ENGR 0012 – Engineering Problem Solving

Goals for this week:

- Incorporate and manipulate images
  - Create functions
- Solve systems of equations

Please submit your HW!

Please submit through both new and old submission systems

In addition to if statements, you can do the same thing through a different method: switch-case-otherwise

Structure of “Switch”:

```
switch expression
        case test 1
                commands
        case test 2
                commands
        …
        otherwise
                commands
end
```

Switch statements differ from if statements:

- Used when choosing between specific values
- Cannot perform <, >, etc., only ==
- Can somewhat minimize typing

Using if statement versus switch-case-otherwise

```
%Enter number
user_num=input('Please enter a number: ');

%Check if odd or even
remainder=rem(user_num,2);

%Display message, using if statement
if remainder==0
    disp('You entered an even number!')
else
    disp('You entered an odd number!')
end

%Display message, using switch case
switch remainder
    case 0
        disp('You entered an even number!');
    otherwise
        disp('You entered an odd number!');
end
```

You can upload images to MATLAB!

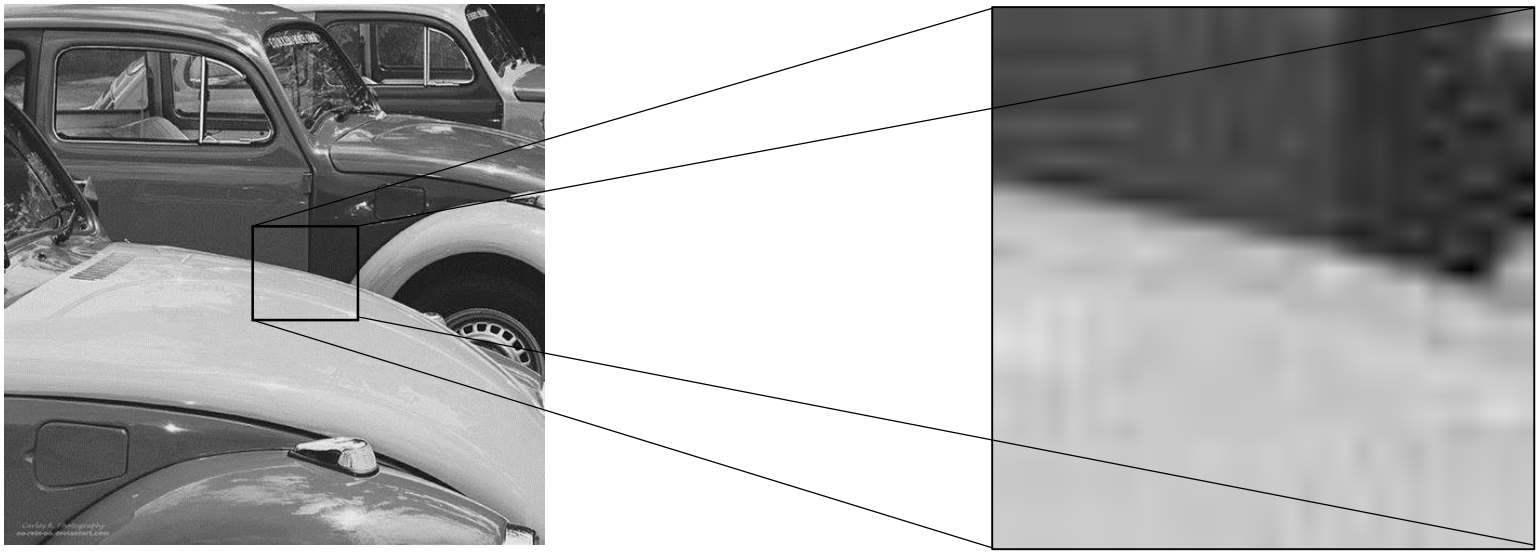
- Let's create a program that will ask the user to provide a number, determine if it is even or odd, display the appropriate message, and display an image of an even or odd number

You can upload images to MATLAB!

```
clear
clc

%Enter number
user_num=input('Please enter a number: ');

%Check if odd or even
remainder=rem(user_num,2);

%Read images
num1=imread('Num1.png');
num2=imread('Num2.png');

%Display message, using switch case
switch remainder
    case 0
        disp('You entered an even number!');
        imshow(num2);
    otherwise
        disp('You entered an odd number!');
        imshow(num1);
end
```

