Department of Chemical Engineering
ChE-101: Approaches to Chemical Engineering Problem Solving

MATLAB Tutorial III

Arrays and Matrix Operations/For Loops (last updated 4/27/06 by GGB)

Objectives:

These tutorials are designed to show the introductory elements for any of the topics discussed. In almost all cases there are other ways to accomplish the same objective, or higher level features that can be added to the commands below. Before working on this tutorial you should understand the concepts described in H-2 (flowchart diagrams) and H-3 (matrix operations).

Any text below appearing after the double prompt (>>) can be entered in the Command Window directly or in an m-file.

The following topics are covered in this tutorial;

Introduction (array definitions and uses)
Constructing
Terminology, Array Referencing, and Transposing
Array_Array Mathematics (Matrix Operations, see H-3 for details)
Element_by_Element, Array_Array Mathematics
Special Matlab functions for Arrays (length, sum, min, max)
Checking Arrays Stored in Memory
For loops (See H-2 for their description and their used in flowchart diagrams)
Solved Problems (guided tour)
Proposed Problems

Introduction: Array definitions and uses:

An Array is a list of numbers or expressions arranged in horizontal rows and vertical columns. When an array has only one row or column, it is called a vector. An array with n rows and m columns is called a matrix of size n x m. In computer programming arrays can be constituted by numerical and non-numerical variables. When an array is constituted by numerical variables it is called a matrix; See H-3 (Working with matrices and arrays for more details). Arrays are very useful in computer programming because they can be used to store information in an organized way which is easily accessible when needed.

In the majority of programming applications you will encounter, information is stored or provided to in the form of arrays. Arrays provide a convenient way to store information for calculations, printing, plotting, etc. There are many times when you want to perform the same operation on multiple numbers. Other times we want to do a calculation repeatedly and save the intermediate values. There are some built_in Matlab files which require the information in this form. This tutorial will explain the basics of creating, manipulating and performing calculations with arrays.

In this lab session we are going to work on the use of arrays. Last week we solved a problem to estimate the grades of the class. Because we were not using arrays it was not convenient to store

the grades (remember that we had to print the grades as soon as they were calculated). As an example of the use of arrays to enhance the performance of a code and to demonstrate their ability to store and access information we are going to modify that code (see solved problems) using arrays.
IMPORTANT: Brackets [ ] are used for matrix and array operations in Matlab. DO NOT USE parenthesis () to SPECIFY arrays and matrices.

Constructing Arrays
All the operations that are going to be demonstrated below work for the m files, function files, and command window. To avoid debugging m files, please practice the examples given below in the COMMAND WINDOW.

There are a number of ways to construct arrays:

Entering the values directly.

```
>> y = [  0.75  1.24  5.96  2.97  ]

y =

    0.7500    1.2400    5.9600    2.9700

>>
```

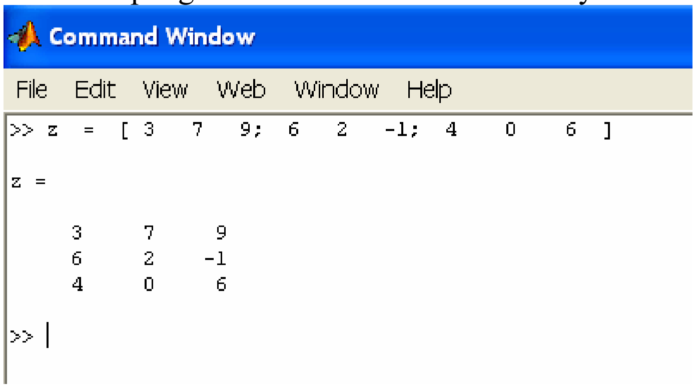
```
>> x = [ 0  0.1*pi  0.2*pi  0.3*pi  0.4*pi ]

x =

         0    0.3142    0.6283    0.9425    1.2566

>>
```

Note that mathematical operations can be performed while addressing the elements. Examples given above are row vectors.

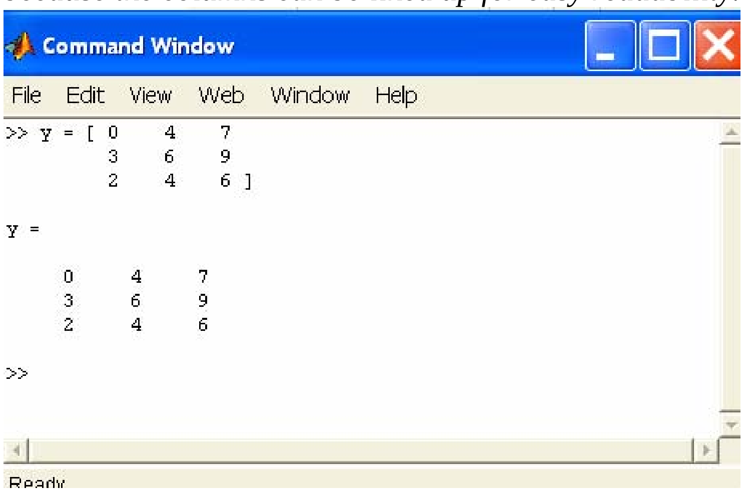
Each row of an array can be ended by a semi_colon.
The example given below is to create a column vector

```
>> x = [ 0;  0.1*pi;  0.2*pi;  0.3*pi;  0.4*pi ]

x =

         0
    0.3142
    0.6283
    0.9425
    1.2566

>>
```

The example given below is to create an array of numbers (also known as matrix):

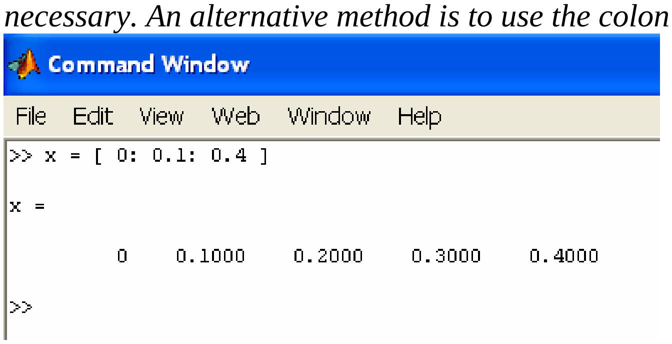


Rows can also be ended by using a return. This is the preferred way of creating large matrices, because the columns can be lined up for easy readability.



