



Exported for Brian O'Connell on Tue, 27 May 2025 19:30:41 GMT

Programming 5 Pre-LAB: Intro to MATLAB

The following is our start into MATLAB. You will not need your Arduino for this lab but we will eventually get MATLAB communicating with Arduino hardware.

We're going to start with installing MATLAB

Installing MATLAB

The purpose of this assignment is to instruct and guide you on how to do the following:

- Access or install MATLAB
- Start a new script in MATLAB
- Save your work and recover your work
- Properly set up and submit documents for MATLAB assignments

Getting Started

As with all the software we will be using, it is accessible through VLab or available for download and installation on your personal computer. There are also pros and cons to each option.

Before going over those Pros and Cons though, due to the nature of our MATLAB curriculum and some of the limitations of VLAB, I will be requiring students download and install MATLAB locally. If that is an extreme hardship, we can work to get you set up through VLAB. This will take some troubleshooting because VLAB has historically had trouble connecting with external hardware on some operating systems. We will eventually have Arduino and MATLAB communicating, so that's just easier to do if done locally.

FOR NOW THOUGH, you can access through VLAB if that's most convenient. Just be aware that we will get to a point where you have to install it locally.

Pros and Cons for Local or VLAB

Installing Locally - Pros

- Instant access
- Not susceptible to internet access
- Easier to connect with external hardware
- Graphics not an issue for the lab assignments

Installing Locally - Cons

- Requires significant hard drive space – 4+ GB
- Requires minimum CPU power – Up to 8GB of RAM

VLAB - Pros

- All course instructions assume you're using the versions available on the VLAB
- You can save your files on your partition of the server, saving more hard drive space

VLAB - Cons

- Dependent on internet access
- If many students are accessing the server, it may slow down
- Has problems connecting to external hardware, which we will do

Access through VLab

- Connect to VLab through the VMWare Horizon Client
- Search for MATLAB
 - VLAB is typically 1 year or so behind on the version they have installed so it's likely a 2022 version but might be 2023. It'll handle what we're doing in class.
- Select and start MATLAB
- Now skip ahead to Getting Started with MATLAB

Install MATLAB Locally

Do not try to install MATLAB on VLAB. It's already there. It won't let you.

Everyone should eventually get MATLAB installed locally but if you're using VLAB for now, skip this section but you should come back to it eventually.

The FYELIC sharepoint has a video on installing MATLAB which walks you through the steps on this link: https://service.northeastern.edu/tech?id=kb_article&sys_id=3155c16c1b08f01064c11f8a234bcbb9.

My only update would be:

- When you get to step 10: "**Select** the products to install and **select** Next." include the following products to save yourself time later:
 - **Database toolbox**

You can include any products you want. Those listed are available to you as a NU student. You can also add them on later as well.

If there are any issues with your installation, contact the IT service desk. Information is at the bottom of the instructions page linked above. You can also message me on Teams or visit FYELIC and meet with a Red Vest for help as well. We're familiar with a lot of the common issues. Please let me know if there are any issues, even if resolved, so I can let IT know or include the fix in here for next semester.

You should now have your own copy of MATLAB on your personal computer to use with our MATLAB curriculum in this course.

Getting started with MATLAB

After completing P5L, you should be able to:

- Access MATLAB
- Run an M-File
- Print the M-File and Command window as a pdf

This lab will go over the basics of MATLAB. This includes:

- Some basics about developing code
 - Reiterate the importance of pseudo-code
 - Commenting your code

- Debugging
- General MATLAB features
- Variables
- Data Types
- Scalar, array, matrix
- Arithmetic
- Program Inputs & Outputs
- Array Creation
- Array Math
- File input/output
- Plot command
- Plot Features

This may seem like a lot but it's mainly introductions to these elements within MATLAB. Some will be a little familiar as they have corollaries in other topics, some we've covered.

What is MATLAB?

MATLAB is a programming platform that was designed for lab settings with engineers and scientist specifically in mind. This matrix-based programming language accommodates primarily natural expressions of computational mathematics. The language, variety of integrated apps, and built-in functions enable quick development of programmatic solutions for your engineering design problems. With it you can:

- Analyze data
- Develop algorithms
- Create advanced models
- Develop custom functions and full applications
- Connect with embedded devices

MATLAB is used by millions of engineers and scientists in their industrial and academic institutions. It is used for a wide range of applications including signal processing, control systems, image and video processing, machine learning, control systems, testing and measurements, finance, and even computational biology.

You will have to make use of MATLAB in your engineering education and will likely need it in your engineering career.

Why use MATLAB?

MATLAB is designed primarily to speak math. It expresses and processes information in matrix form natively and therefore handles matrix and array mathematics easily. Linear algebra in MATLAB looks like linear algebra in your textbooks and the same is true for controls, data analytics, signal processing and many more complex mathematical disciplines.

In the same way that it is designed specifically to function for mathematics, it uses language similar to that used by engineers and scientists. The function names are familiar to the standards you already know. The environment and their supporting documentation is design and written with engineers and scientists in mind.