Images are stored as arrays of numbers (pixels)

- Images are made up of several pixels (zoom in enough to note)
- Image files are stored and read as an array of numbers, each number representing a single pixel intensity



500 x 500 pixel grey scale image

30 x 30 pixel crop

From slides by M. Mandala

Pixels and intensities mapping

30x30 pixel crop grey scale

78	78	78	79	80	81	81	82	80	80	80	78	77	77	79	81	88	89	91	70	56	56	51	53	55	56	55	53	107	130
77	78	78	79	80	81	81	82	78	78	78	77	75	75	77	78	77	78	82	64	53	56	51	53	66	63	57	82	130	172
70	70	71	72	73	74	75	75	76	76	76	75	74	73	75	76	69	71	76	60	52	56	50	52	43	56	66	86	67	72
75	75	76	77	78	79	79	80	74	75	76	75	73	73	74	75	71	72	75	60	52	57	50	51	72	59	57	71	57	63
73	73	74	74	75	75	76	76	74	75	76	76	74	73	74	75	73	72	75	59	52	57	49	49	57	37	45	52	58	56
71	71	71	71	71	71	71	71	74	76	77	77	75	74	74	75	71	70	73	58	52	57	49	49	47	45	65	59	58	41
81	80	80	79	79	78	77	77	73	75	77	77	75	74	73	74	72	71	73	58	52	58	49	48	63	51	42	43	59	65
74	73	72	71	70	69	68	68	72	74	76	76	74	73	73	73	77	74	76	59	53	58	48	47	46	61	50	52	50	58
68	72	78	83	84	81	78	76	76	75	75	73	72	71	70	70	74	70	73	61	54	56	46	49	56	50	53	53	47	54
86	74	57	47	48	57	68	75	79	77	75	75	77	79	80	80	70	66	69	56	51	55	48	52	54	49	57	61	44	40
173	163	148	132	115	95	73	58	58	57	59	64	71	74	72	69	79	73	73	57	50	54	48	52	53	45	42	44	49	57
191	191	190	188	185	179	168	160	137	118	92	73	65	65	66	66	71	70	77	67	60	59	46	45	57	50	55	59	42	22
197	199	200	199	197	196	195	195	188	184	179	172	159	137	111	92	69	60	55	39	36	50	55	66	46	58	50	37	64	112
179	190	202	206	203	199	196	195	204	198	191	187	188	188	187	186	173	156	137	98	69	54	35	32	36	63	58	46	91	158
204	204	202	197	190	186	186	187	191	191	191	192	192	192	190	188	187	194	210	199	180	156	117	99	74	23	9	75	156	170
205	205	206	207	207	205	203	201	193	193	193	192	190	187	184	183	189	189	196	189	188	195	190	193	163	139	115	127	152	143
210	210	210	210	210	209	209	209	202	203	204	204	204	203	201	201	191	190	189	188	189	192	194	196	198	198	189	173	161	157
211	212	212	212	210	206	203	201	201	202	203	203	203	202	201	200	203	201	199	197	194	192	190	189	189	196	199	196	193	195
216	217	217	216	212	207	201	198	201	201	202	202	201	201	200	199	202	202	202	202	201	199	197	196	181	187	192	192	191	192
215	215	214	213	209	205	201	199	200	200	200	200	200	200	199	199	197	198	200	202	203	205	206	206	201	201	197	191	185	182
206	205	203	201	200	199	199	199	201	200	200	199	199	199	199	199	202	201	199	199	199	201	202	204	210	209	206	203	199	195
202	201	199	198	198	199	201	202	201	201	200	199	199	199	200	200	201	200	198	196	197	198	201	202	202	202	206	210	210	206
203	202	202	201	201	202	203	203	202	201	200	199	199	200	200	201	197	197	197	197	199	200	202	203	205	203	205	210	211	207
201	202	202	202	202	200	199	198	203	202	201	200	199	200	201	202	201	201	201	201	201	199	198	197	203	197	194	196	196	194
202	202	202	202	202	202	202	202	203	202	202	202	202	202	203	204	205	204	203	201	200	198	197	197	198	198	198	198	198	198
202	202	202	202	202	202	202	202	203	202	202	202	202	202	203	203	202	202	202	201	200	200	199	199	199	199	199	199	199	199
203	203	203	203	203	203	203	203	203	202	202	201	201	202	202	203	200	200	201	201	201	201	202	202	200	200	200	200	200	200
203	203	203	203	203	203	203	203	203	202	202	201	201	201	202	202	201	201	201	201	201	202	202	202	202	201	201	202	202	202
203	203	203	203	203	203	203	203	203	202	202	201	200	201	201	201	203	203	202	202	201	201	201	200	202	202	202	202	202	204
204	204	204	204	204	204	204	204	203	202	202	201	200	200	200	200	205	204	203	203	201	201	200	199	202	202	203	203	204	204