Equally Spaced Arrays

Typing every entry is time consuming for long arrays. If the points are equally spaced this isn't necessary. An alternative method is to use the colon notation [first_value: interval: last_value].



Notice that we have created an array with the first element being "0", the last element being "0.4" and the elements in between are spaced using and interval of 0.1
If the interval is left out Matlab uses a default interval of 1.

```
Command Window
File  Edit  View  Web  Window  Help
>> d = [ 2:8 ]

d =

     2     3     4     5     6     7     8

>>
```

*Another way of specifying an equally-spaced array is to use the **linspace** command. This allows you to specify the number of data points, but not the interval to be used. The notation is linspace(first_value, last_value, number_of_values):*

```
Command Window
File  Edit  View  Web  Window  Help
>> x = linspace( 0,  0.4,  5 )

x =

         0    0.1000    0.2000    0.3000    0.4000

>>
```

Arrays can be combined.

```
Command Window
File  Edit  View  Web  Window  Help
>> a = [ 0   1   2 ];
>> b = [ 4   6   8 ];
>> c= [a b]

c =

     0     1     2     4     6     8

>>
```

Notice that Array "c" is constituted by the elements of array "a" and "b".

Terminology, Array Referencing, and Transposing

Matlab uses the term array to indicate any set of information that is stored with a single variable name. In mathematics we typically speak of vectors or matrices. A vector is simply an array that has either a single row or column. A matrix can be of any size in terms of the rows or columns. We speak in terms of rows and columns, and indicate the size of arrays by specifying the (rows x columns). Six elements might be in any of the following forms: (1x6, 6x1, 2x3, 3x2)

Assigning and Addressing Array Elements

Individual array elements can be accessed and used in calculations. Parenthesis () is used to do this, see the example below

```
Command Window
File Edit View Web Window Help
>> x= [0 0.1 2 3 1 4];
>> y= [1 3 5
       2 1 9];
>> x(2)    %This will access the second element of vector x. Notice that parenthesis is used to access the particular element we are looking for.

ans =

    0.1000

>> y(2,3) %This will access the element stored in row 2 column 3 of the y array.

ans =

     9

>>
```

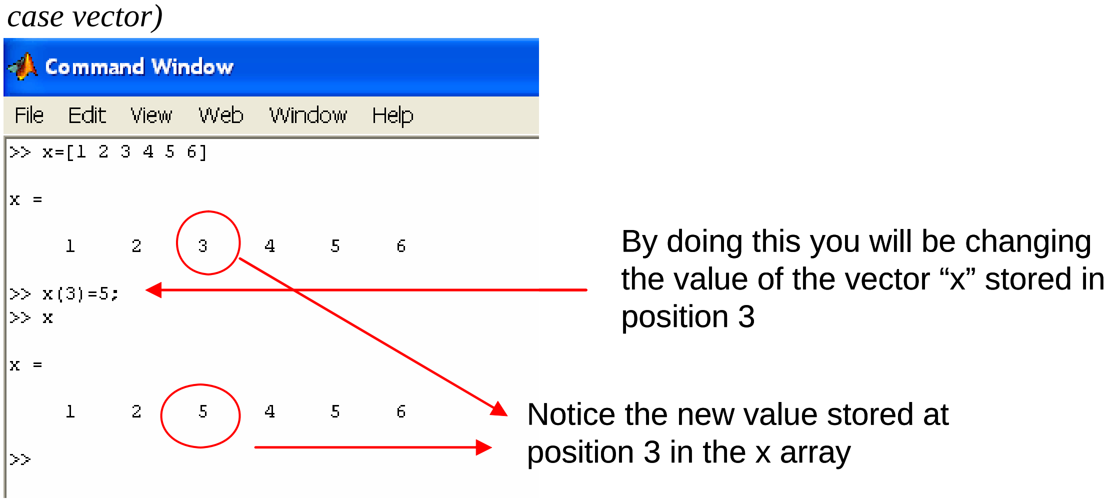
Notice that a "," is used to access elements in arrays because they have two dimensions (rows and columns).

IMPORTANT: You need to know the positions where your elements were stored to manipulate them.

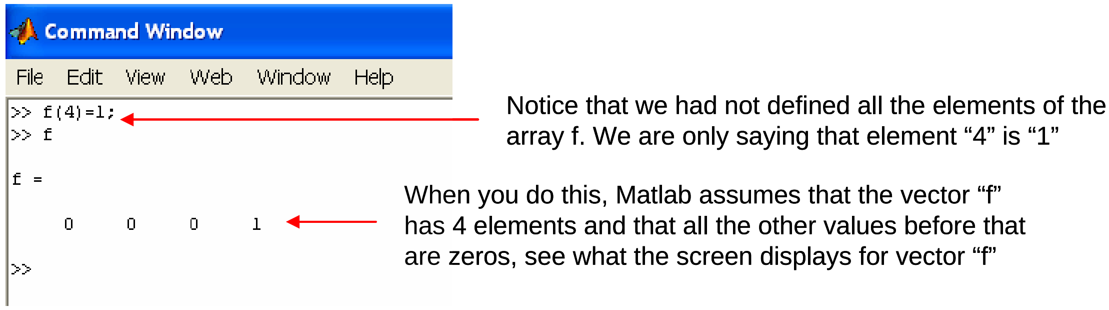
Now we can do calculations involving different elements. For example, using the arrays that we already created we can do the following calculations:

```
Command Window
File Edit View Web Window Help
>> x= [0 0.1 2 3 1 4];
>> y= [1 3 5
       2 1 9];
>> x(2)    %This will access the second element of vector x. Notice that parenthesis is used to access the particular element we are looking for.

ans =

    0.1000

>> y(2,3) %This will access the element stored in row 2 column 3 of the y array.

ans =

     9

>> t=2*x(2)+y(2,3)

t =

    9.2000

>> |
```

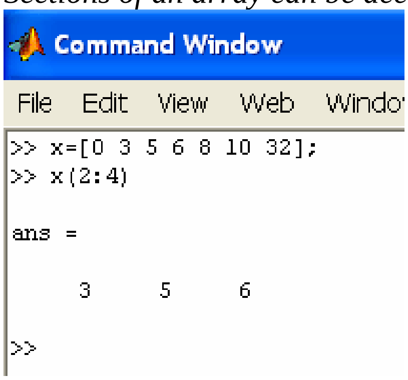
You can also manipulate and modify the elements stored in an array. See example below where we are changing the value of the element that was stored in position "3" of the array "x" (in this case vector)



