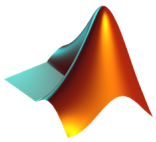


Getting Started with MATLAB

This Live Script introduces some basic MATLAB commands and provides practice with displaying images and writing a new script.

Executable code appears in boxes that have a gray background. This Live Script is divided into sections separated by thin horizontal lines. The code within each section can be executed by placing your cursor somewhere in the section and clicking the **Run Section** icon in the LIVE EDITOR tab at the top of the MATLAB desktop.

With your partner(s), work through the content of this Live Script at your own pace. Run the code in each section and carefully observe the results. Wherever you see questions or exercises, add comments or code directly to this file, and test your code by running the section again. Fix errors as you go, or comment out statements that generate errors.

You can save this file by clicking on the **Save** icon near the left side of the toolbar in the LIVE EDITOR tab. The MATLAB printout and your added comments and code will all be included in the saved file. If you want to remove the MATLAB printout from this file, click on the **Clear all Output** icon in the VIEW tab at the top of the MATLAB desktop.

Most of the tasks here can be completed within the Live Script. There are instructions near the end for displaying an image by entering commands directly in the MATLAB Command Window, and for creating a separate script file.

Enjoy your first taste of MATLAB!

Create variables

```
num1 = 10                    % create a variable and assign a value

vect1 = [1 5 -3 7 0]         % create a vector of values (1D matrix)

num1                         % print value of an existing variable

num2 = 20.0;                 % semicolon suppresses printout

whos                      % list current variables in the Workspace - note that by default,
                          % numbers are stored as "double" type (floating point)
```

Perform simple calculations

```
result = (2 * num1) + num2      % perform simple calculations, using existing variables

dist = sqrt(num1^2 + num2^2)

area = pi * num1^2              % pi is a built-in constant

bignum = num1/0                 % note: no error! Inf is also a built-in constant

abs(result - round(area))       % with no assignment, value is assigned to generic variable ans

vect1                           % vector of numbers created earlier

vect2 = 5 * (4 + vect1)         % perform calculations on entire vector all at once

vect3 = abs(vect2 - (10 * vect1))

% exercise: suppose vect1 represents a set of temperatures in degrees Celsius - write
% one statement to convert these temperatures to degrees Fahrenheit, F = (9/5)C + 32
```

Create 2D matrices

```
mx1 = [3 8 -4; -1 0 2]          % create a 2x3 matrix - the semicolon separates rows

mx2 = (mx1 * 2) + 10            % perform calculations on entire matrix all at once

mx0 = zeros(3,2)                % create a 3x2 matrix containing all 0's

vect0 = zeros(1,10)             % to create a vector of 0's, need to specify two dimensions

% exercise: what happens if the number of elements listed for each row of mx1 is not the same?

% exercise: create a 4x2 matrix of numbers
```

Get dimensions of vectors and matrices

```
size(mx1)                   % get dimensions of a matrix or vector (returns a 1x2 vector)

size(vect1)

size(mx1, 1)                % question: what does the second (optional) input do?

size(mx1, 2)

dims = size(mx1)            % store dimensions in one or two variables

[rows, cols] = size(mx1)

length(mx1)                 % length returns the largest dimension

mx1'                        % question: what does the ' operator do?
```

Perform element-by-element computations

```
mx1 = [2 4 0; 6 1 3]    % create two new matrices
mx2 = [0 5 1; 4 2 0]

% the operations below are applied to corresponding elements of mx1 and mx2,
% element-by-element - content of multiple vectors can be combined in the same way

mx1 + mx2

mx1 - mx2

mx1 * mx2               % oops! to multiply (or divide) the contents of two matrices
                        % element-by-element, use the .* operator instead
mx1 .* mx2

mx1 ./ mx2

squares = mx1 .^ 2

% exercise: what happens if mx1 and mx2 have different dimensions?

% exercise: compute the element-by-element average of mx1 and mx2
```