Intensity of Each Pixel of Grey Scale Image (left)
(0 low intensity – dark … 255 high intensity – bright)

From slides by M. Mandala

Intensity scale of each pixel

- 1-bit monochrome: 0 for dark and 1 for bright.
- 8-bit color or greyscale: maximum number of colors/intensity per pixel is 256, stored in a 8-bit Byte.
- 16-bit color (high color): This allows 32,768 possible colors for each pixel.
- 24-bit color(true color): This allows 16,777,216 color variations. The human eye can discriminate up to ten million colors.
  - This is the most common format used today.
  - Most color images imported into MATLAB are at 24-bit color.

From slides by M. Mandala

How are color images stored?

Three Grey-Scale images each representing intensities of three primary colors: Red, Green, and Blue

Blue - Channel Intensity Grey scale image

Green - Channel Intensity Grey scale image

Red - Channel Intensity Grey scale image

The Emir of Bukhara, Alim Khan, in a 1911 color photograph by Sergey Prokudin-Gorsky

From slides by M. Mandala

For example:
The color image is a combination of the RGB images

Red - Channel Intensity Grey scale image

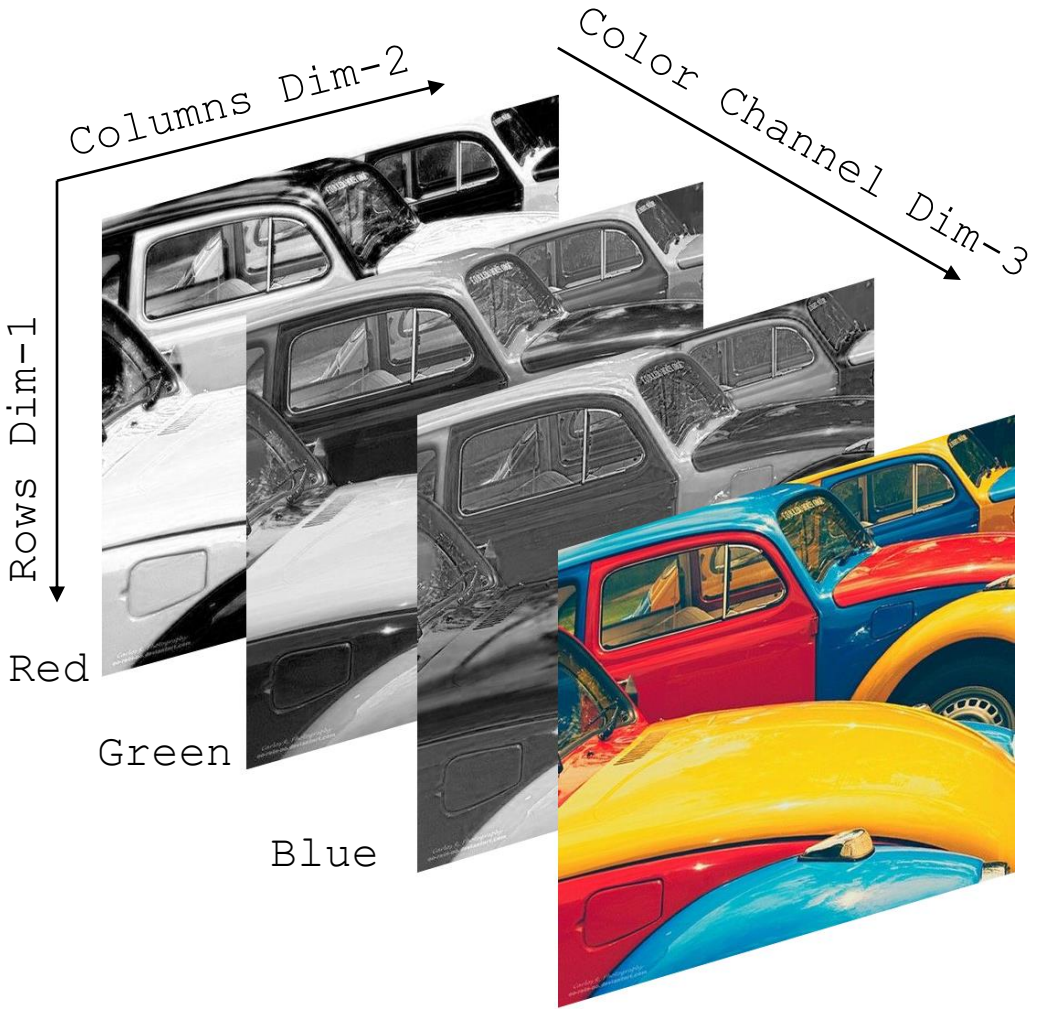
Note areas with red are brighter (closer to white) and areas with less red color are darker (closer to black)

