MATLAB: A Comprehensive Introduction

October 2018

Catherine Paverd

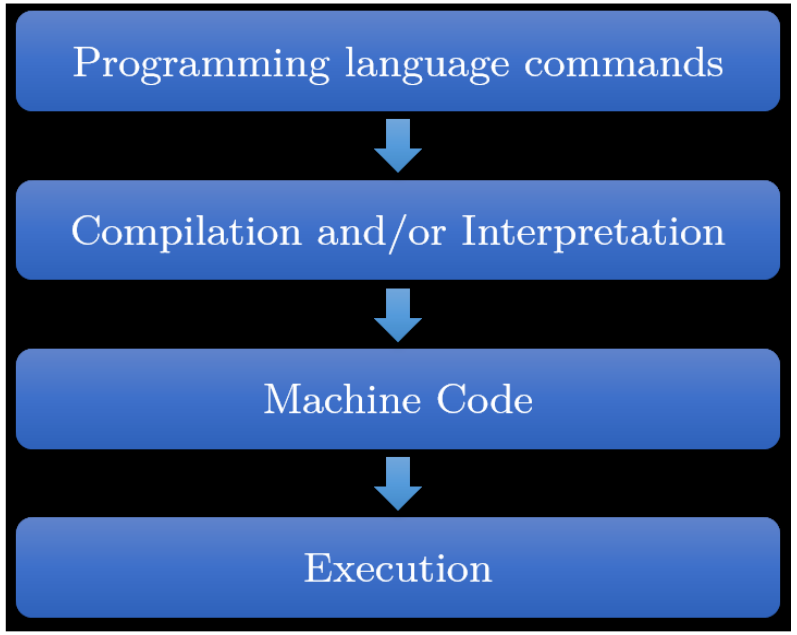
catherine.paverd@eng.ox.ac.uk

Introduction

- This course is an introduction on how to use MATLAB, with a specific focus on data analysis
- The course will consist of 4 sessions, 3 hours each
  - **Part 1: Getting Started** – Fundamentals, Data Types and Matrices
  - **Part 2: Programming Basics –** Operators and Control, Programming, Error Handling
  - **Part 3: Working with Graphics** –Figures, Axes, Graphs, Objects and Images
  - **Part 4: Further Programming** – File Handling and Performance
- Administration
  - Notes are available in a PDF
  - MATLAB software is available for all university departments and colleges: http://www.eng.ox.ac.uk/~labejp/TAH/matlabTAH.html

Fundamentals

- At the most basic level, computers are made up of 1s and 0s, but this is too tedious to write out as a programmer.
  - ‘Welcome!’ in binary would look like this: ‘`01010111 01100101 01101100 01100011 01101111 01101101 01100101 00100001’
  - So we invented programming languages that allow us to write in English at a higher level, and then use compilers or interpreters to translate what we say into something a computer understands



- High level languages are easier to write but take longer to translate (e.g. MATLAB), while mid level languages (e.g. C) take longer to write but can execute faster

Fundamentals

- MATLAB = MATrix LABoratory
- High level programming language used for numerical computations, data analysis, algorithm development, data visualisation
- MATLAB consists of various parts:
  - The Language
  - Desktop tools and development environment
  - Function libraries
  - Graphics Libraries
  - External Interfaces libraryti

The Language

- Control flow statements
- Functions
- Data structures
- Input/output functionality

Desktop Tools and Development Environment

- Command Window
- Editor/Workspace etc.
- Debugger

Mathematical Function Library

- Collection of computational algorithms (e.g. sine, cosine, but also matrix inversion, FFT, etc)

Graphic Functionality

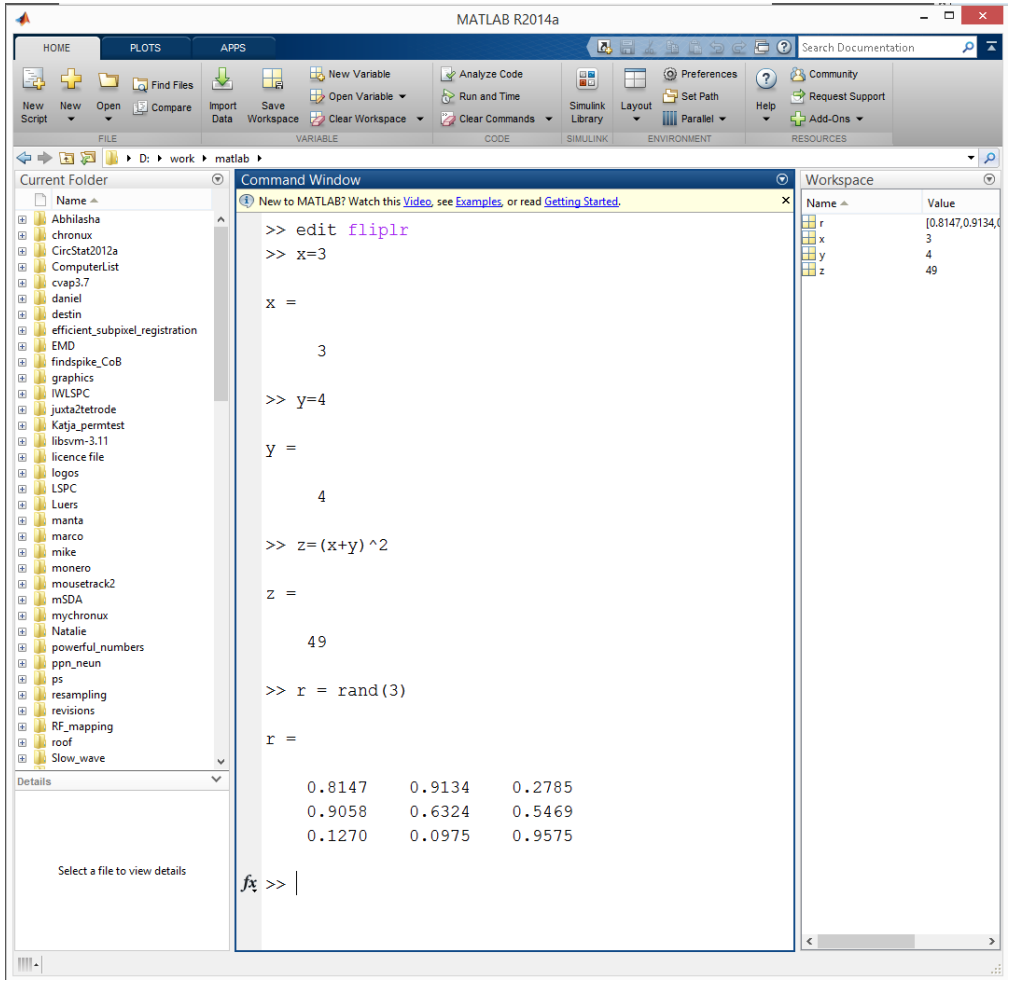
- Vector and matrix display
- 2D and 3D data visualization
- Image processing
- GUI design

External Interfaces

- C/ Fortran interaction with Matlab
- Dynamic linking of routines with Matlab.

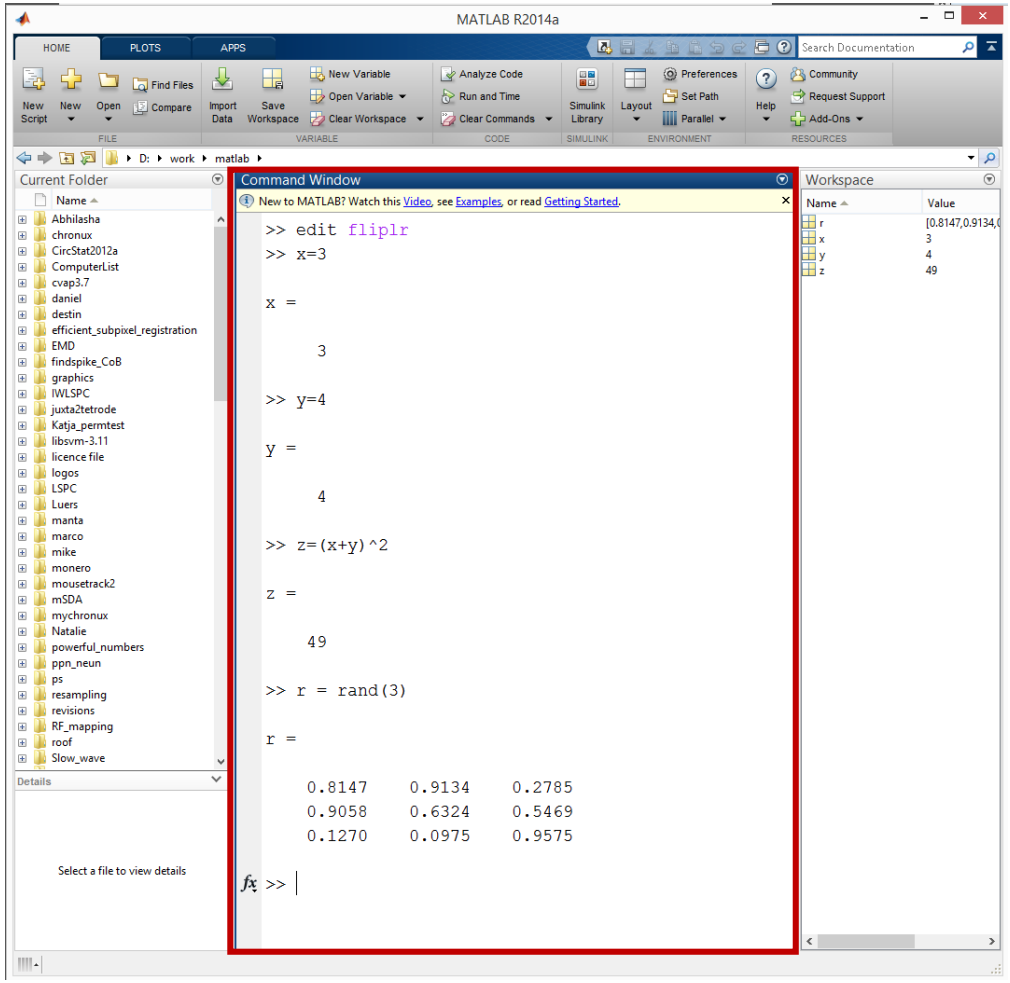
The MATLAB Desktop

- Command Window (centre)
- Current Folder information (left)
- Workspace (right)
- *Note: Some installations of MATLAB may be setup slightly differently on first opening depending on the version installed.*