Use indices to access the content of vectors and matrices

```
% recall the vector and matrix we created earlier
vect1
mx1

vect1(3)            % use one index to refer to elements of a 1D vector

vect1(0)            % question: how do we refer to the first element of a vector,
                    % and how does this differ from Python and Java?

vect1(end)          % keyword "end" refers to the last index

vect1(4) = 20;      % change content with an assignment statement

mx1(1,2)            % use two indices to refer to elements of a 2D matrix

mx1(1,end)          % question: what does "end" refer to in these two statements?
mx1(end,1)

mx(2,3) = 30;       % change content of an individual matrix location
```

Use colon notation to generate sequences of numbers or indices

```
seq1 = 4:9          % consecutive integers from 4 to 9

seq2 = 1:2:10       % sequence of odd integers

% exercise: add a brief comment about what each statement does, given the results

seq3 = 2:4:24

seq4 = 27:-3:4

seq5 = 10:1:3

vect1 = [1 4 2 6 3]

vect1(2:4)

vect1(3:end) = 20

mx1 = [1 2 3 4; 5 6 7 8; 9 10 11 12]

mx1(2,2:4)

mx1(end,3:end)

mx1(2,:)

mx1(:,4)

patch = mx1(:,1:3)

mx1(2:3,2:4) = 20
```

Handy matrix operations

```
% recall the vector and matrix we created earlier
vect1
mx1
```

```
sum(vect1)          % add up all the contents of the vector

sum(mx1)            % question: what does this do?

% exercise: write one statement using sum, to add up all the contents of mx1
% (returning a single total number)

min(vect1)          % returns the minimum value in a vector

max(vect1)          % returns the maximum value in a vector

% exercise: write one statement that returns the minimum value of all of mx1

% exercise: write one statement that returns the maximum value of all of mx1
```

Help!

```
help cos                        % access the help facility ...

doc magic                       % ... or documentation pages

% exercise: use randi (check out its documentation!) to write a statement
% to create a 2x4 matrix of random integers between 1 and 10
```

Read and display an image

An image can be read from a file into the MATLAB workspace using **imread**, and can be viewed with the functions **imshow** and **imtool**.

To begin, execute the following two statements directly in the Command Window (you can just cut-and-paste):

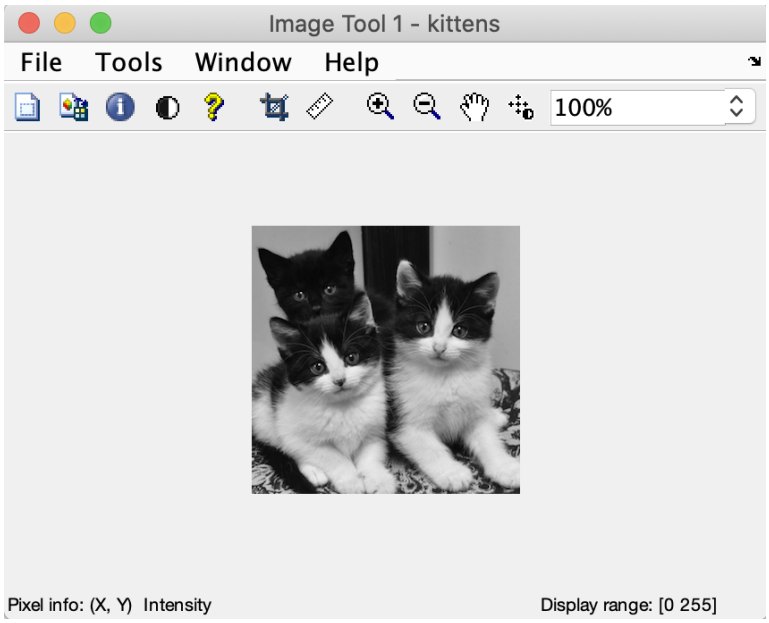
```
kittens = imread('kittens.png');

imshow(kittens)
```

A separate figure window will open up, displaying the image. You can enlarge the window by dragging the lower right corner.