If a value is assigned to a specific array element, any elements before that one which have not specifically been assigned a value, are given a default value of zero. See the example below

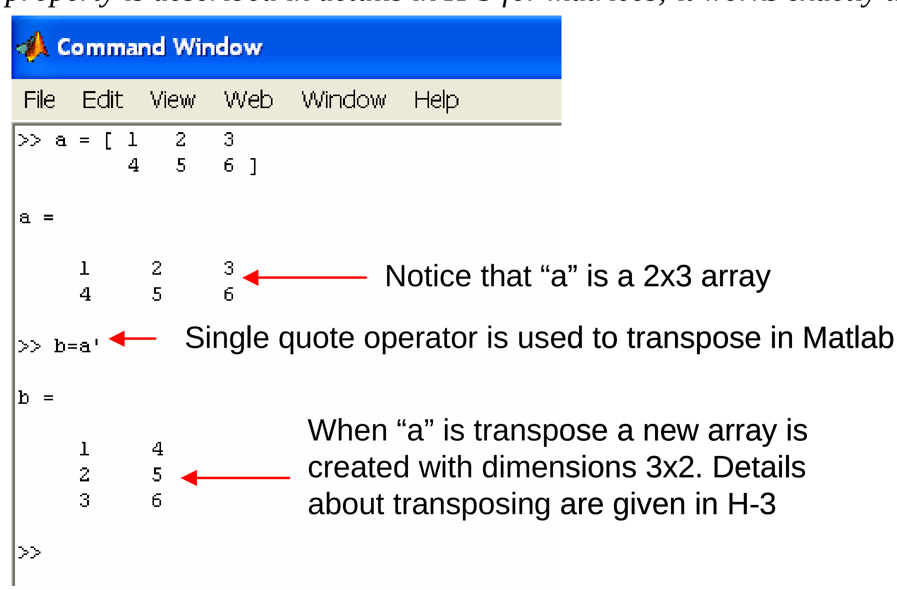


Sections of an array can be accessed. See example given below:



<u>Transposing</u>

The orientation of an array can be transposed by use of the single quote notation. The transpose property is described in details in H-3 for matrices; it works exactly the same for arrays



This is the transpose of 'a', and is a (3x2) array. The first row of 'a' becomes the first column of 'b', and so forth.

Array_Array Mathematics

This section explains how Matlab performs the Matrix operations that were described in H-3 (adding matrices, subtracting matrices, and multiplying matrices). You need to be careful about the dimensions of the matrices to perform the matrix operations (see H-3). Remember that to add and subtract matrices the matrices must have the same dimensions. Also there are special rules for the dimensions of the matrices in order to multiply them.
Examples of Matrix operations in Matlab are given below:

Example 1-Adding Matrices: Given the matrices, A, B, and C obtain matrix D = A + B and E=B+C

$$A=\begin{bmatrix}1 & 2\\ 2 & 3\end{bmatrix} \quad B=\begin{bmatrix}2 & 3\\ 4 & 1\end{bmatrix} \quad \mathrm{C}=\begin{bmatrix}2 & 2 & 1\\ 0 & 4 & -1\\ 2 & 5 & 0.5\end{bmatrix}$$

Command Window

File Edit View Web Window Help

```
>> A=[1 2
      2 3]; %creates matrix A
>> B=[2 3
      4 1]; %creates matrix B
>> C=[2 2 1
      0 4 -1
      2 5 0.5]; %creates matrix C
>> D=A+B %adds matrices A and B

D =

     3     5
     6     4

>> E=B+C %Can you do this operation. Notice the message
??? Error using ==> +
Matrix dimensions must agree.

>> |
```

+ is the operator to add matrices

Matrices B and C can't be added because they don't have the same dimensions (this was explained in H-3). Notice the error message given by Matlab, this indicates that there is a problem with the dimensions of the matrices used.

Example 2-Subtracting Matrices: Use the matrices A and B defined above to calculate F=B-A

Command Window

File Edit View Web Window Help

```
>> A=[1 2
      2 3]; %creates matrix A
>> B=[2 3
      4 1]; %creates matrix B
>> C=[2 2 1
      0 4 -1
      2 5 0.5]; %creates matrix C
>> D=A+B %adds matrices A and B

D =

     3     5
     6     4

>> E=B+C %Can you do this operation. Notice the message
??? Error using ==> +
Matrix dimensions must agree.

>> F=B-A

F =

     1     1
     2    -2

>>
```

- is the operator to subtract matrices

Example 3-Multiplication of a matrix by a scalar: Given the matrices A, B, C, D. Perform the following calculations:

F=A-B+0.5D

G=2(A+B+C)-D

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad C = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad D = \begin{bmatrix} -1 & -1 \\ -1 & 0 \end{bmatrix}$$

```
Command Window
File  Edit  View  Web  Window  Help
>> A=[1 2
      2 3]; %creates matrix A
>> B=[1 1
      1 1]; %creates matrix B
>> C=[0 1
      1 0]; %creates matrix C
>> D=[-1 -1
      -1 0]; %creates matrix D
>> F=A-b+0.5*D %notice what happens if we try to do this. Can you forsee a problem?
??? Error using ==> -
Matrix dimensions must agree.

>> F=A-B+0.5*D %notice the answer now

F =

   -0.5000    0.5000
    0.5000    2.0000

>> G=2*(A+B+C)-D %calculates G

G =

     5     9
     9     8

>>
```

Even though the error message indicates that there is a problem with the dimensions of the matrix, the real problem is that matrix “b” does not exist. Remember that Matlab is case sensitive then “b” is not the same that “B”. This was probably a typo from the user. **Message: check that you don’t have typos before checking the dimensions of your matrix**

Notice how the matrix “D” is multiplied by the scalar. The operator used is *

Example 4-Multiplication of a matrices: Given the matrices X and Y, Perform the following calculations: P = X Y, T= X Y

$$X = \begin{bmatrix} 3 & 1 \\ 8 & 6 \\ 0 & 4 \end{bmatrix} \quad \mathrm{Y} = \begin{bmatrix} 5 & 9 \\ 7 & 2 \end{bmatrix}$$

```
Command Window
File  Edit  View  Web  Window  Help
>> X=[3 1
      8 6
      0 4]; %creates X matrix
>> Y=[5 9
      7 2]; %creates Y matrix
>> P=X*Y  %Notice the operator * is used to multiply matrices

P =

    22    29
    82    84
    28     8

>> T=Y*X %Can you do this? Notice the message. What is the problem?
??? Error using ==> *
Inner matrix dimensions must agree.

>> |
```

* Is the operator used to multiply matrices

The problem is that the dimensions of the matrices X and Y do not allow Y to be multiplied by X. See H-3 for more details

Element_by_Element, Array_Array Mathematics

We need to be careful when we are performing array_array calculations. Addition and subtraction of arrays or matrices only exist on an element_by element basis. This means the operation is performed between corresponding elements of the arrays. To do this the arrays must be of the same size and orientation.