Green - Channel Intensity Grey scale image

Blue - Channel Intensity Grey scale image

From slides by M. Mandala

Color images are stored as 3-dim array of numbers



Dim-1 = rows of pixels
Dim-2 = columns of pixels
Dim-3 = color channels

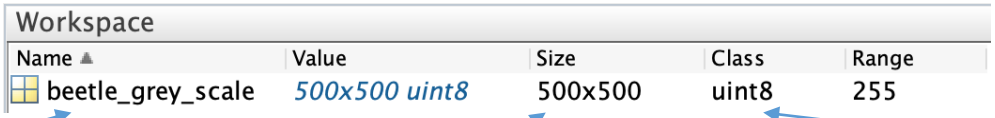
Most images contain 8-bit color channels, i.e. each channel represents 0-255 intensity levels of color. Therefore, a 3-channel color image contains 24-bits of color information.

From slides by M. Mandala

Examples using imread()

```
>> beetle_grey_scale = imread('WVBeetle.jpg');
```

Workspace:



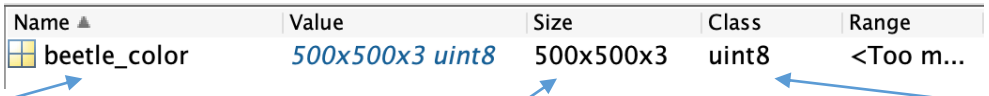
Variable Name

500x500 pixel
Row x Col
Single Channel

Data type:
Unsigned 8-bit Integer
Min: 0000 0000 = 0
Max: 1111 1111 = 255

```
>> beetle_color = imread('beetle.jpg');
```

Workspace:



Variable Name

500x500x3 pixel
Row x Col x channel
RGB Channel

Data type:
Unsigned 8-bit Integer
Min: 0000 0000 = 0
Max: 1111 1111 = 255

From slides by M. Mandala

The color image is a 3D numeric array

beetle_color is a 3D numeric array:

beetle_color(:,:,1) = Red Channel
beetle_color(:,:,2) = Green Channel
beetle_color(:,:,3) = Blue Channel

beetle_color(:,:,:) = Color Image

From slides by M. Mandala

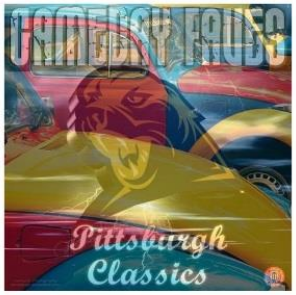
Since images are stored as numeric arrays, math operations can be performed on them

A = 

A+B = 

A-B= 

B= 

A*0.5+B*0.5= 

A.*B= 

To multiply, matrix dimensions must agree. You can multiple 1 channel at a time.

From slides by M. Mandala

Try manipulating one of your images, to change from darker to brighter

- Remember: 0 is low intensity – dark, and 255 is high intensity – bright

Images can be manipulated as follows:

- To blend images:
  - Multiply images (as long as dimensions agree, one channel at a time) or element-by-element, all channels at once
  - Add (element-by-element, all channels at once)
  - Subtract (element-by-element, all channels at once)
  - Multiply by scalar (element-by-element, all channels at once)

- To manipulate shape/size:
  - Concatenate: horzcat, vertcat
  - Extract elements

- Resulting values above 255 are clipped at 255 (since data type can only store values up to 255)
  - Similarly, values below 0 are clipped at 0

From slides by M. Mandala

Let’s try an example!