The image can be examined in more detail with the "Image Tool" - execute the following statement directly in the Command Window:

```
imtool(kittens)
```

Again, drag the lower right corner to expand the window size so that you can see all the icons in the toolbar at the top of the window and the printout along the bottom of the window, as shown here:



To explore the image with the Image Tool, try the following:

- **move the cursor around the image** and note the changing **Pixel info:** in the bottom left corner, which indicates the (X,Y) coordinates of the image location and the intensity value stored at this location (an integer 0 to 255). *How are the X and Y coordinates defined?*
- **zoom in** to view part of the image in more detail - click on the "+" magnifying glass icon at the top of the Image Tool window, move the cursor to a central area of the image, and click - note the change in the appearance of the image (click the "+" magnifying glass icon again to turn off the zoom-in function)
- **zoom out** by clicking in the "-" magnifying glass icon and then clicking on a central region of the image
- use the **Pixel Region** tool to view the content of the image matrix in a separate window, by clicking on the second icon at the top of the Image Tool window. A separate window will pop up showing a grid of intensity values - this window can be expanded by dragging any corner. You can **move the subwindow around** to different locations in the kittens image using the scroll bars along the right and bottom edges of the Pixel Region window or by dragging the cursor icon around the original Image Tool window. You can also **zoom in and out** of the grid of intensities by clicking on the icons in the toolbar at the top of the Pixel Region window.

Clean up the workspace and windows

The **clear** command can be used to delete individual variables from the workspace, or clear all of the existing variables, for example:

```
clear vect1 vect2
```

```
clear all
```

Individual figure windows can be closed by clicking on the red 'x' in the upper left corner, or you can close all of the figure windows (created with imshow) or Image Tool windows (created with imtool) at once:

```
close all
```

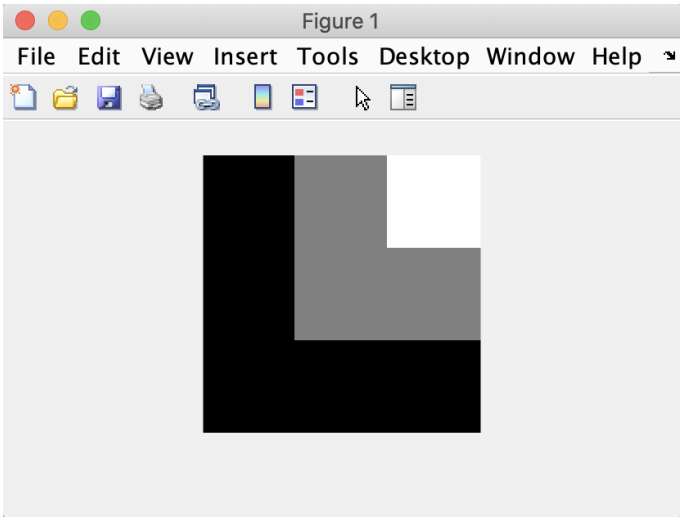
```
imtool close all
```

Create a new script

Some MATLAB code files are referred to as M-Files and have a .m file extension. In this final section, you will create a type of M-file called a *script*, which contains a sequence of MATLAB commands that can be executed all at once. Executing a script produces the same results that would be obtained if the individual commands were entered, one-by-one, in the Command Window.

From the **New** menu near the left end of the LIVE EDITOR or HOME tab at the top of the MATLAB desktop, select **Script**. A blank file will appear in a new tab in the editor.

Your task is to add code to this script to create and display the image shown in the figure window below:



A file can be saved by clicking on the **Save** icon in the toolbar that appears in the EDITOR tab at the top of the MATLAB desktop. To execute the script, click on the **Run** icon in this toolbar, or enter the first name of the file in the Command Window.

To create the above image:

- use the **zeros** function to create a 300x300 matrix of 0's
- use what you learned in the section titled, "Use colon notation to generate sequences of numbers or indices" to set an upper right region of the image matrix to an intermediate shade of gray (e.g. 128) and then set a smaller region to white (255)
- **when calling imshow, provide empty brackets as a second input so that MATLAB displays the full range of intensities from 0 to 255, e.g. imshow(myImage, [])**