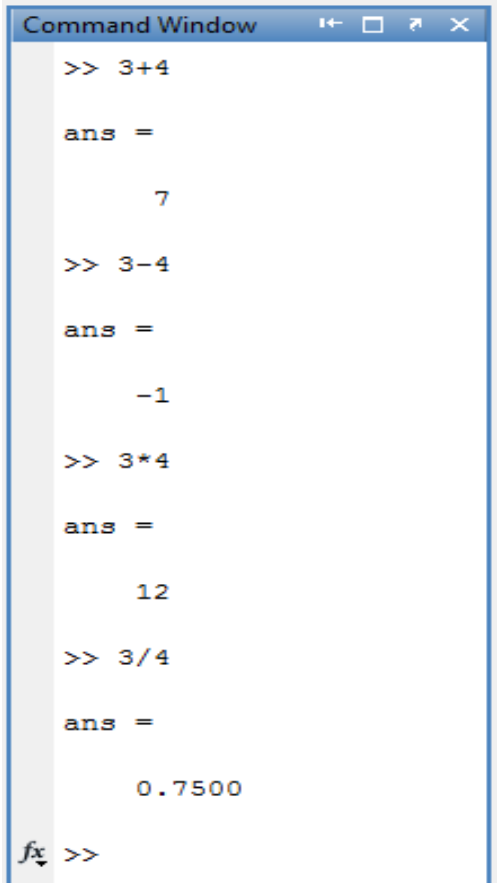
The MATLAB Command Window

- Used to create variables, enter commands and run programs interactively
- If you type a command here and press ENTER, the command will execute immediately (in most cases), and the output (if there is one) will display in the same window
- The command window is generally used for quick calculations – anything more substantial should go in a separate file



The MATLAB Command Window

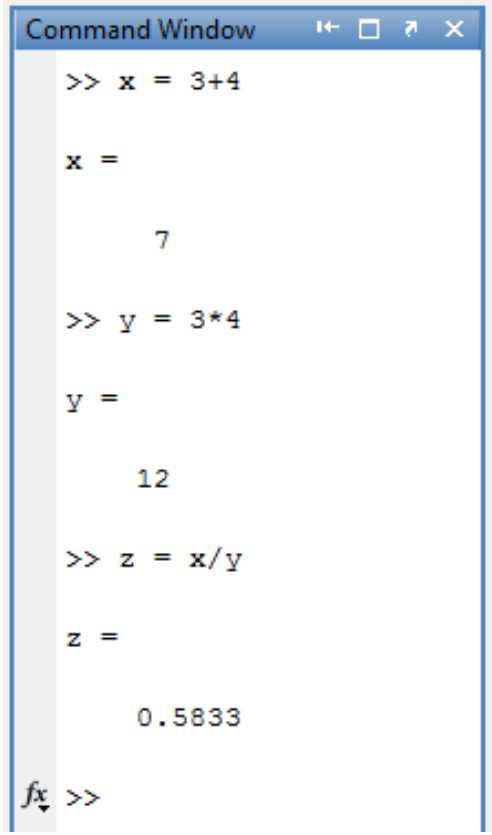
- Example:
  - You can use basic arithmetic operators (+, -, *, / etc.) to do simple calculations.
  - The output is shown in a variable called *ans*. This is a generic variable and will be overwritten every time you do a new calculation
  - One way to stop it being overwritten is to use variables



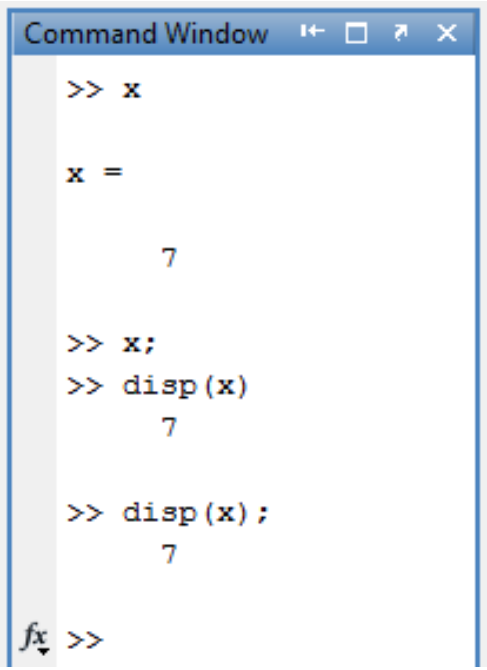
The MATLAB Command Window

- Example:
  - Here is the same example using variables
  - The variables are stored in the Workspace (top right) and can be accessed at any time
  - You will need to give each variable a different name in order to avoid overwriting data

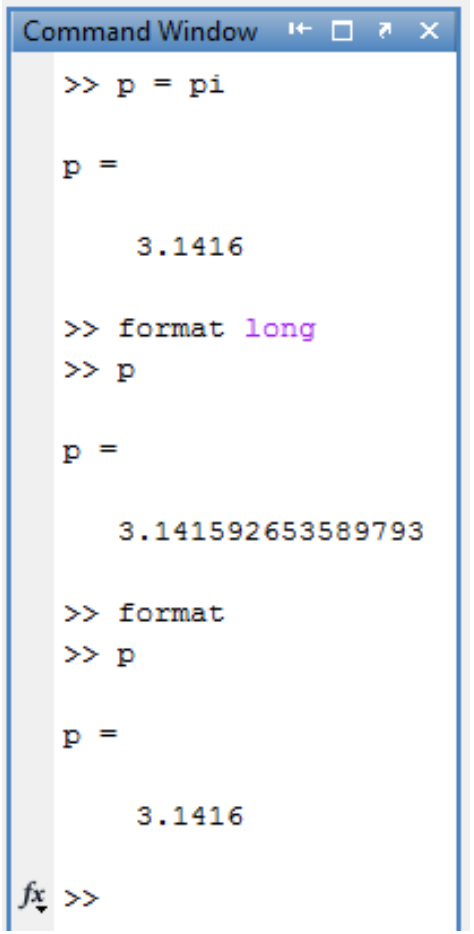
- *Note: To see what is stored in a variable, double click on the variable in the workspace window, or type the variable in the command window. To see all variable at the same time, type 'whos' in the command window.*



The MATLAB Command Window

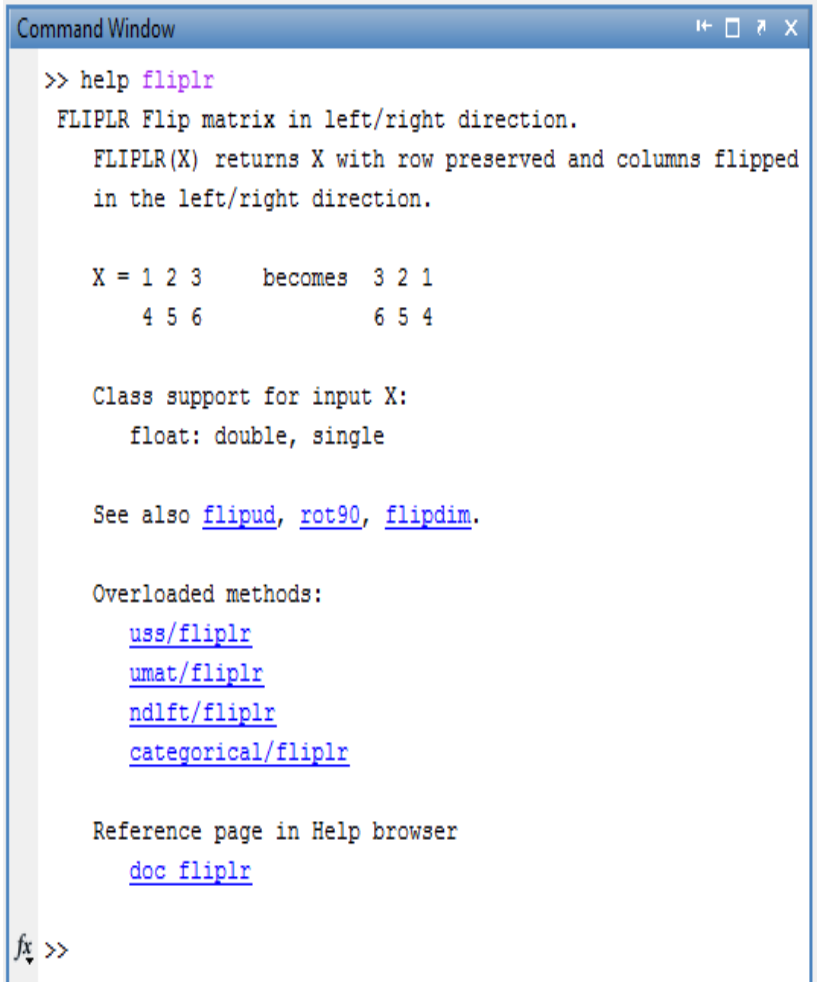


- There are several ways to control the display of variables' values in the command window, as shown on the left.
- The *format* command can change the display output of numerical values – note that this does not affect the calculations. An example is shown on the right.
- Note that a useful command is **clc,** which clears the command window.



Getting help in MATLAB

- There are two ways of getting help in MATLAB:
  - By using the Help icon  at the top right part of the MATLAB Desktop. It provides detailed documentation of MATLAB functionality.
  - By typing in the Command Window **help** or **doc** followed by the command or operator name.



Understanding File Locations

matlabroot

- Where MATLAB is installed

Current folder

- Where MATLAB looks for files
- We can view and change it in two ways:
  - Using in the command window the **cd** or **pwd** commands
  - On the MATLAB Desktop using the tools provided.

Startup folder

- Current folder at startup
- **userpath** command shows the default startup folder.