- Use the provided image and ask the user if they would like to make it darker or lighter

- Change image as indicated by the user, and display new image

Let's try an example!

```
clear
clc

%Get file name
filename=input('Please enter the file name: ','s');

%Load image
my_img=imread(filename);

%Get user choice
choice=input('Select your option: 1-make image darker or 2-make it lighter');

%Modify image
%Get size of image
[rows,cols]=size(my_img);
switch choice
    case 1
        new_img=my_img*0.5;
    case 2
        new_img=my_img*5;
end

imshow(new_img)
```

Remember the 's' – it is a string input!

Modular programming can be very helpful

- You don’t have to type in the same commands every time you create a script (for example, to load a file)
- Create many m files, then “call” the one you need
- (Similar to using commands in your script – these are m files)

This…

load_my_data

Commands to enter the file name, load the data, and extract the variables

solve_Axb

Commands to solve system of equations

graph_and_label

Commands to graph and place labels on graph

…becomes this!

load_my_data
solve_Axb
graph_and_label

Instead of one long program, we now have a short main program, with several subprograms

Subprograms

Commands to enter the file name, load the data, and extract the variables

Commands to solve system of equations

Commands to graph and place labels on graph

load_my_data
solve_Axb
graph_and_label

Main program

In MATLAB, you can do this in two different ways (see your textbook)

- The preferred way is using functions

Let’s try this!

- Take the program we just created, and split it up into functions

Let's try this!

Main program

```
clear
clc

%Call img_header function
img_header;

%Get the name of the image file
filename=img_file();

%Load image
my_img=img_load(filename);
```

No variables coming into or out of the function

```
%Function to display a header
function img_header
disp('This program will manipulate images!')
```

No variables coming into the function; one going out

```
%Get file name
function filename=img_file()

filename='0';

while exist(filename)==0
    filename=input('Please enter the file name: ','s');
end
```

One variable coming into the function; one going out

```
%Load image
function my_img=img_load(filename)


my_img=imread(filename);
```

Let's try this!

Main program

```
clear
clc

%Call img_header function
img_header;

%Get the name of the image file
filename=img_file();

%Load image
my_img=img_load(filename);

%Get user choice
choice=input('Select your option: 1-make image darker, or 2-make it lighter');

%Modify image
new_img=img_modify(choice, my_img);

%Display image
img_display(new_img)
```

Two variables coming into the function; one going out

```
function new_img=img_modify(choice, my_img)

switch choice
    case 1
        new_img=my_img*0.5;
    case 2
        new_img=my_img*5;
end
```

One variable coming into the function; none going out

```
function img_display(new_img)

imshow(new_img)
```

With functions, we don't need to use the same variable names

- To create a function, include this as the first line in the script you want to call and be able to use over and over:

function[list of variables]=function_name(list of variables)

Variables being passed from the subprogram to the main program – use [] and separate with commas

Variables being passed from the main program to the subprogram – use () and separate with commas

Let's try this!

Note that you have to start with function

- Create m-file with the following function:

```
function [a, b] = first(x,y)
a = x+y;
b = x-y;
```

Note that this m-file must be saved first.m

- Create the following main program, main.m:

```
clear
x1=4;
y1=3;
a = 20;
b = 25;
[a, b]=first(x1,y1)
```

Note the use of the function name

- Now run the main program

Let's try this!

- Create .m file with the following function:

```
function [a, b] = first(x,y)
a = x+y;
b = x-y;
```

What are the values of a and b after running the program?

a=7 and b=1

- Create the following main program, main.m:

```
clear
x1=4;
y1=3;
a = 20;
b = 25;
[a, b]=first(x1,y1)
```

Note that we provide the values to be used as x and y in the subprogram

- Now run the main program

Note that we get the values of a and b from the subprogram → can suppress output with ;

Now try this one! First solve by hand, then see if you get the same results when you run it

```
function [a, b, c] = simple(x,y,z)
    a = x+y+z;
    b = (x*y)/z;
    c = (y*z)/x;
```

Create the following main program, main.m:

```
clear
a=1;
b=1;
c=1;
x1=3;
x2=5;
y1=6;
y2=2;
z1=1;
z2=8;
[a1, b1, c1]=simple(x1,y1,z1)
[a, b, c]=simple(x2,y2,z2)
```

From slides by N. Vidic

Regarding function names, also remember:

- They should not be the same as any of the built-in MATLAB commands
- There should not be any spaces in the function names

Our script can be modified to run with different data files

- Let user enter the data file name so that the program doesn't need to know it and the user doesn't need to know the variable name for the data:

```
filename=input('Enter file name: ', 's');
data = load(filename);
```

's' required because it is a string input!