Addition and subtraction of element_by_element in arrays in the same operation described in Matrix operations (see Array_Array Mathematics section above).

Element_by_element multiplication and division of arrays is different to what it was described in the Array_Array Mathematics section. For doing this operation the matrices need to have the same dimensions.

For element-by-element multiplication and division the notation needs to be changed because there are other types of matrix multiplication (see Array_Array Mathematics section). The notation for element_by_element operations uses a (. or ./).*

See example below to perform element-by-element operations in arrays

Command Window

File Edit View Web Window Help

```
>> A=[1 1
      1 1];
>> B=[2 2
      2 2];
>> C=A.*B %This is an element-by-element product

C =

     2     2
     2     2

>> C=B.*A %Notice that the answer is the same because it is an element-element product

C =

     2     2
     2     2

>> D=A*B %Notice that the answer is different to C because this is other type of product. This is multiplication of matrices

D =

     4     4
     4     4

>> E=D./C %This is how you perform an element-by-element division

E =

     2     2
     2     2

>>
```

You can also power all the elements of an array by using .* See the example below:

Command Window

File Edit View Web Window H

```
>> A=[2 2
      2 2]

A =

     2     2
     2     2

>> B=A.^2

B =

     4     4
     4     4

>> |
```

You can calculate sin, square root, exponential, etc of the elements of an array by:

Command Window

File Edit View Web Window Help

```
>> A=[1 1
      2 2];
>> B=sin(A)

B =

    0.8415    0.8415
    0.9093    0.9093

>> C=sqrt(A)

C =

    1.0000    1.0000
    1.4142    1.4142

>> D=exp(A)

D =

    2.7183    2.7183
    7.3891    7.3891
```

Calculates the sin of each of the elements stored in A

Calculates the square root of each of the elements stored in A

Calculates the exponential of each of the elements stored in A

Special Matlab functions for Arrays (length, sum, min, max)
Matlab has a number of built-in functions that are of use. Some of those are presented here.

<u>The ‘length’ command</u>

The length command will identify the number of elements in a vector.

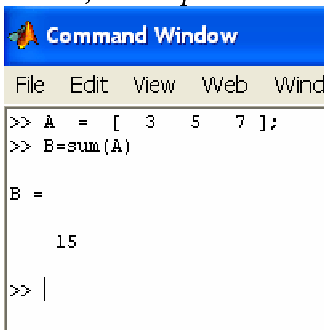
Command Window

File Edit View Web Window Help

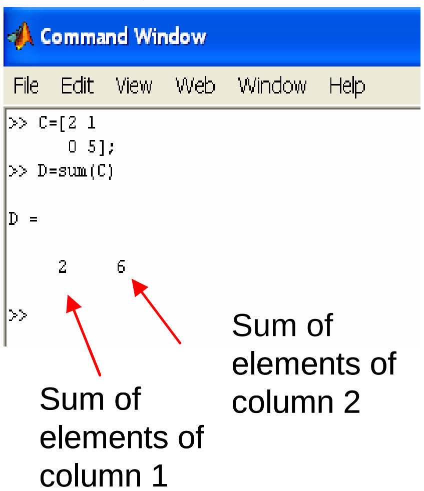
```
>> A  =  [  12  43  24  75  13  54 ];
>> Z=length (A)

Z =

     6

>>
```

<u>The 'sum' command</u>

The elements of an array or matrix can be summed using the sum command. If the array is a vector, it will produce a single value:

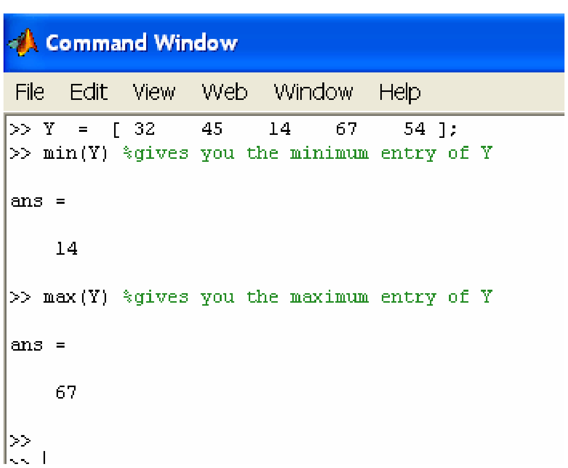


If the array has two dimensions it will sum all the elements of each column and it will produce two values:

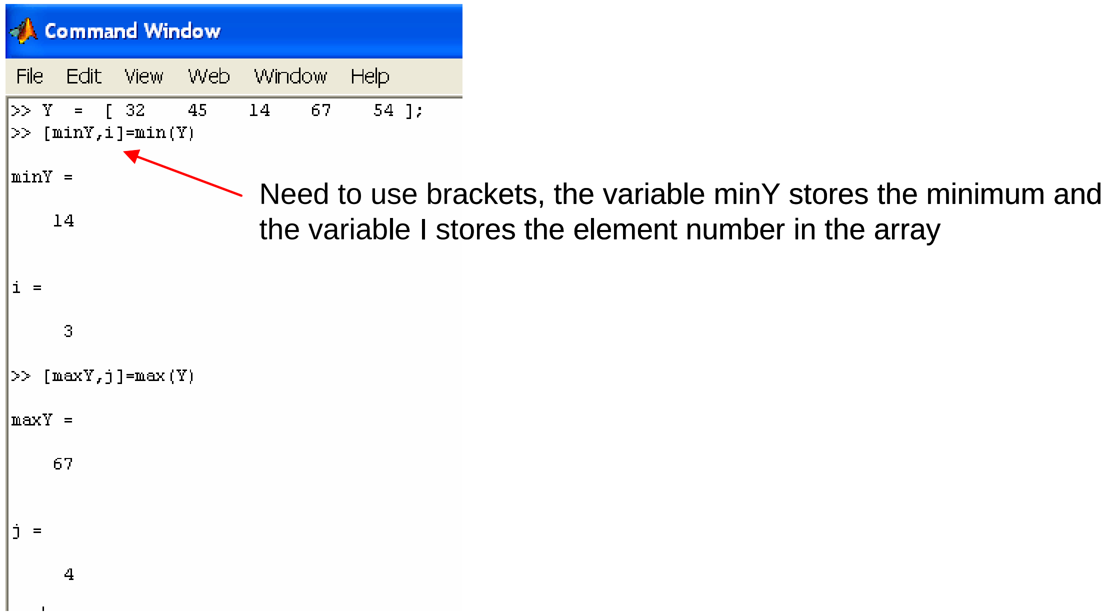


<u>The 'min' and 'max' commands</u>

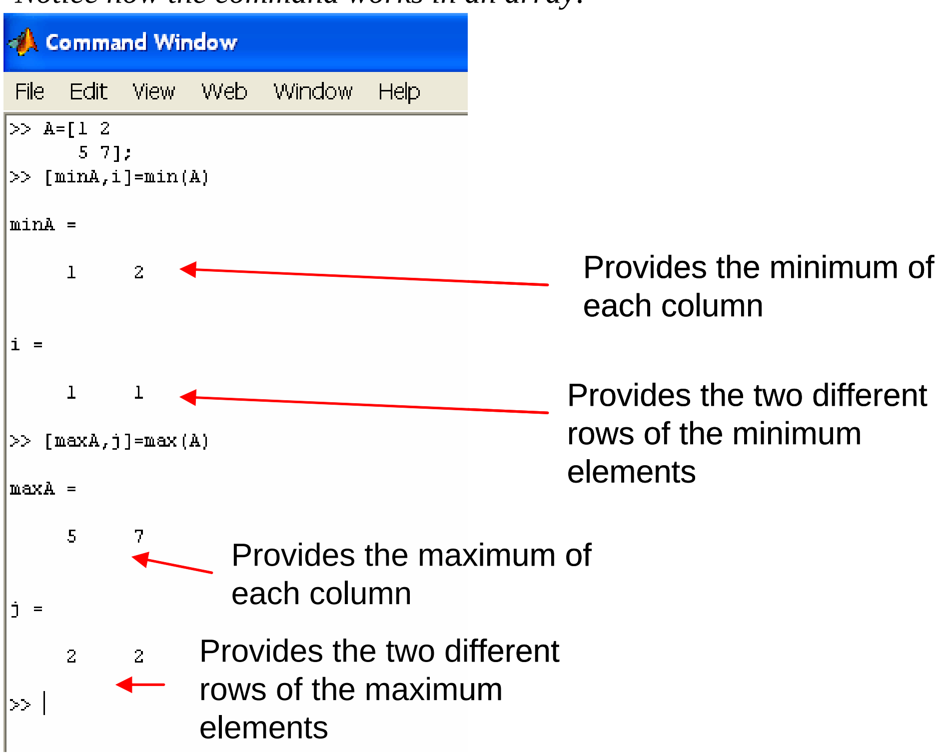
The 'min' and 'max' commands can be used to locate the minimum or maximum entry in a vector.



You can also have the 'min' or 'max' command give you which element of the array is identified.



Notice how the command works in an array:



More examples

```
>> D=[1 2 3
     0 5  1
     7 1  2];
>> [minD,i]=min(D)

minD =

     0     1     1


i =

     2     3     2

>> [maxD,j]=max(D)

maxD =

     7     5     3


j =

     3     2     1

>> |
```

Minimums in rows 2, 3 and 2, respectively for each column

Checking Arrays Stored in Memory

Whereas using the 'who' command gives a list of variables, the 'whos' command gives the additional information of the array dimensions. For example:

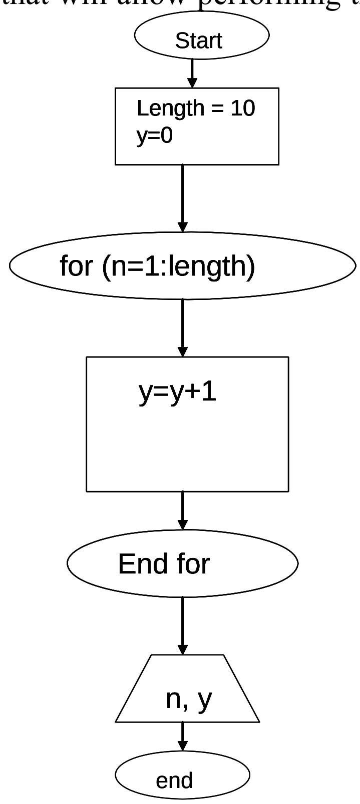
```
Command Window
File  Edit  View  Web  Window  Help
>> A=[2 3
      1 1]

A =

     2     3
     1     1

>> B=[1 1 1 1];
>> C=2*B

C =

     2     2     2     2

>> whos
  Name      Size           Bytes  Class

  A         2x2               32  double array
  B         1x4               32  double array
  C         1x4               32  double array

Grand total is 12 elements using 96 bytes
```

For Loops (See Chapter 7 of the book, section 7.4.1)

for loops allow for a group of MATLAB commands to be repeated for a fixed, predetermined number of times.