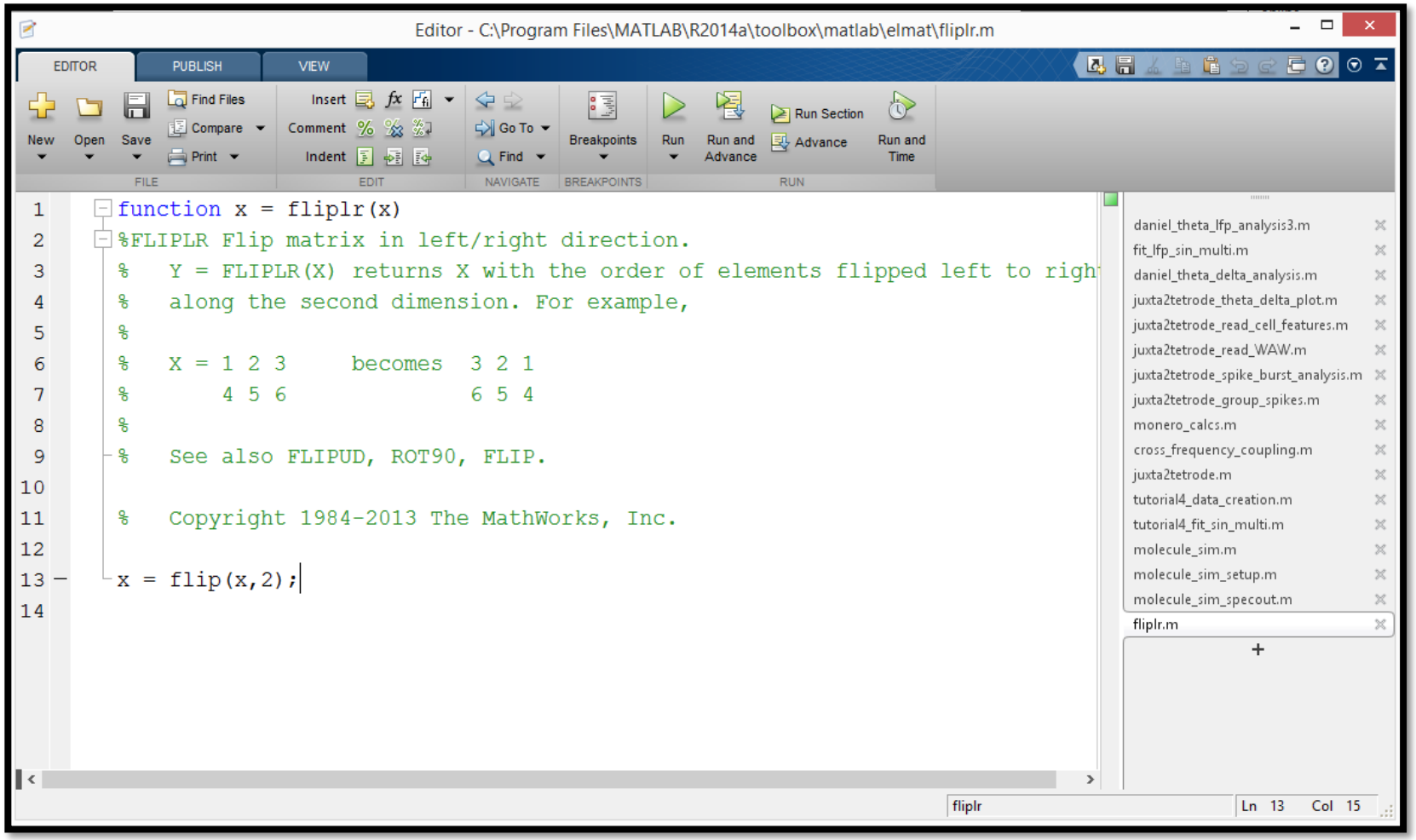
The MATLAB Path

- For performance reasons, MATLAB limits where it looks for files.
- Files are only accessible if they are in either the current folder, or the search path.
- The path is a list of folders where MATLAB will search for program and data files.
- To see the path, type ***path*** in the command window.

The MATLAB Editor

- The MATLAB editor is used to view and edit MATLAB program files. You can open it by typing ***edit*** and pressing ENTER in the command window.
- MATLAB programs are text files with **.m** extensions and they are called M-files.
- The MATLAB editor opens automatically when you open or create a new M-file.
- It provides code highlighting and debugging features, and allows you to save blocks of code so that you do not need to type out every line each time you write a program.

The MATLAB Editor



MATLAB Scripts

- They are the simplest kind of MATLAB programs, just containing a list of commands.
- When a script runs, the result is the same as if the commands were entered one-by-one in the command window.
- To open a new blank script:
  - Use the key combination **Ctrl+N**, or
  - Click the **New** button in the editor, then choose the **Script, or**
  - Click the **New Script** option on the desktop Home tab.
- To open an existing file:
  - Use the key combination **Ctrl+O,** or
  - Click the Open button in the editor or desktop Home tab, or
  - Use the **edit** or **open** commands in the Command Window.

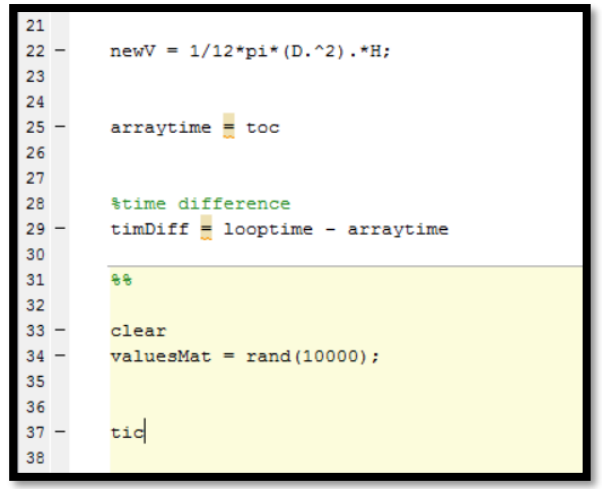
MATLAB Scripts

- There are two ways to run a script:
  - From the Command Window, by typing the script name and pressing ENTER.
  - From the Editor, by pressing F5 or by clicking the Run button.

- To run only a section of the script, we select this section and press F9.

- By typing two % signs you can create a new section (the editor adds a thin black line to separate sections), or use the insert section button: 

  - To run a section make sure you are in the section (it will be highlighted yellow) and press **Ctrl+ENTER**

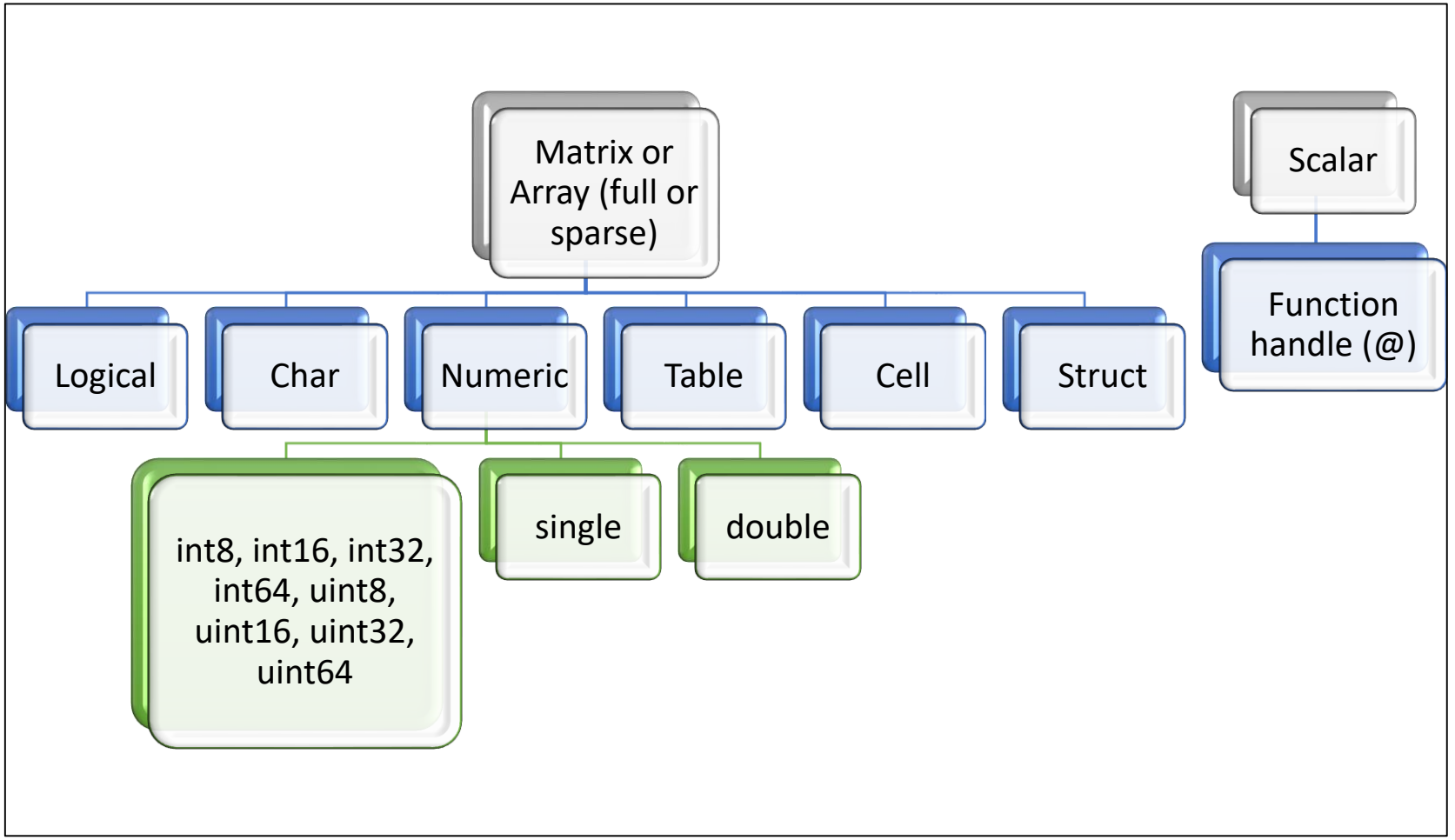


End of Part 1
Please start on Exercises 1 and 2

Part 2
Data Types

Data Types

- Just like humans remember numbers and words, computers can too, however we need to specify what we want them to remember
- Each datatype below allows certain types of information to be stored, and certain operations

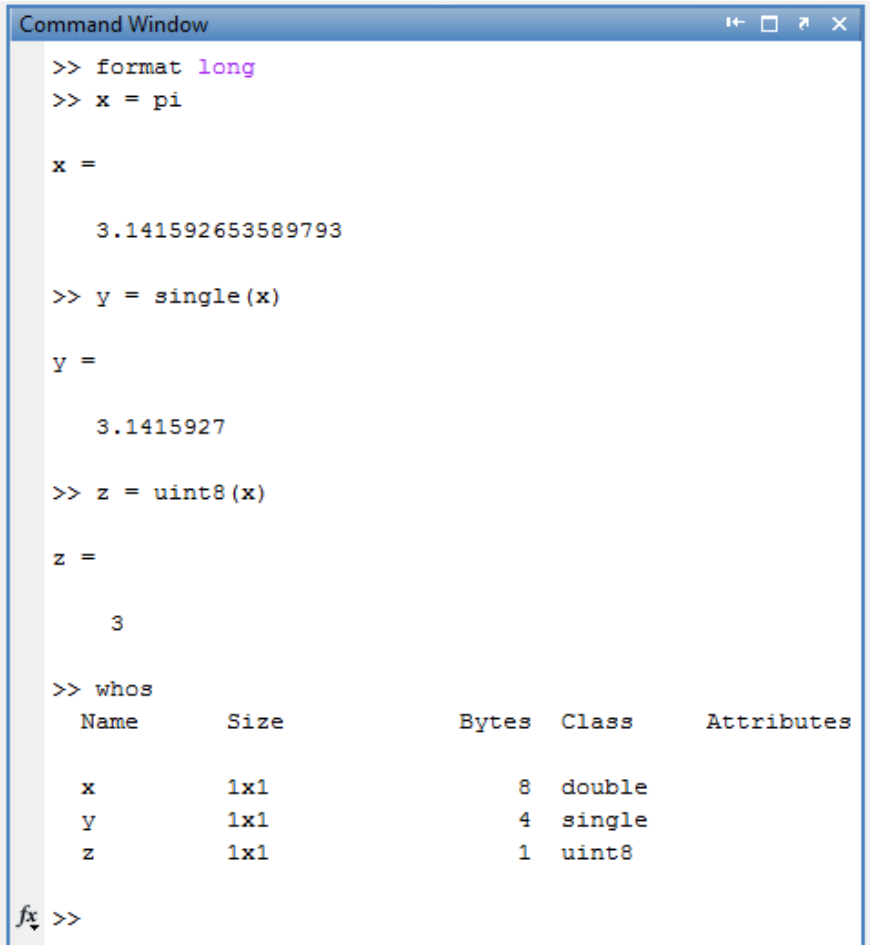


Data Types – Numeric Classes

- Default numerical data are double precision floating point (occupies 64-bits in memory)
- Single precision floats are supported in more recent versions of MATLAB and they occupy 32-bits at the cost of lower precision.
- Signed and unsigned integer types are defined.
- Not all functions work for all data types. For example, *sin* is a built in MATLAB trigonometric function which is not defined for integer types.
- Some commands (e.g. sequential indexing) require integer valued inputs but work with both integer and floating point data types. For example, you can’t go to position 1.5 in an array, it is only position 1 or position 2.
- The help documentation will help you identify which data type to use in each situation (help and doc).