- This will create a variable called filename that is a string variable and a variable called "data" that contains the data

Let’s try this!

Create data file:
1 2 3 4 5 6 7
5 10 15 20 25 30 35

Let's try this!

Note that this function doesn't require any variables passed from the main to it, but it returns x and y

simple.m | mainSimpleEx.m | load_data_Example.m | mainload_data_Example.m

```
1  function[x,y]=load_data_Example
2
3  %Ask user for file
4  filename=input('Please enter the name of your data file:  ','s');
5  data=load(filename);
6  [rows,cols]=size(data);
7
8  %Assign variables based on whether they are in rows or columns
9  if rows==2
10     x=data(1,:);
11     y=data(2,:);
12 elseif cols==2
13     x=data(:,1);
14     y=data(:,2);
15 else
16     disp('Error in data file')
17 end
18
```

So, regarding functions:

- This is how the built-in MATLAB functions work
- We are now creating our own functions to do the computations or data manipulations that we do frequently

Practice problem 1

- Write a MATLAB program that will compute a monthly loan payment (A). The main program should display the purpose of the program. It will then call two functions (so you will have three m files).
- ***user_inputs*** should not receive any variables but should return the user-entered values for a principle amount of money to be borrowed (P), the number of months to pay the loan back (N), and a monthly interest rate (i)
  - Make sure the user is able to enter the interest value as a percent and have the program divide it by 100 for the calculation
- ***calculate_A*** should receive the three values entered above and return the payment (A) to the main program.
- The main program should then display the payment (A).
- Put everything in a while loop so the user can try several payment plans.

Try your program with the following values:

P=$3000 N=30 i=1%

P=$100,000 N=360 i=.5%

P=$20,000 N=48 i=.2%

Answers you should get are:

116.24; 599.55; 437.40

$$A = P\left[\frac{i(1+i)^N}{(1+i)^N - 1}\right]$$

Submit .m file called “Mena_Time_MyFunctions1Team#” (“Mena_10am_MyFunctions1L01”) into Classwork folder

From slides by N. Vidic

Practice problem 2: Calculate future value

- The main program should provide information about the purpose of this program
- **Get_inputs** should not receive any variables but should return the user-entered values for P, N, i, and A
  - Make sure the user is able to enter the interest value (i) as a percent and have the program divide it by 100 for the calculation
- **Future_withP** should convert to a future value (F_1) given a present single payment (P) → $F\_1 = P\,[(1 + i)^n]$
- **Future_withA** should convert to a future value (F_2) given an annuity (A) per interest period → $F\_2 = A\,[(\,(1 + i)^n - 1\,) / i\,]$
- The main program should display "The calculated value for F_1 is ## and for F_2 is ##"
- *Practice using different variable names within each function!*

Submit .m file called "Mena_Time_MyFunctions2Team#" ("Mena_10am_MyFunctions2L01") into Classwork folder