The range for the loop counter is set as follows: *for n=1:5* (This means that n will start at 1 and the loop will be repeated until n = 5. Each time the loop runs it automatically adds 1 to the loop counter "n"

Example 5: Write a Matlab algorithm that will allow performing the operations described in the flowchart diagram given below:



This is the m file (example_5_TIII)

C:\Documents and Settings\Dr Gerri Botte\My Documents\travel folder\che 101\Spring-05\Tutorials\Matlab files\example_5_TIII.m

File Edit View Text Debug Breakpoints Web Window Help

```
% This program calculates the operations described in the flowchart diagram Tutorial III Example5
% Using for loops in Matlab
% Developed by Gerardine Botte
% Created on: 04/21/05
% Last modified on: 04/21/05
% Che-101, Spring 05
%--------------------------------
clc
clear

%-------------------------------------------------------------------------------------
disp ('This program operations described in the flowchart diagram Tutorial III Example5')
disp ('Using for loops in Matlab')
disp ('Developed by Gerardine Botte')
disp ('Che-101, Spring 05')
disp ('Solution to Solved Problem 1, Tutorial II')
disp ('The user will be prompt to input variables depending on his/her choice')
disp ('_______________________________________________________________')
%-------------------------------------------------------------------------------------

%Program calculations begin in this section
Lenght=10;
y=0;
for (n=1:Lenght)
    y=y+1;
end
disp ('n=')
disp (n)
disp ('y=')
disp (y)
```

This is what the program will display

```
Command Window
File Edit View Web Window Help

This program operations described in the flowchart diagram Tutorial III Example5
Using for loops in Matlab
Developed by Gerardine Botte
Che-101, Spring 05
Solution to Solved Problem 1, Tutorial II
The user will be prompt to input variables depending on his/her choice
_____________________________________________________________________________
n=
    10

y=
    10

>> |
```

If you want a different increment instead of 1 you should use the following *for s = 1.0: -0.1: 0.0* This means that the loop will start with s = 1.0, it will decrease the value of s until s = 0 by subtracting -0.1 each time the loop is repeated.

See example given below:

1. M file

```
C:\Documents and Settings\Dr Gerri Botte\My Docu
File Edit View Text Debug Breakpoints Web

clc
clear

y=0;
for (n=12:-2:0)
    y=y+1;
end
disp ('n=')
disp (n)
disp ('y=')
disp (y)
```

2. Results

```
Command Window
File Edit View Web Window Help

n=
     0

y=
     7

>>
```

You cannot short_circuit the loop by reassigning the loop variable, n, within the loop.

Arrays can be used and manipulated by using for loops. This is an example of how it works:.

1. M file

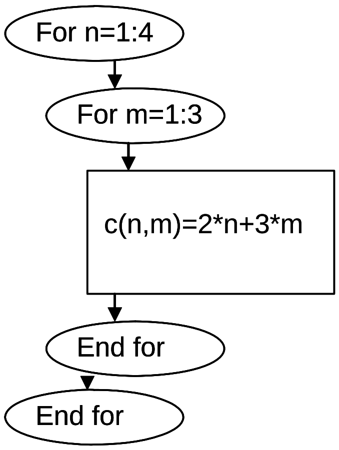
```
C:\Documents and Settings\Dr Gerri Botte\My Documents\travel folder\che 101\Spring-05\Tutorials\Matlab files\test.
File Edit View Text Debug Breakpoints Web Window Help

clc
clear

data = [ 3   9   45   6 ]; %array to be manipulated
for n = 1:4
    result(n) = data(n)*3; %This creates a new array "results' that consits of multiplying
                           %each element of "data" by 3
end
result
```

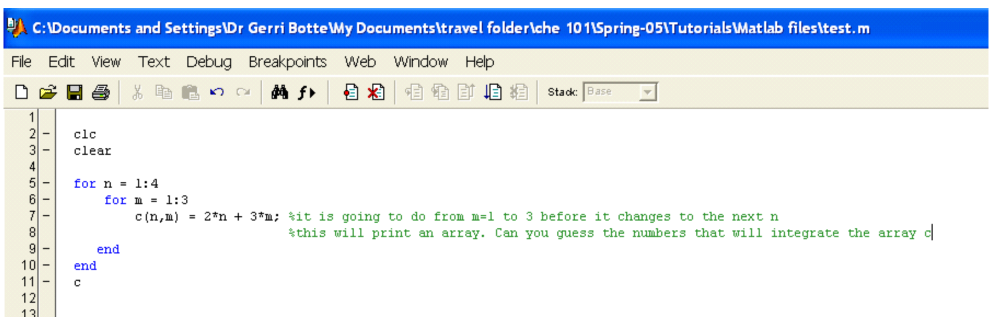
2. Results

```
Command Window
File Edit View Web Window Help

result =

     9    27   135    18

>>
```

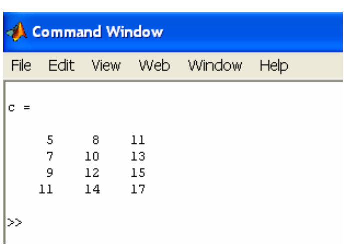
By keeping 'n' as an integer loop counter it is much easier to organize and access arrays.

***for loops** can be nested, ie. stacked within each other*. See the example given below:
The flowchart diagram can be coded as





M file



results

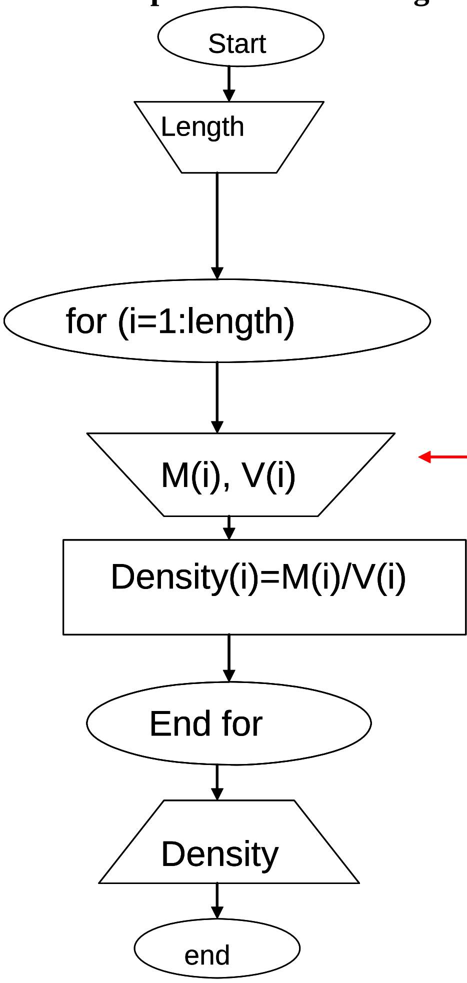
Input and manipulate data using for loops: for loops can be used to create vectors that store information. This information can then be manipulated/restore/and-or printed. For example, the table given below has information on the mass and the volume of the samples. Create a Matlab program that will calculate the density of the different samples.