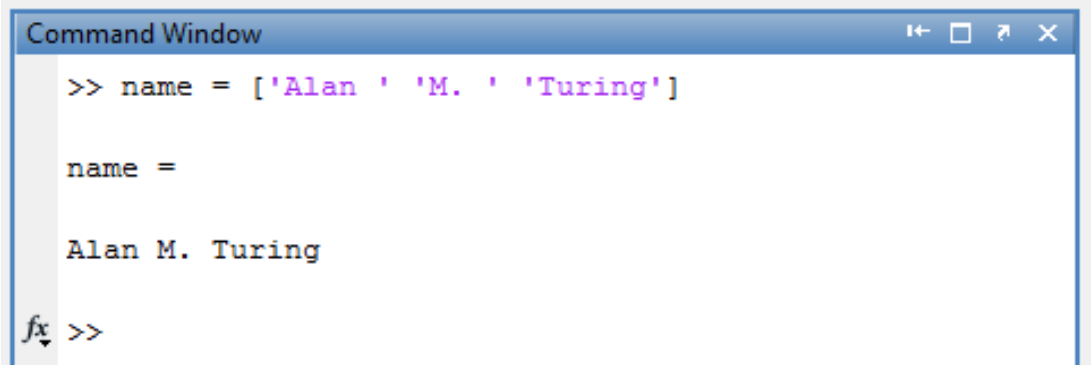
Data Types – Numeric Classes

- You can convert between datatypes using the function described in **doc datatypes**
- Converting between data types is called ‘casting’

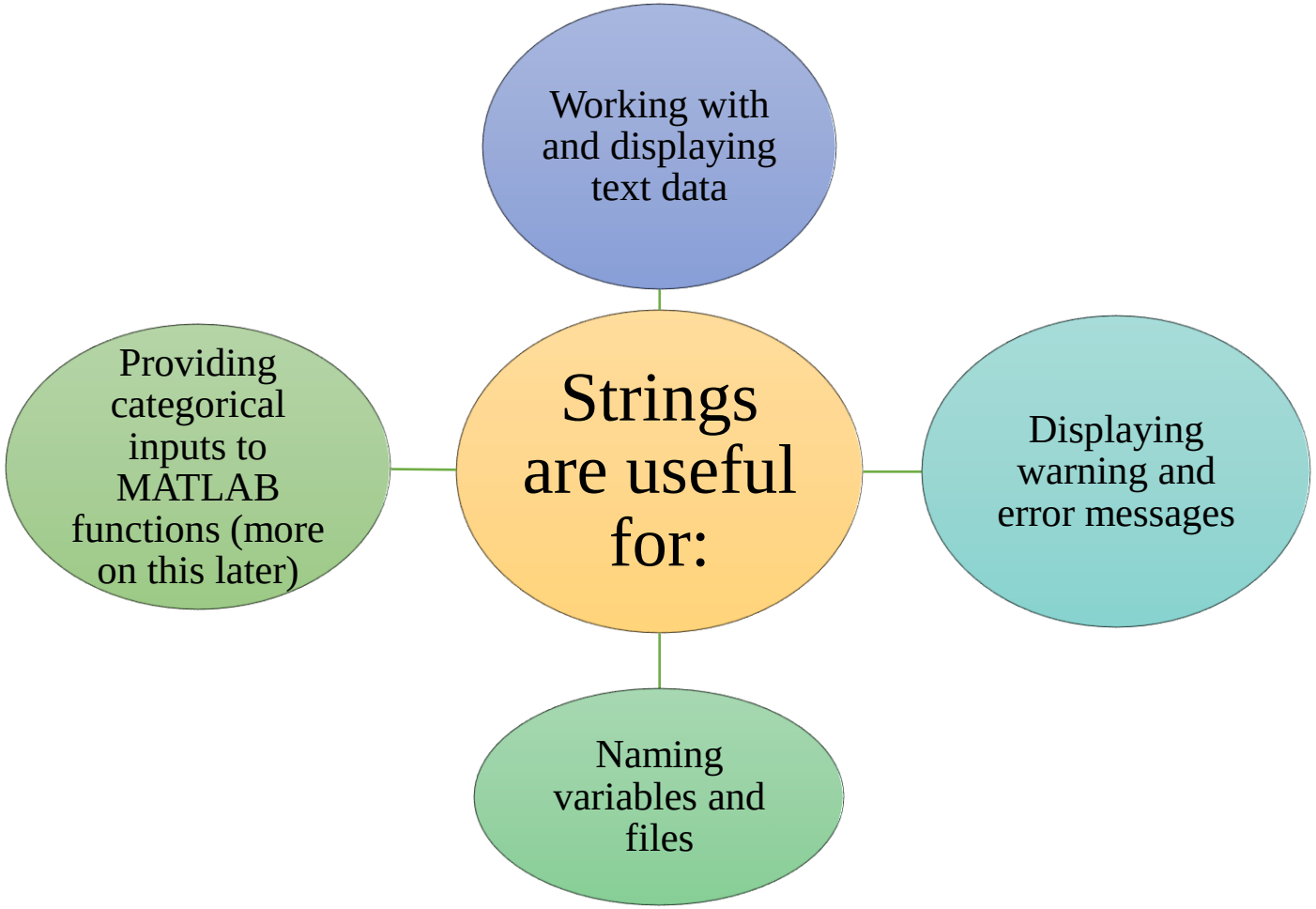


Data Types – Character Class

- Text data are represented using the **character (char)** data type.
- A **string** is a vector (series stored in sequential order) of characters.
- String variables are surrounded by single quotes and highlighted in pink.

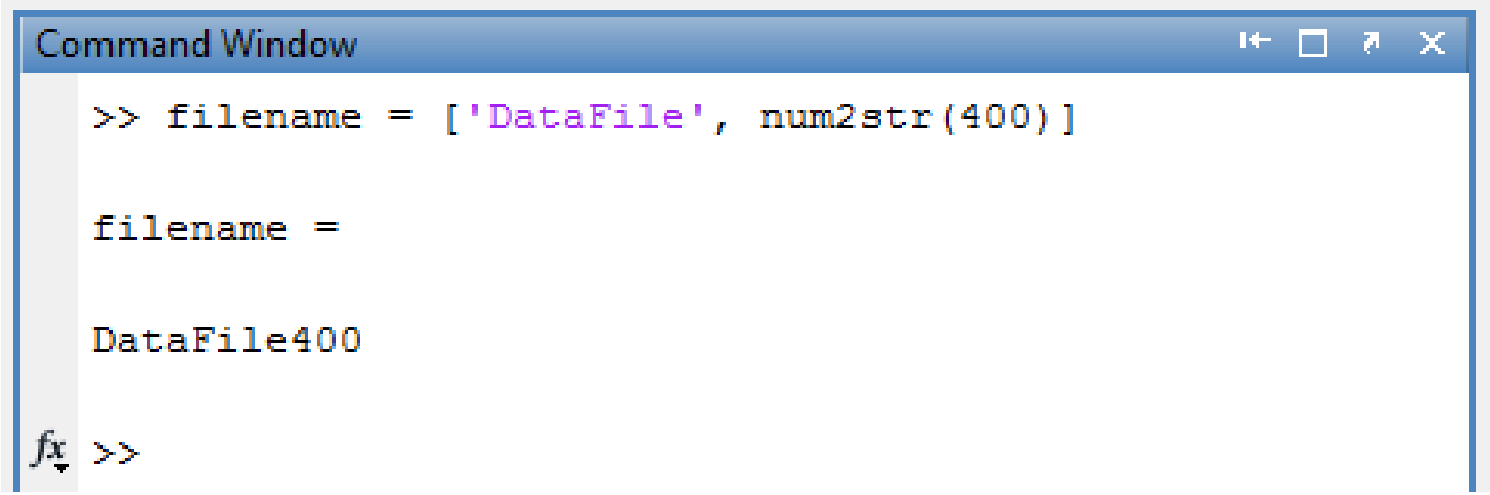


Data Types – Characters and Strings



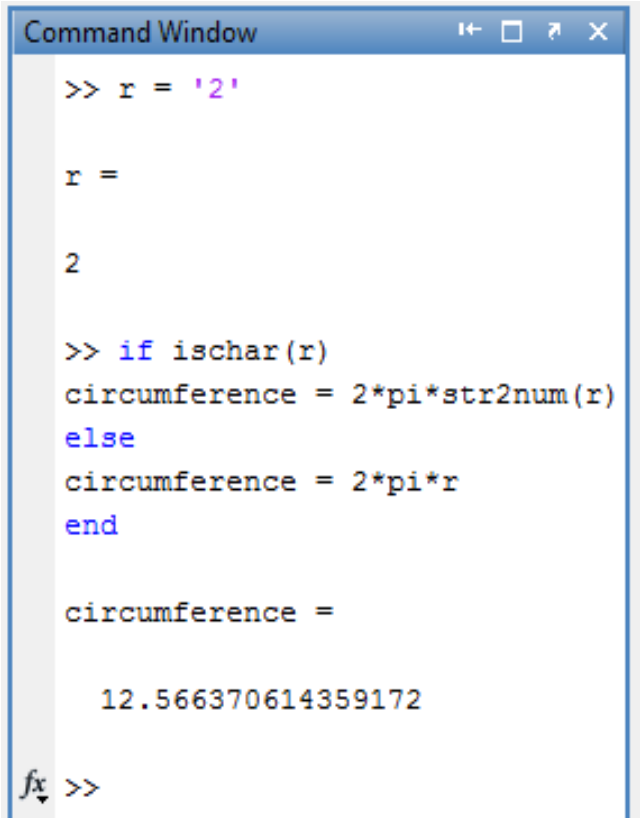
Data Types – Characters and Strings

- Some useful functions:
  - **str2num**: converts strings to numeric values
  - **num2str**: converts numbers to characters or string values
    - It can be combined with matrix concatenation (more on this in the next section) to reference a numbered file.