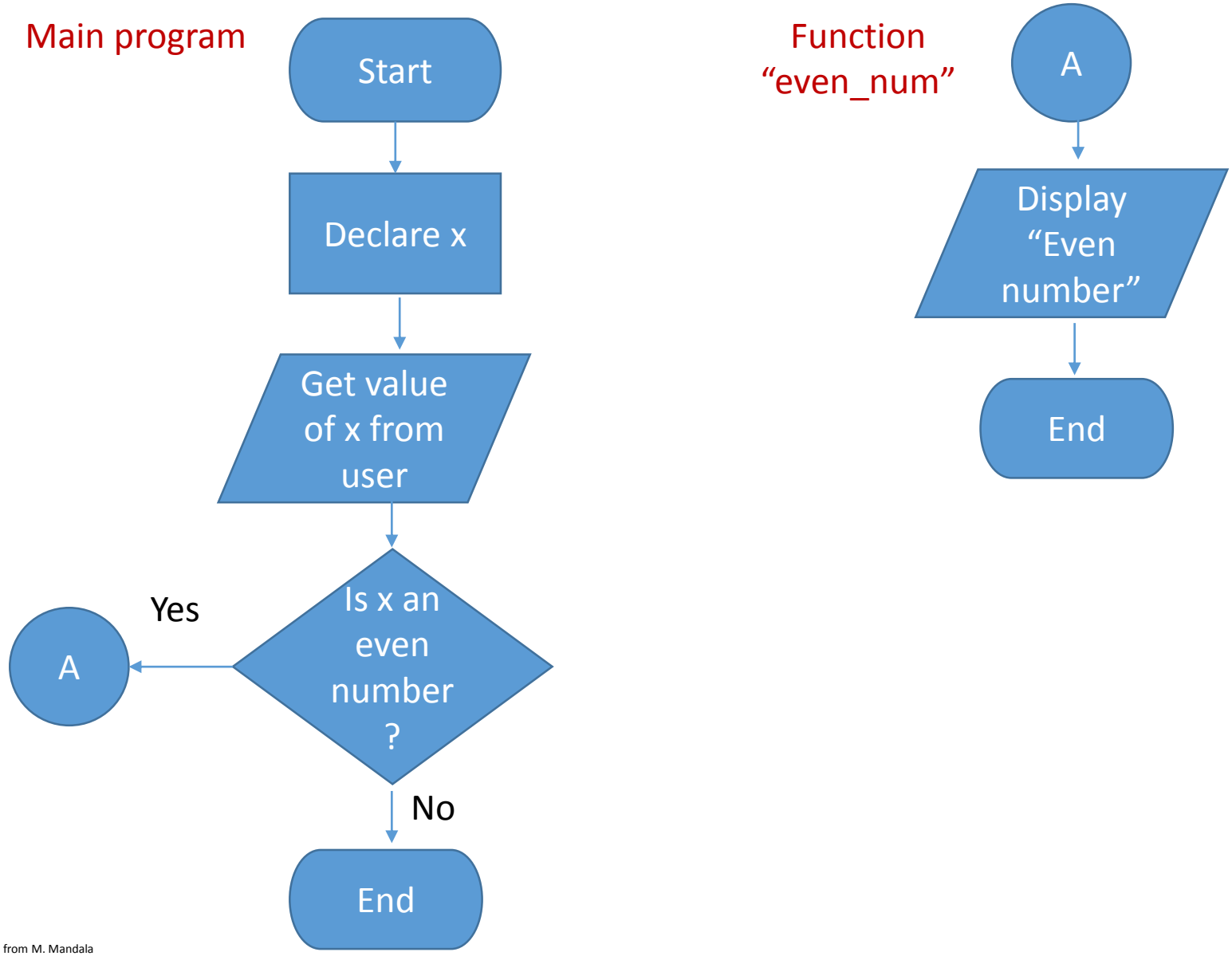
Try P=3000, i=2, n=30, A=1200
you get
F_1=5434.08 and F_2=48681.70

Functions can be represented in flowcharts

- Create a program that will ask the user to provide a number
- If the number is even, the program will call function “even_num”, which will display a message to the user saying it is an even number
- If the number is odd, the program ends

Example from M. Mandala

Use a circle to represent functions in a flowchart

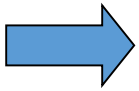


Example from M. Mandala

Quick review

- How would you represent this as a system of linear equations?

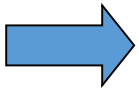
$$3x + 5z = 2y$$
$$z = 2y - 8$$
$$4(x + y) = 2(0.5 - z)$$



$$3x - 2y + 5z = 0$$
$$0x + 2y - z = 8$$
$$4x + 4y + 2z = 1$$

You can represent these in matrix form

$$3x + 5z = 2y$$
$$z = 2y - 8$$
$$4(x + y) = 2(0.5 - z)$$



$$3x - 2y + 5z = 0$$
$$0x + 2y - z = 8$$
$$4x + 4y + 2z = 1$$

$$A = \begin{bmatrix} 3 & -2 & 5 \\ 0 & 2 & -1 \\ 4 & 4 & 2 \end{bmatrix} \quad x = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 8 \\ 1 \end{bmatrix}$$

You can solve a linear system of equations in MATLAB – we solve Ax=b

$$A = \begin{bmatrix} 3 & -2 & 5 \\ 0 & 2 & -1 \\ 4 & 4 & 2 \end{bmatrix} \quad x = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 8 \\ 1 \end{bmatrix}$$

- So how do you solve for x?

Usually:
Ax=b → x=b/A

But matrix division is not possible!