Sample #	1	2	3	4
Mass (g)	10	25	50	15
Volume (ml)	100	300	1000	800

<u>Solution</u>:

1. Follow the “tips for solving problems”
2. Write a flowchart diagram (see H-2).

Solution Option A: Calculating density vector <u>one element</u> at a time



Legend:

Length= number of samples (for loop control)

i= counter of the for loop

M= vector that stores the mass of each sample, g

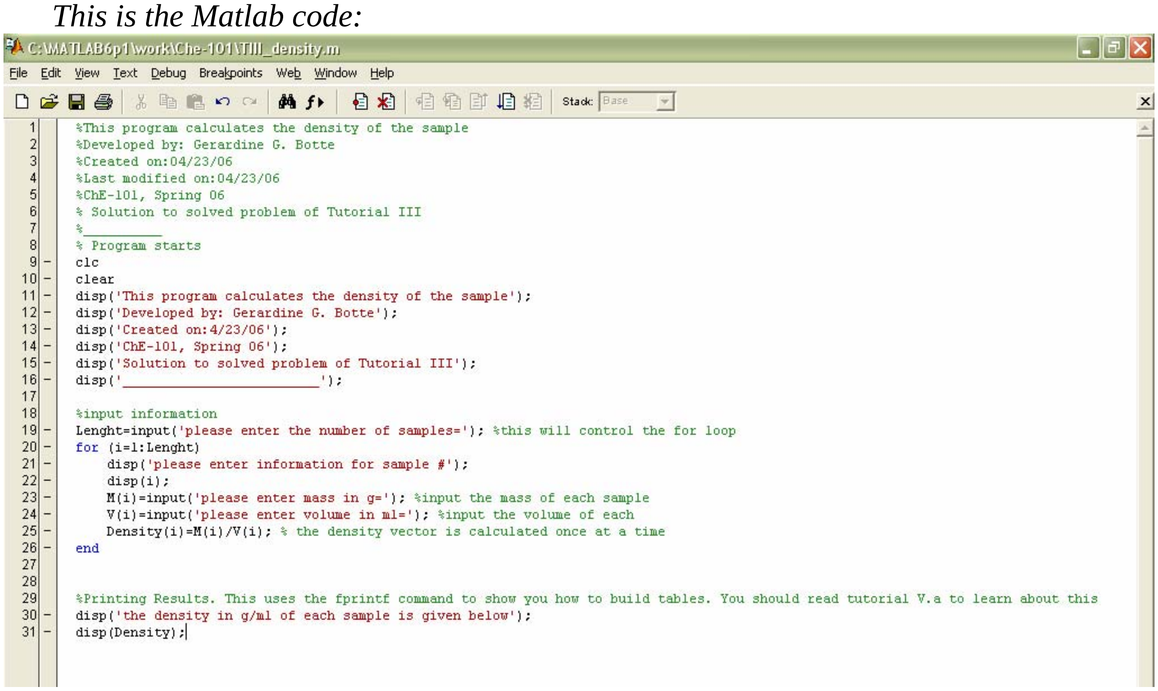
V= vector that stores the volume of each sample, ml

Density= vector that stores the density of each sample, g/ml

In this case we will input the each element of the vectors "M" and "V" individually (as explained on pages 5 and 6 of this tutorial

Option A: In this case we are calculating each element of the vector density inside the for loop by using each of the elements of the M and V vectors on an individual basis

This is the Matlab code:

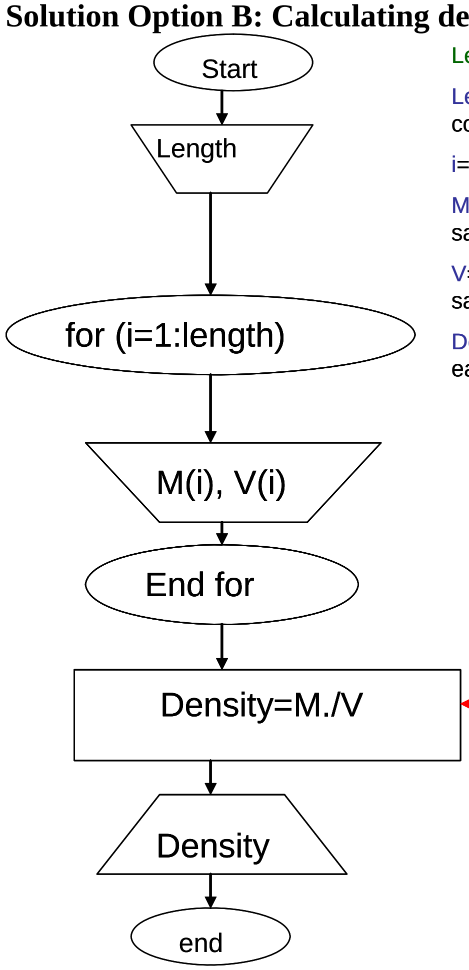


```
%This program calculates the density of the sample
%Developed by: Gerardine G. Botte
%Created on:04/23/06
%Last modified on:04/23/06
%ChE-101, Spring 06
% Solution to solved problem of Tutorial III
%___________
% Program starts
clc
clear
disp('This program calculates the density of the sample');
disp('Developed by: Gerardine G. Botte');
disp('Created on:4/23/06');
disp('ChE-101, Spring 06');
disp('Solution to solved problem of Tutorial III');
disp('__________________________');

%input information
Lenght=input('please enter the number of samples='); %this will control the for loop
for (i=1:Lenght)
    disp('please enter information for sample #');
    disp(i);
    M(i)=input('please enter mass in g='); %input the mass of each sample
    V(i)=input('please enter volume in ml='); %input the volume of each
    Density(i)=M(i)/V(i); % the density vector is calculated once at a time
end


%Printing Results. This uses the fprintf command to show you how to build tables. You should read tutorial V.a to learn about this
disp('the density in g/ml of each sample is given below');
disp(Density);
```

This is what will show up on the command screen:

```
ork\Che-101

Command Window
This program calculates the density of the sample
Developed by: Gerardine G. Botte
Created on:4/23/06
ChE-101, Spring 06
Solution to solved problem of Tutorial III
_________________________
please enter the number of samples=4
please enter information for sample #
     1

please enter mass in g=10
please enter volume in ml=100
please enter information for sample #
     2

please enter mass in g=25
please enter volume in ml=300
please enter information for sample #
     3

please enter mass in g=50
please enter volume in ml=1000
please enter information for sample #
     4

please enter mass in g=15
please enter volume in ml=800
the density in g/ml of each sample is given below
    0.1000    0.0833    0.0500    0.0187

>> |
```