Data Types – The Logical Class

- The logical data type represents a logical true or false state as 1 and 0, respectively.
- A **logical expression** is a piece of code that returns a logical value.
- Logical values are used:
  - For matrix indexing
  - In conditional statements (more on this later)

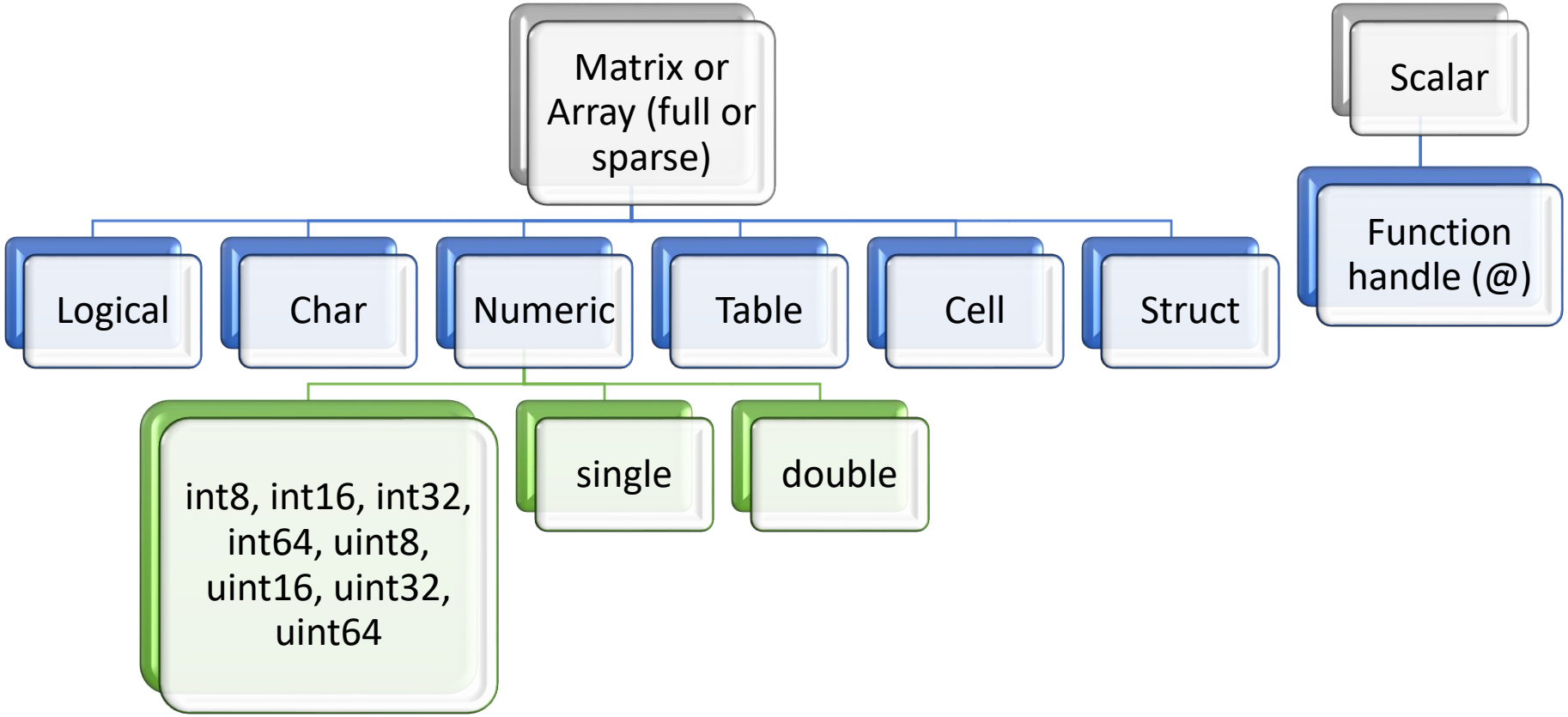


End of Part 2
Please start on Exercise 3

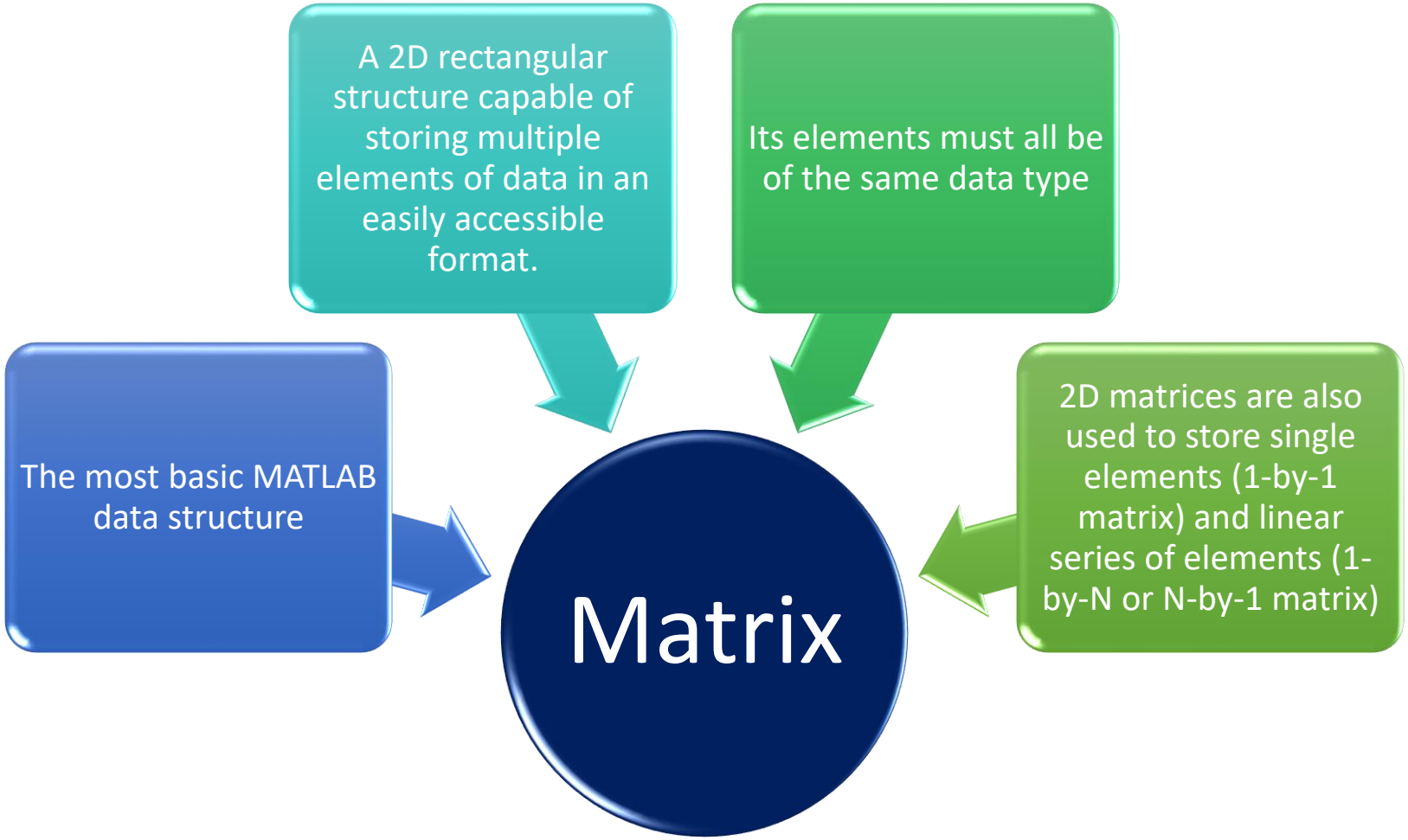
Part 3
Matrices

Matrices

- Almost everything in MATLAB is stored as a matrix, even all of the data types we discussed in the previous section

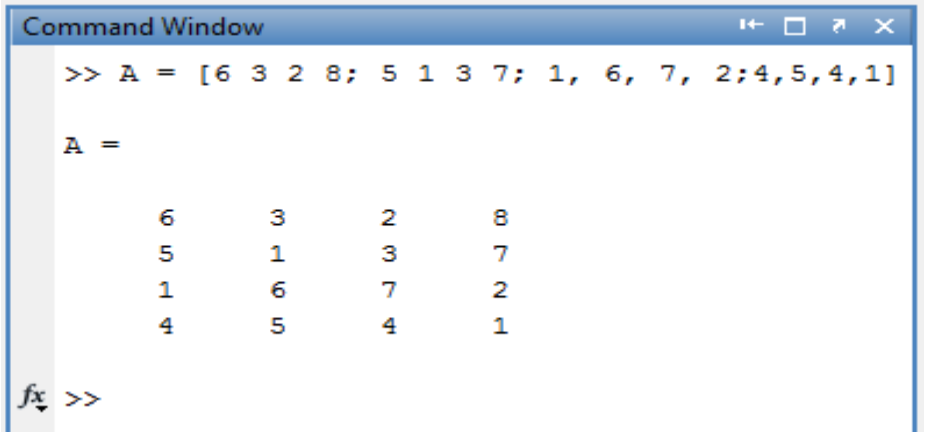


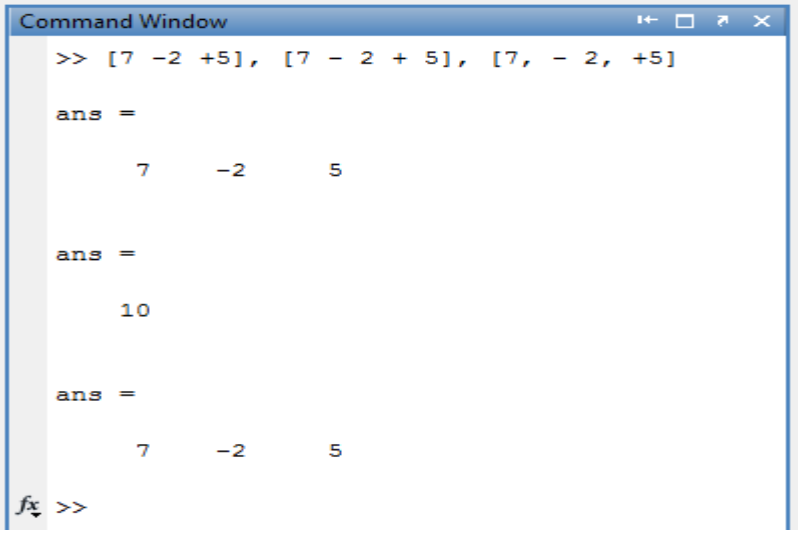
Matrices

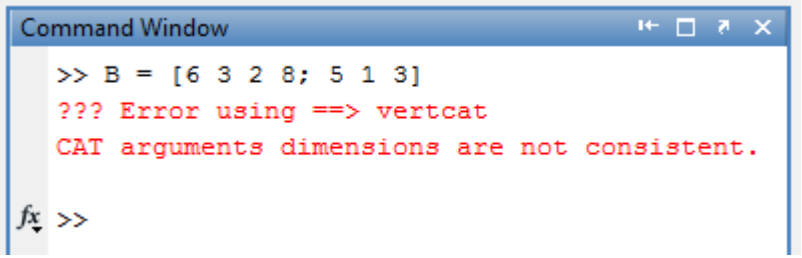

A 2D rectangular structure capable of storing multiple elements of data in an easily accessible format.
Its elements must all be of the same data type
The most basic MATLAB data structure
Matrix
2D matrices are also used to store single elements (1-by-1 matrix) and linear series of elements (1-by-N or N-by-1 matrix)

Matrix Creation (1/2)

- Using the **matrix constructor operator [ ]**
- Within the matrix constructor:
  - Matrix elements are defined row-by-row with each row terminated with a semi-colon (;)
  - Elements within a row are separated by commas or spaces.
  - Spaces matter with signed data (commas are safer).
  - As matrices are rectangular, all rows must have the same number of elements.