From slides by N. Vidic

In Ax=b, how do you solve for x?

- The concept of identity matrix is helpful here → eye(n)
- $A^{-1}*A=I$ → A^{-1} is the inverse, or inv(A)

$$A * x = b$$

$$A^{-1} * A * x = A^{-1} * b$$

$$I * x = A^{-1} * b$$

$$x = A^{-1} * b$$

Using MATLAB to solve Ax=b

```
1  clear
2  clc
3
4  %Load data
5  load new_data.txt
6
7  %Display data
8  new_data;
9
10 %Extract matrix A and b
11 %A=new_data(:,1:3);
12 %b=new_data(:,4);
13
14 %Sometimes we don't know the matriz dimensions
15 Matrix_dim=size(new_data);
16 NumRows=Matrix_dim(1);
17 NumCols=Matrix_dim(2);
18
19 %Extract A and b
20 A=new_data(:, 1:NumRows);
21 b=new_data(:, NumCols);
22
23 %Now solve for x
24 x=inv(A)*b;
25
```

To transpose a matrix, use ‘

- Or transpose

- This is useful when you need something stored in rows but it is actually stored in columns, or vice-versa

Practice Problem

Find the 5 numbers that satisfy the following:

- The sum of all the digits is 18
- The third digit is the sum of the first and second digits
- Subtracting the third digit from the fourth yields 4
- The 1st and 3rd digits added together give the fifth digit
- The first digit is twice the second digit

Create a data file ("MatData") with the data, and create an m-file called that loads the data and solves this problem

Name your file Mena_time_SolveTeam#

(for example: "Mena_10am_SolveTeamL01"),

and submit to Classwork folder

From slides by N. Vidic

How can we check for errors in the data?

- We know A should be a square matrix
  - num_rows==num_cols-1
- We know equations should be independent of each other
  - det(A)~=0 → you have independent equations
- How would you include this in the m-file we just worked on?

Editor - C:\Users\imena\Documents\MATLAB\solveAxb.m*

solveAxb.m*

```
clear
clc

%Load data
load MatData.txt

%Get matrix dimensions
MatDim=size(MatData);

%Get num rows and num columns
numRows=MatDim(1);
numCols=MatDim(2);

%Check for square matrix
if numRows==numCols-1
    %Extract data into A and b
    A=MatData(:,1:numRows);
    b=MatData(:,numCols);
    %Check for independence of equations (det~=0)
    if det(A)~=0
        %Solve for x
        x=inv(A)*b
    else
        disp('Error in file!')
    end
else
    disp('Error in file!')
end
```

MATLAB has some options for generating random numbers

- rand, rand(n), rand(m,n)
- randi(MaxNumber), randi(MaxNumber,n), randi(MaxNumber,m,n)
- The sequence is determined by the state of the generator
  - rand('state',sum(100*clock)) resets the generator to a different state each time
  - Change the "state" before using it, otherwise the same sequence of numbers will be generated each time you run a program
  - Can also use rng('shuffle')

From slides by N. Vidic

Sometimes you need values NOT between 0 and 1

You can:

- Use R=randi(imax) %Integer between 1 and imax
- Multiply by a value
- Add a value for a different starting point
- For random numbers between (a,b): (b-a)*rand+a

- Use round, floor, ceil to get values in the format you would like

We can use MATLAB for games: Let's simulate a coin flip!

```
heads=0;
tails=0;

for i=1:1000000
        x=rand;
        if x<0.5
        heads=heads+1;
        else
                tails=tails+1;
        end
end
disp(heads)
disp(tails)
```

Creating scripts that count for us

- How many negative numbers are in array x=[-1 6 -5 2 -9 5 2 -3 9]?
- How would the script change if x were a matrix instead of an array?

Counting in an Array

```
%Data
x=[-1 6 -5 2 -9 5 2 -3 9];

%Set running sum to 0
count=0;

%Count how many negative numbers appear in x
for i=1:length(x)
    if x(i)<0
        count=count+1;
    end
end

count
```

Counting in a Matrix

```
%Data
x=[-1 6 -5; 2 -9 5; 2 -3 9];

%Get dimensions of x
[rows, cols]=size(x);

%Set running sum to 0
count=0;

%Count how many negative numbers appear in x
for i=1:rows
    for j=1:cols
        if x(i,j)<0
        count=count+1;
        end
    end
end

count
```