Solution Option B: Calculating density vector <u>all</u> <u>elements</u> at a time



Legend:

Length= number of samples (for loop control)

i= counter of the for loop

M= vector that stores the mass of each sample, g

V= vector that stores the volume of each sample, ml

Density= vector that stores the density of each sample, g/ml

Option B: In this case we are calculating all the elements of the density vector outside the for loop by using the element_by_element array mathematics as explained on p. 9 of this tutorial

This is the Matlab code:

```
C:\MATLAB6p1\work\Che-101\TIII_density_B.m

%This program calculates the density of the sample
%Developed by: Gerardine G. Botte
%Created on:04/23/06
%Last modified on:04/23/06
%ChE-101, Spring 06
% Solution to solved problem of Tutorial III
%__________
% Program starts
clc
clear
disp('This program calculates the density of the sample');
disp('Developed by: Gerardine G. Botte');
disp('Created on:4/23/06');
disp('ChE-101, Spring 06');
disp('Solution to solved problem of Tutorial III');
disp('_________________________');

%input information
Lenght=input('please enter the number of samples='); %this will control the for loop
for (i=1:Lenght)
    disp('please enter information for sample #');
    disp(i);
    M(i)=input('please enter mass in g='); %input the mass of each sample
    V(i)=input('please enter volume in ml='); %input the volume of each
end

%Calculations
Density=M./V; % all the elements of the density vector are calculated at a time by using element_by_element array mathematics

%Printing Results. This uses the fprintf command to show you how to build tables. You should read tutorial V.a to learn about this
disp('the density in g/ml of each sample is given below');
disp(Density);
```

This is what will show up on the command window:

```
Command Window
This program calculates the density of the sample
Developed by: Gerardine G. Botte
Created on:4/23/06
ChE-101, Spring 06
Solution to solved problem of Tutorial III
_________________________
please enter the number of samples=4
please enter information for sample #
     1

please enter mass in g=10
please enter volume in ml=100
please enter information for sample #
     2

please enter mass in g=25
please enter volume in ml=300
please enter information for sample #
     3

please enter mass in g=50
please enter volume in ml=1000
please enter information for sample #
     4

please enter mass in g=15
please enter volume in ml=800
the density in g/ml of each sample is given below
    0.1000    0.0833    0.0500    0.0187

>>
```

SOLVED PROBLEMS

1. Write a program in Matlab to calculate the sum of the first n terms of the series: $\sum_{k=1}^{n} \frac{(-1)^k k}{2^k}$.

Execute the program for n= 4 and n=20

<u>Solution</u>:

3. Follow the "tips for solving problems"
4. Write a flowchart diagram (see H-2). In the space given below draw your flowchart diagram

5. Write the code in Matlab. See the solution given below.

C:\Documents and Settings\Dr Gerri Botte\My Documents\travel folder\che 101\Spring-05\Tutorials\Matlab files\prob_1_TIII.m

File Edit View Text Debug Breakpoints Web Window Help

Stack: Base

```
% This program solves Problem 1 of Tutorial III: executes the sum
% Using for loops in Matlab
% Developed by Gerardine Botte
% Created on: 04/21/05
% Last modified on: 04/21/05
% Che-101, Spring 05
%--------------------------------
clc
clear

%--------------------------------------------------------------------------------
disp ('This program solves problem 1 of tutorial III')
disp ('Using for loops in Matlab')
disp ('Developed by Gerardine Botte')
disp ('Che-101, Spring 05')
disp ('Solution to Solved Problem 1, Tutorial II')
disp ('The user will be prompt to input variables depending on his/her choice')
disp ('____________________________________________________________')
%--------------------------------------------------------------------------------

%Program calculations begin in this section
n=input('enter the number of times you want to perform calculation= '); %this is the n defined in the equation
sum=0; %This initializes the accumulator sum which is going to store the results of the sum operation

for (k=1:n)
    sum=sum+(-1)^k*k/2^k; %this performs the calculation
end
disp ('sum=')
disp (sum)
```

Matlab code

Results for n=4

Command Window

File Edit View Web Window Help

```
This program solves problem 1 of tutorial III
Using for loops in Matlab
Developed by Gerardine Botte
Che-101, Spring 05
Solution to Solved Problem 1, Tutorial II
The user will be prompt to input variables depending on his/her choice
____________________________________________________________
enter the number of times you want to perform calculation= 4
sum=
   -0.1250

>>
```

Results for n=20

Command Window

File Edit View Web Window Help

```
This program solves problem 1 of tutorial III
Using for loops in Matlab
Developed by Gerardine Botte
Che-101, Spring 05
Solution to Solved Problem 1, Tutorial II
The user will be prompt to input variables depending on his/her choice
____________________________________________________________
enter the number of times you want to perform calculation= 20
sum=
   -0.2222

>>
```

PROPOSED PROBLEMS

Dr. Botte has decided to ask her ChE-101 class to help her in preparing a Matlab program that will allow calculating the grades for the class and report important statistics by the end of the quarter. Your task will be to write the program that Dr. Botte will use

Here is what Dr. Botte would like her code to be able to do:

1. Calculate the grade of each student based on the different course assignments, exams, and class participation (she will provide the total points for each item for each student in the 100 scale). The weight percentage of the grades activities is as follows: 40% homework, 55% Exams, 5% class participation. Dr. Botte wants the grade of the students to be stored in an array. That way all the grades can be calculated at the end of the program.
2. Provide the arithmetic average for the final grades of the class in the 100 scale. Do the calculation using arrays
3. Calculate the percentage of students that obtain A, B, C, D, and F grades given the following scale:

Final Grade	Scale
A	Grade ≥ 90
B	79≤ Grade <90
C	70≤ Grade <79
D	60≤ Grade <70
F	Grade <60

4. Test your program with the following grades:

Student	Homework	Exams	Class Participation
1	100	98	69
2	86	60	80
3	99	86	100
4	66	89	70

HINT: The idea of this problem is to modify the code that you built last week but using arrays, this will show you how arrays and vectors can help when building the code and manipulating results.