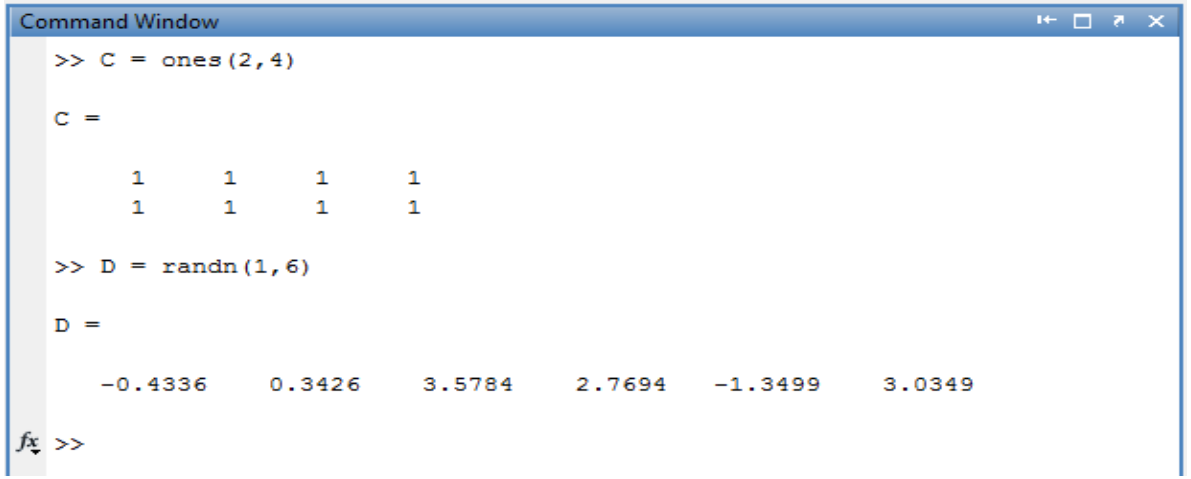






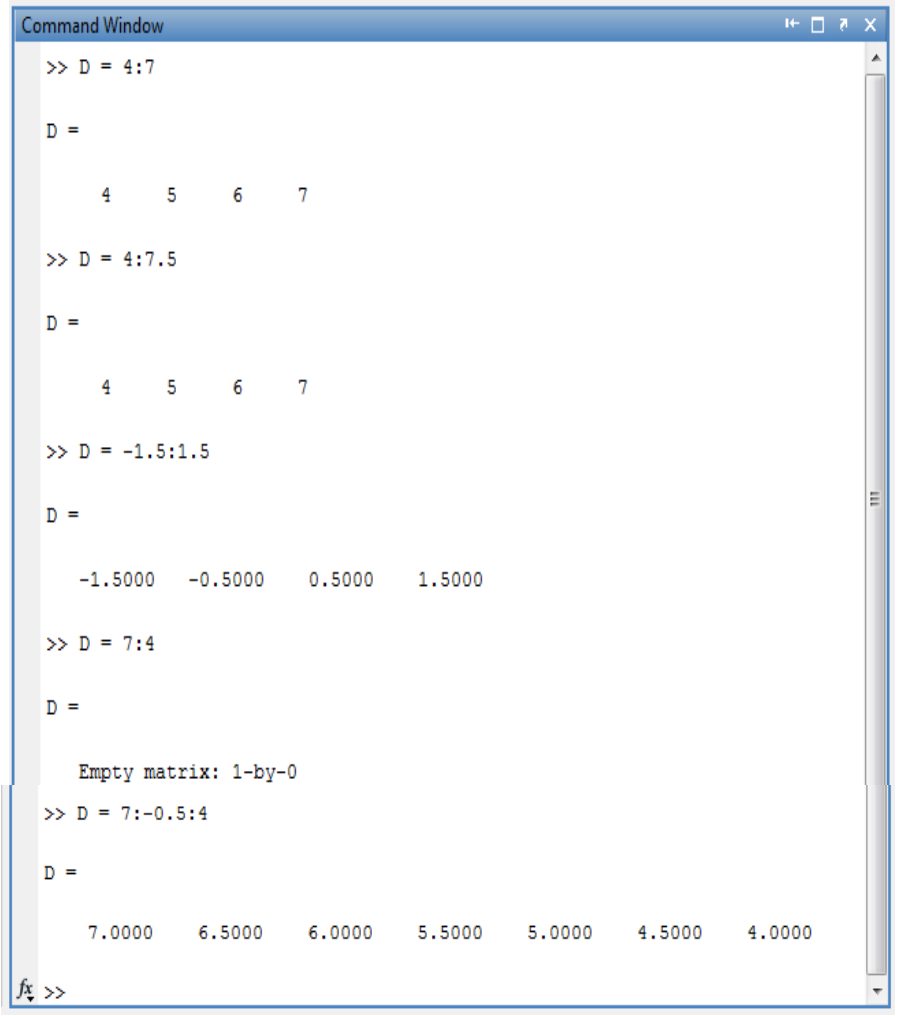
Matrix Creation (2/2)

- Using **built-in commands (functions)**
  - **ones**: Create a matrix of all ones
  - **zeros**: Create a matrix of all zeros
  - **rand**: Create a matrix of uniformly distributed random numbers
  - **randn**: Create a matrix of normally distributed random numbers
- The dimensions of the matrix are defined through the input arguments of the function.



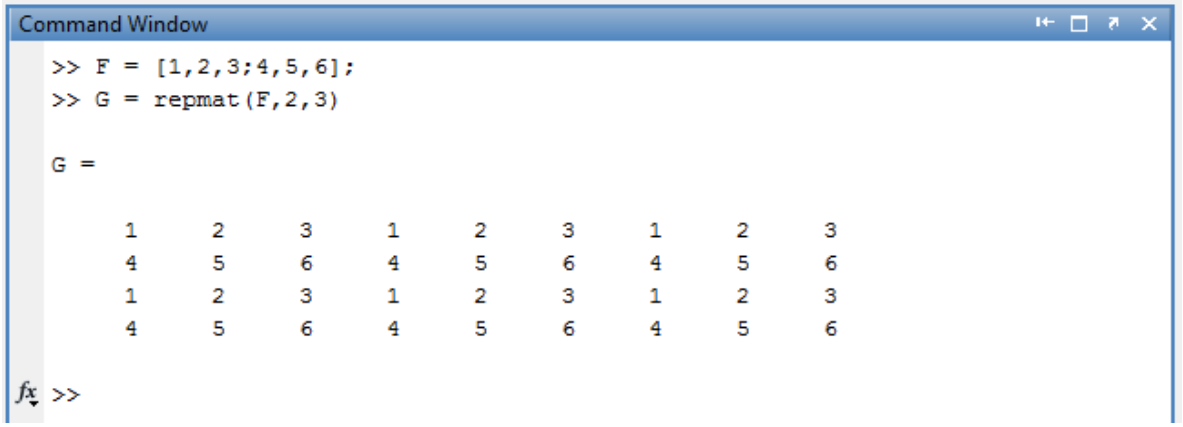
Generating Numeric Sequences

- The colon operator is used to generate numeric sequences
- **first:last** generates a vector of sequential numbers from the **first** value to the **last**. Default increment step is 1.
- A sequence can include also negative and fractional numbers.
- If **last< first**, MATLAB returns an empty matrix.
- For non-default stepping: **first:step:last**



Generating Numeric Sequences

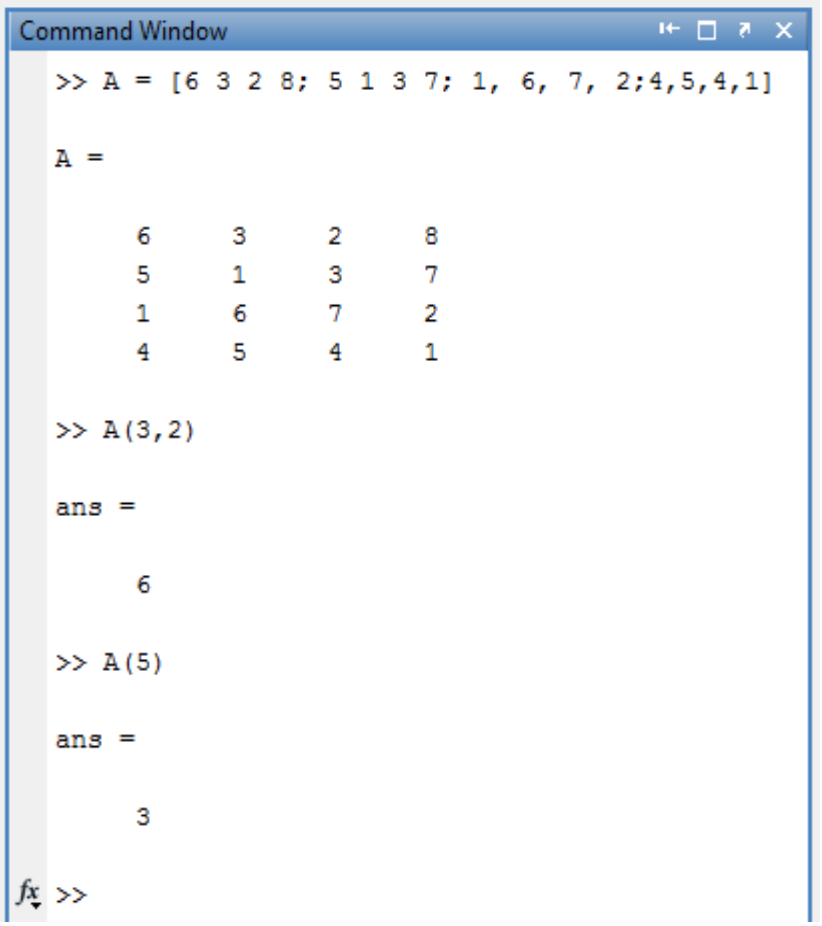
- Concatenation = create a new matrix by joining one or more matrices.
- Using the **brackets operator [ ]**.
  - C = [A B] or C = [A, B] for horizontal concatenation
  - C = [A; B] for vertical concatenation
- Using **built-in functions**
  - **cat, horzcat, vertcat**
  - **repmat**: creates a matrix from tiled copies of a smaller matrix.



