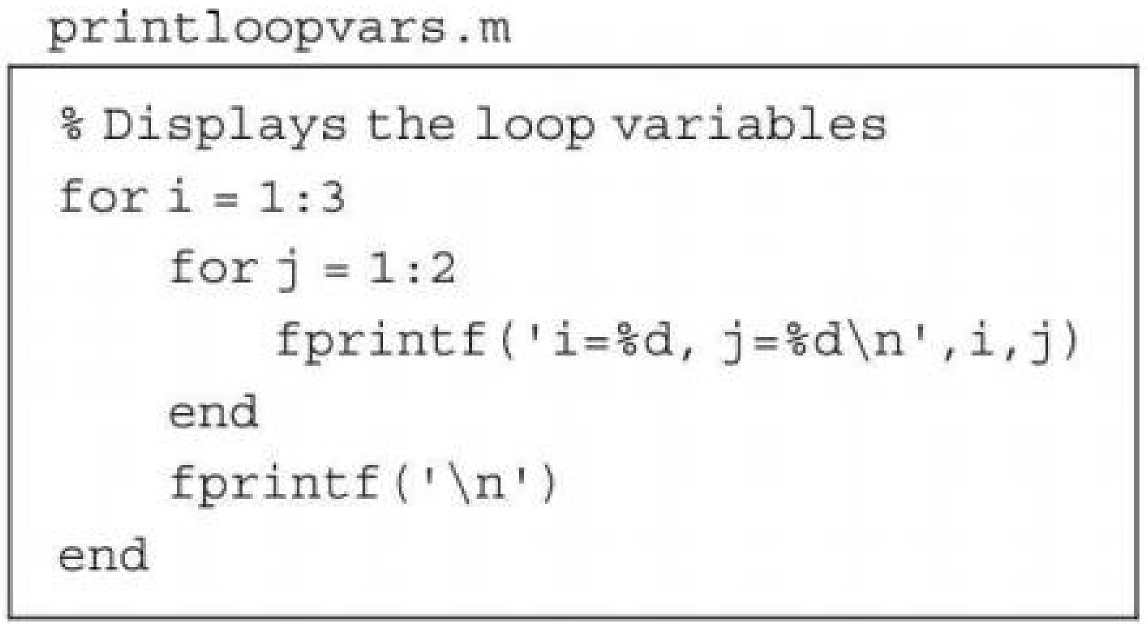
```
*
**
***
```

Answer: In this case, the number of stars to print in each row is the same as the row number (e.g., one star is printed in row 1, two stars in row 2, and so on). The inner **for** loop does not loop to columns, but to the value of the row loop variable (so we do not need the variable *columns*):

printtristars.m

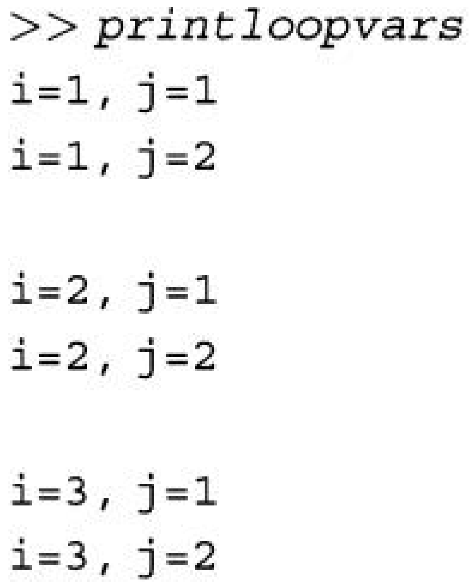
```
% Prints a triangle of stars
% How many will be specified by a variable
%  for the number of rows
rows = 3;
for i=1:rows
    % inner loop just iterates to the value of i
    for j=1:i
        fprintf('*')
    end
    fprintf('\n')
end
```

Nested “for” loop



printloopvars.m

```
% Displays the loop variables
for i = 1:3
    for j = 1:2
        fprintf('i=%d, j=%d\n',i,j)
    end
    fprintf('\n')
end
```



```
>> printloopvars
i=1, j=1
i=1, j=2

i=2, j=1
i=2, j=2

i=3, j=1
i=3, j=2
```