Matrix Indexing

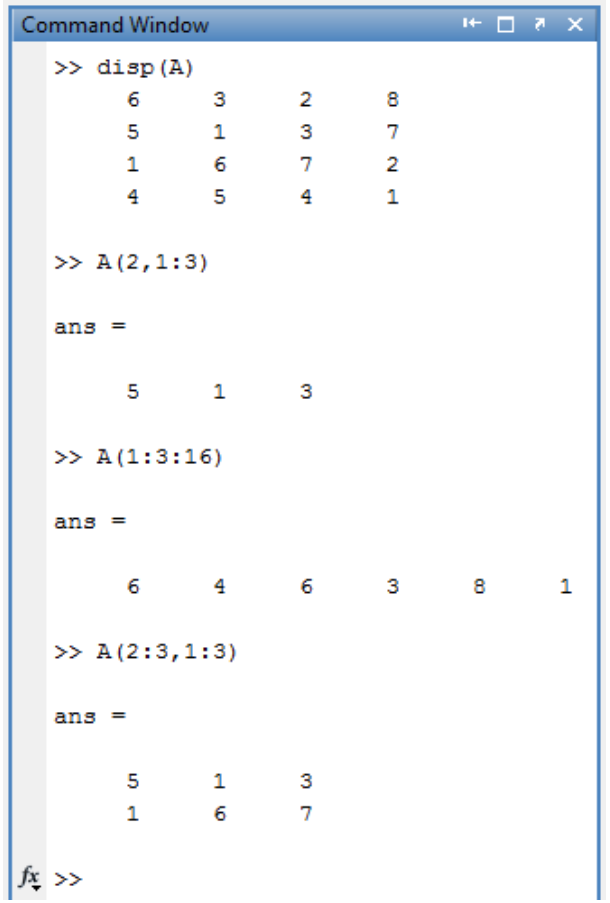
- There are two ways to index a single element in a matrix:
  - *Row-column* (subscript) indexing **A(row, column)**
  - *Linear* indexing **A(ind)** where matrix elements are counted downward through successive columns.

- To convert between the two index styles: **sub2ind** and **ind2sub**.



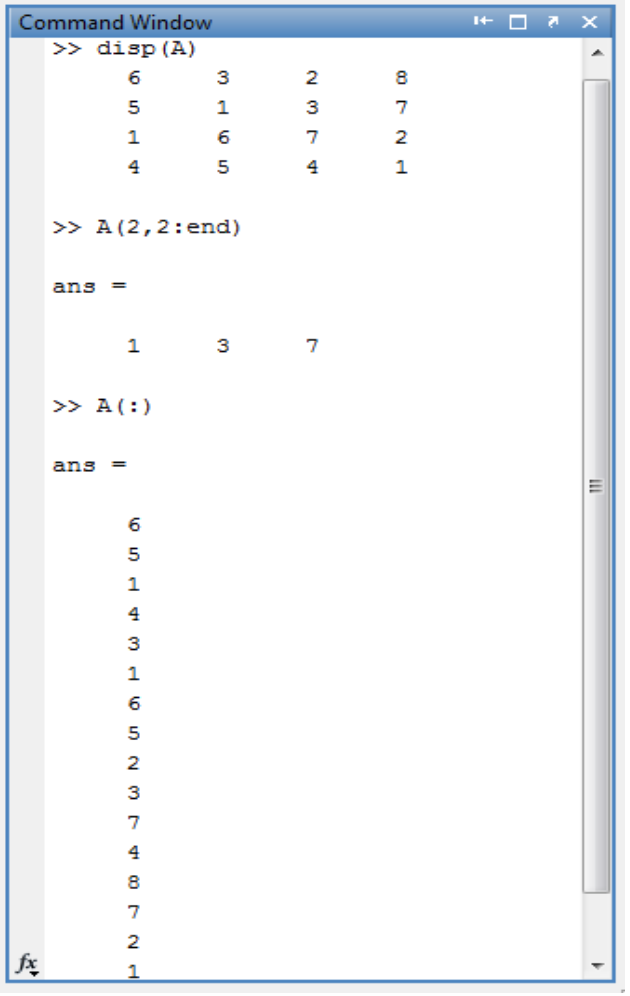
Matrix Indexing – Sequential Elements

- Instead of using a single integer for indexing a single element, we can use an **integer-valued numeric sequence** to access **multiple elements**.

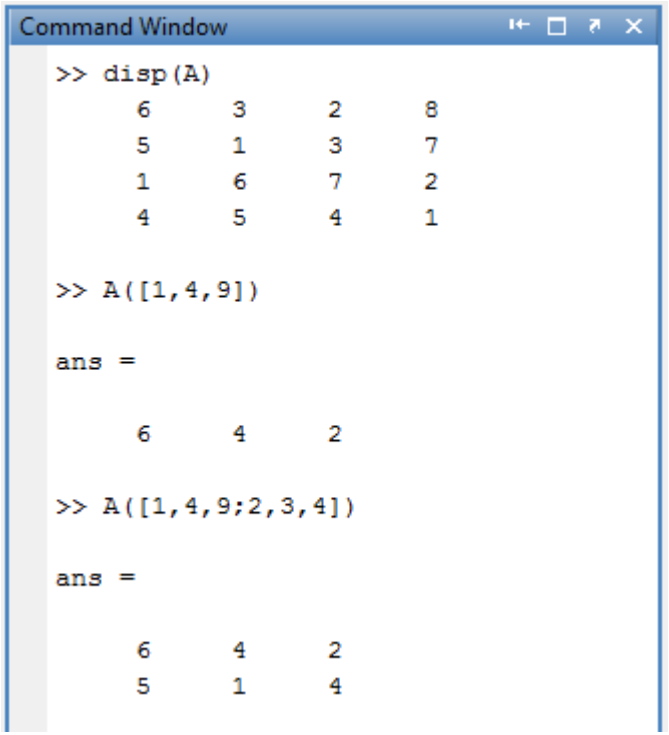


Matrix Indexing – Sequential Elements

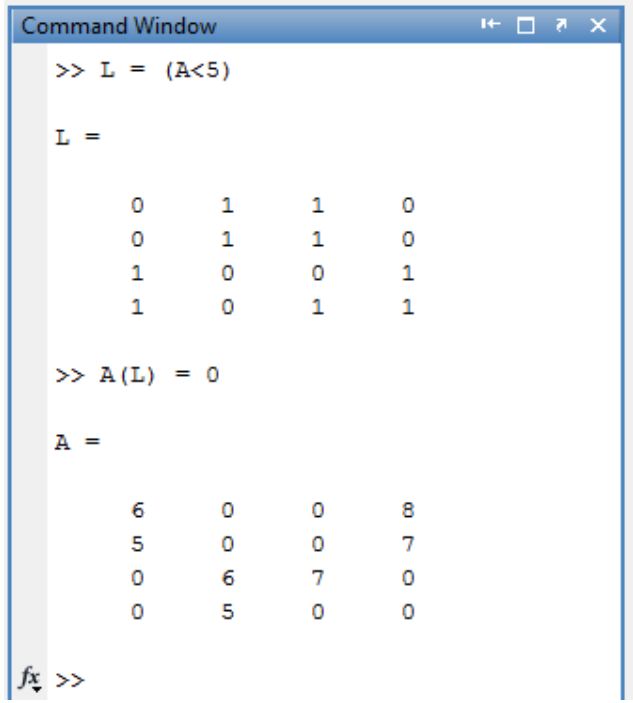
- The keyword **end** designates the last element in a particular dimension of a matrix.
- The colon operator by itself refers to all elements.



Matrix Indexing – Non-Sequential Elements

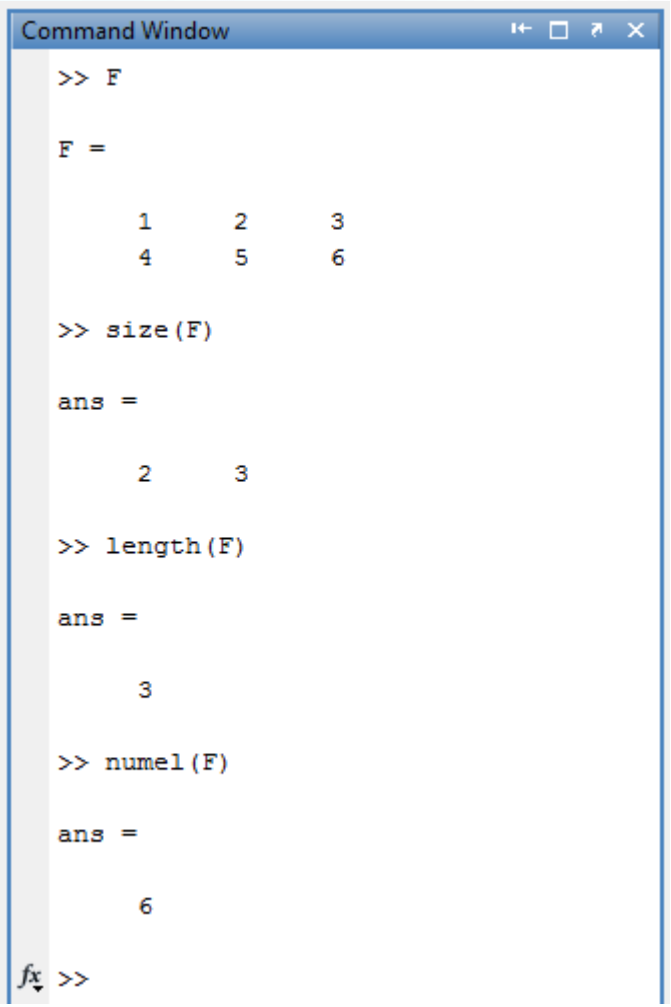


- *Left:* Using an integer-valued matrix
- *Right:* Logical true and false are represented as 1 and 0. Logical values can be used for arbitrary matrix indexing.