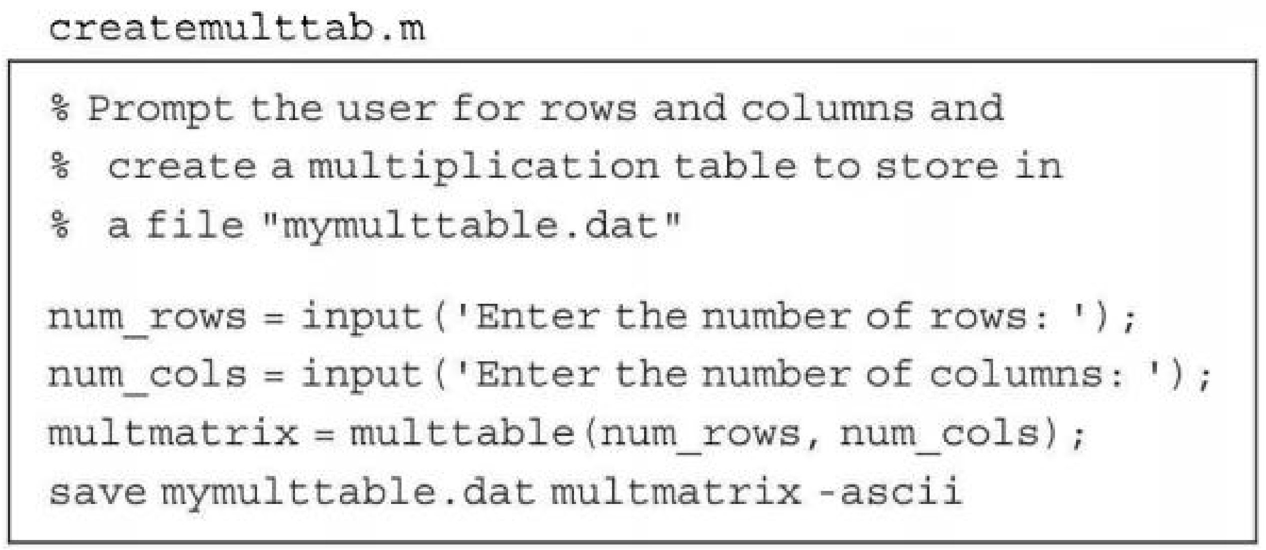
Nested “for” loop

multttable.m

```
function outmat = multtable(rows, columns)
% multtable returns a matrix which is a
% multiplication table
% Format: multtable(nRows, nColumns)

% Preallocate the matrix
outmat = zeros(rows,columns);
for i = 1:rows
    for j = 1:columns
        outmat(i,j)=i*j;
    end
end
end
```

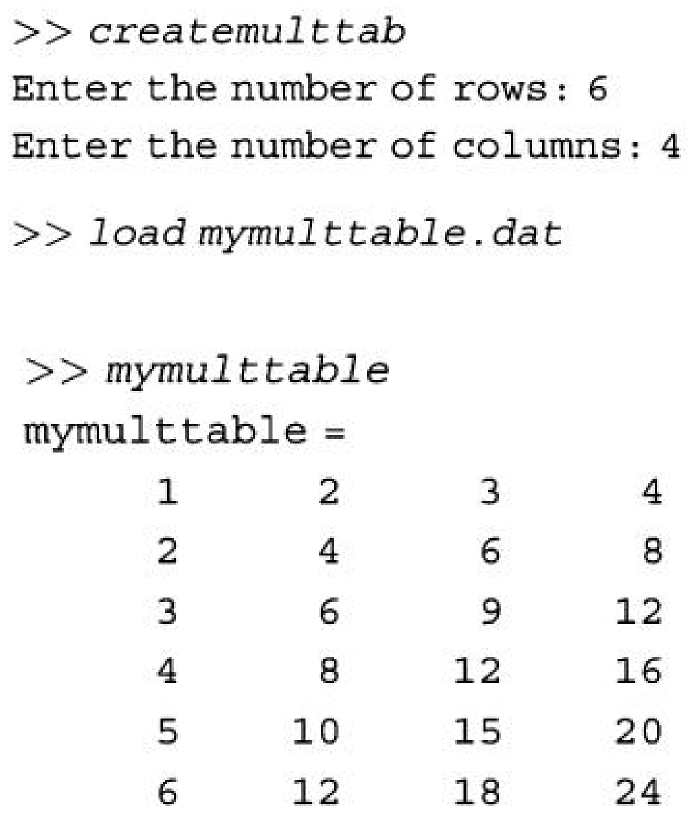
```
>> multtable(3,5)
ans =
     1     2     3     4     5
     2     4     6     8    10
     3     6     9    12    15
```

Nested “for” loop



```
createmulttab.m
```

```
% Prompt the user for rows and columns and
%  create a multiplication table to store in
%  a file "mymulttable.dat"

num_rows = input('Enter the number of rows: ');
num_cols = input('Enter the number of columns: ');
multmatrix = multtable(num_rows, num_cols);
save mymulttable.dat multmatrix -ascii
```



```
>> createmulttab
Enter the number of rows: 6
Enter the number of columns: 4

>> load mymulttable.dat


>> mymulttable
mymulttable =
        1         2         3         4
        2         4         6         8
        3         6         9        12
        4         8        12        16
        5        10        15        20
        6        12        18        24
```