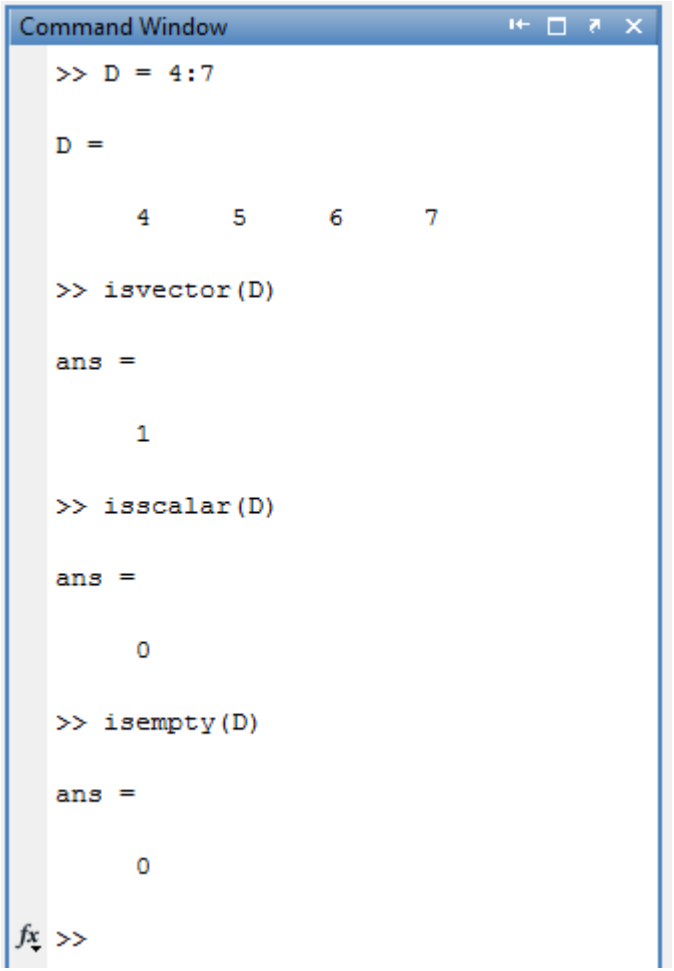
Matrix Information

- The following functions reveal information regarding matrix ***size*** and ***shape***:
  - **size**: The length of each dimension
  - **length**: The length of the longest dimension
  - **numel**: The total number of elements



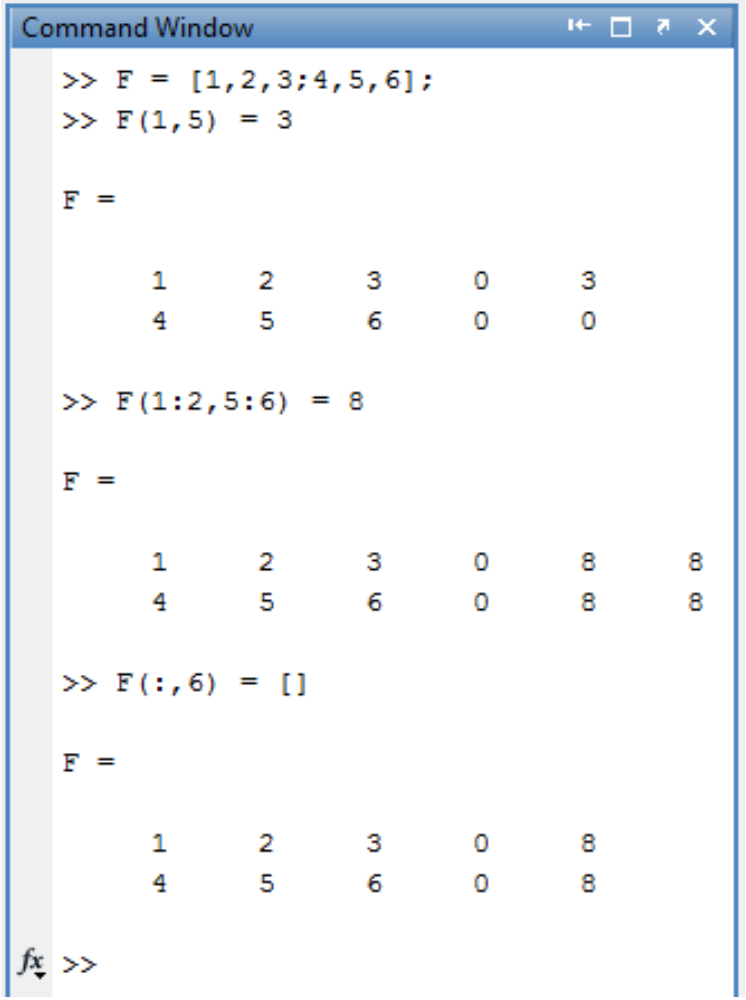
Matrix Information

- The following functions reveal information regarding matrix ***type*** and ***structure*** returning a true value on success or false otherwise.
- Examples are:
  - **isfloat**
  - **isinteger**
  - **isnumeric**
  - **islogical**
  - **isvector**
  - **isscalar**
  - **isempty**



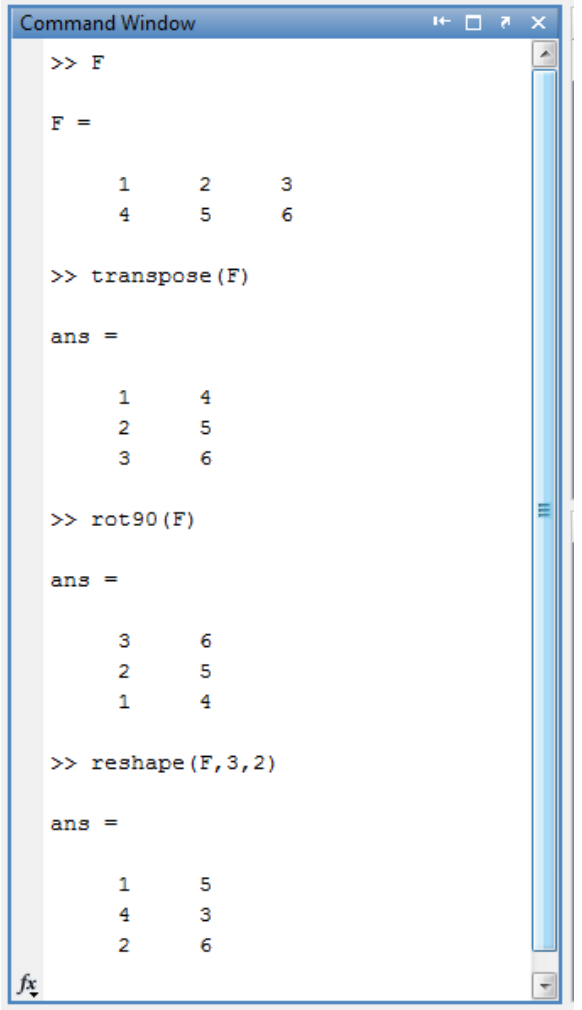
Matrix Information

- Matrices in MATLAB can be dynamically resized.
- Matrix expansion through concatenation.
- If writing to a location outside the current bounds of the matrix, MATLAB automatically pads it with zeros where a row or column is not completely specified.
- By assigning the empty matrix [ ] rows and columns can be deleted.



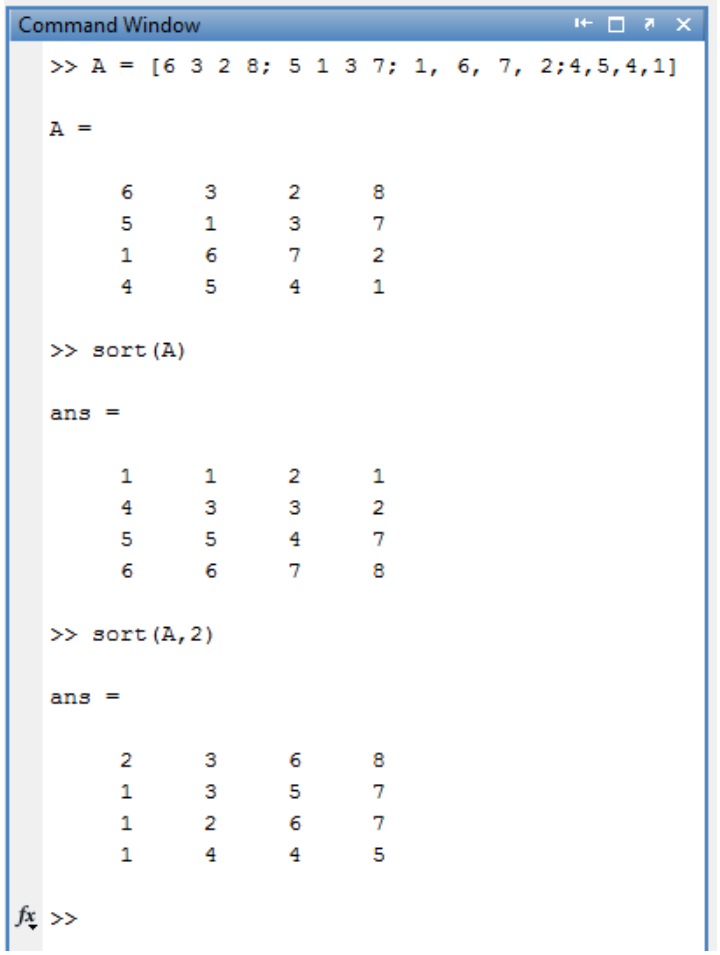
Matrix Reshaping and Shifting

- The following functions modify matrix shape or ordering:
  - **flipud**: Flip matrix in up/down direction
  - **fliplr**: Flip matrix in left/right direction
  - **flipdm**: Flip matrix in the specified direction
  - **rot90**: Rotate matrix anti-clockwise by 90 degrees
  - **transpose**: Flip matrix about its main diagonal, turning row vectors into column vectors and vice versa.
  - **reshape**: Modify the shape of a matrix.
  - **circshift**: Circularly shift matrix contents.



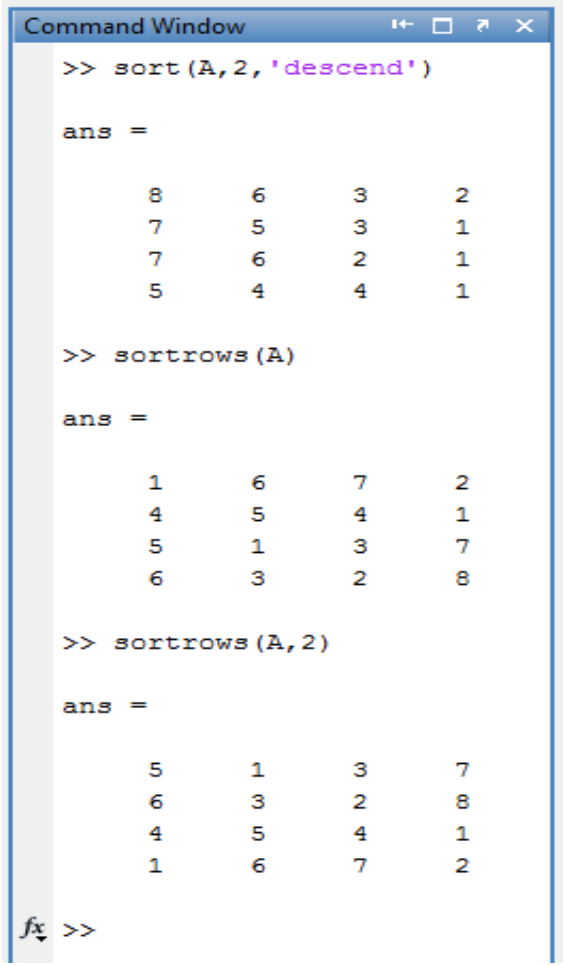
Matrix Sorting

- The **sort** function sorts matrix elements along a **specified dimension** (1 for columns, 2 for rows)
- By omitting the specified dimension causes the function to operate column-wise.



Matrix Sorting

- The **sort** function sorts by default in ascending order.
- Using an additional argument can be used for descending order.
- The **sortrows** function keeps elements of all rows in their original order and sorts the rows according to the order of the elements in a specified column.



End of Part 3
Please start on Exercises 4 and 5