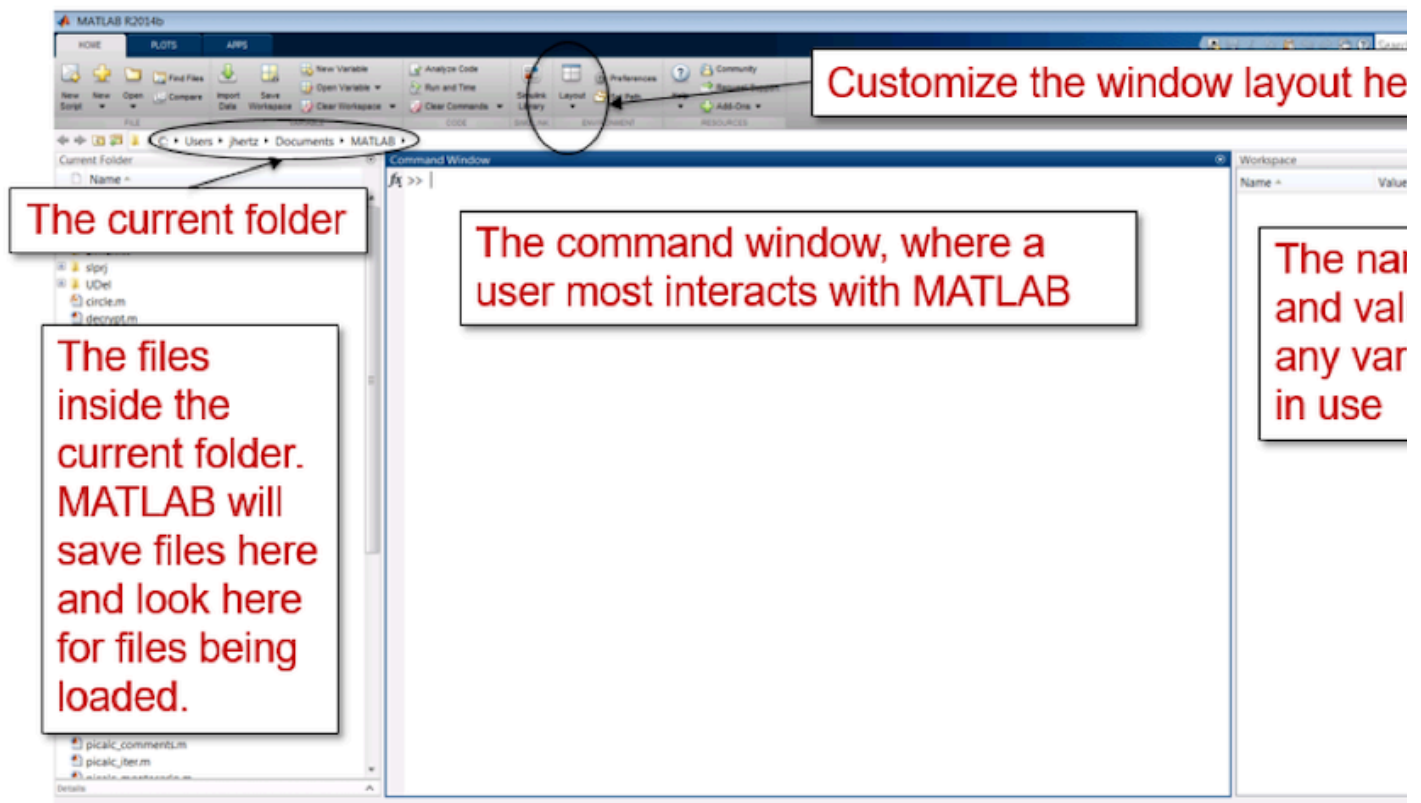
It's extensive use has helped it to be one of the more professionally developed and rigorously field tested support tools available. Interactive applications provide immediate feedback and a large collection of algorithms, functions, and apps for specialized applications are readily available, developed and promoted by MATLAB and their growing user community. That large use community trusts MATLAB. NASA trusted it to help send spacecraft to Pluto and hospitals have trusted it to match transplant patients with organ donors. You can trust it to help you with your homework over the next few lessons and in the coming years.

USING MATLAB

The User Interface

The [MATLAB user interface](#) has a few basic components.

- Ribbon
- Current Folder
- Command Window
- Workspace



This hasn't changed much over

Ribbon

The ribbon functions like in other programs. It provides button click access to many of MATLAB's tools.

Current Folder

This is a feature you must pay attention to. It is simply a directory window like you've dealt with in any operating system. The important aspect is that this indicates the folder you are currently working in. You should make sure it's set to where you want to store your MATLAB assignments so you know where they are and can access them. When we get to writing our own custom functions, you'll need to have that function in the current folder for MATLAB to know where it is when you run its command.

Command Window

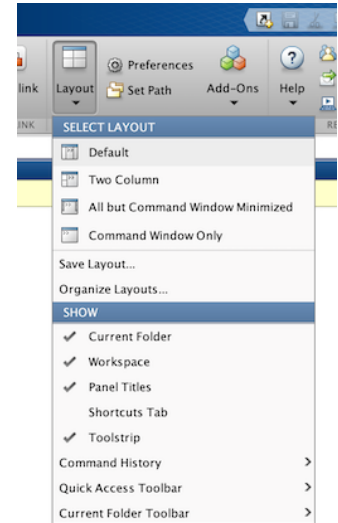
The command window is where you can manually enter commands and where the results of those commands will print out. This will be one of your primary interfaces with MATLAB.

Workspace

The workspace is where all current information is stored in MATLAB. If you create a variable to store information for later, it appears here.

Customize your Window Layout

In the event you accidentally close one of the primary windows, rearrange them inadvertently, or alter the user interface without knowing how to return it to a working layout, this is how. ***This happens more often than you'd think*** so keep the following in mind. Select the **Layout** button of the **Home** tab of the *Ribbon*. This will give you a drop-down of different layout options. Select one that works for you. The one I present above is the default layout. This will come in handy if you ever accidentally close one of the windows and need to reopen it.

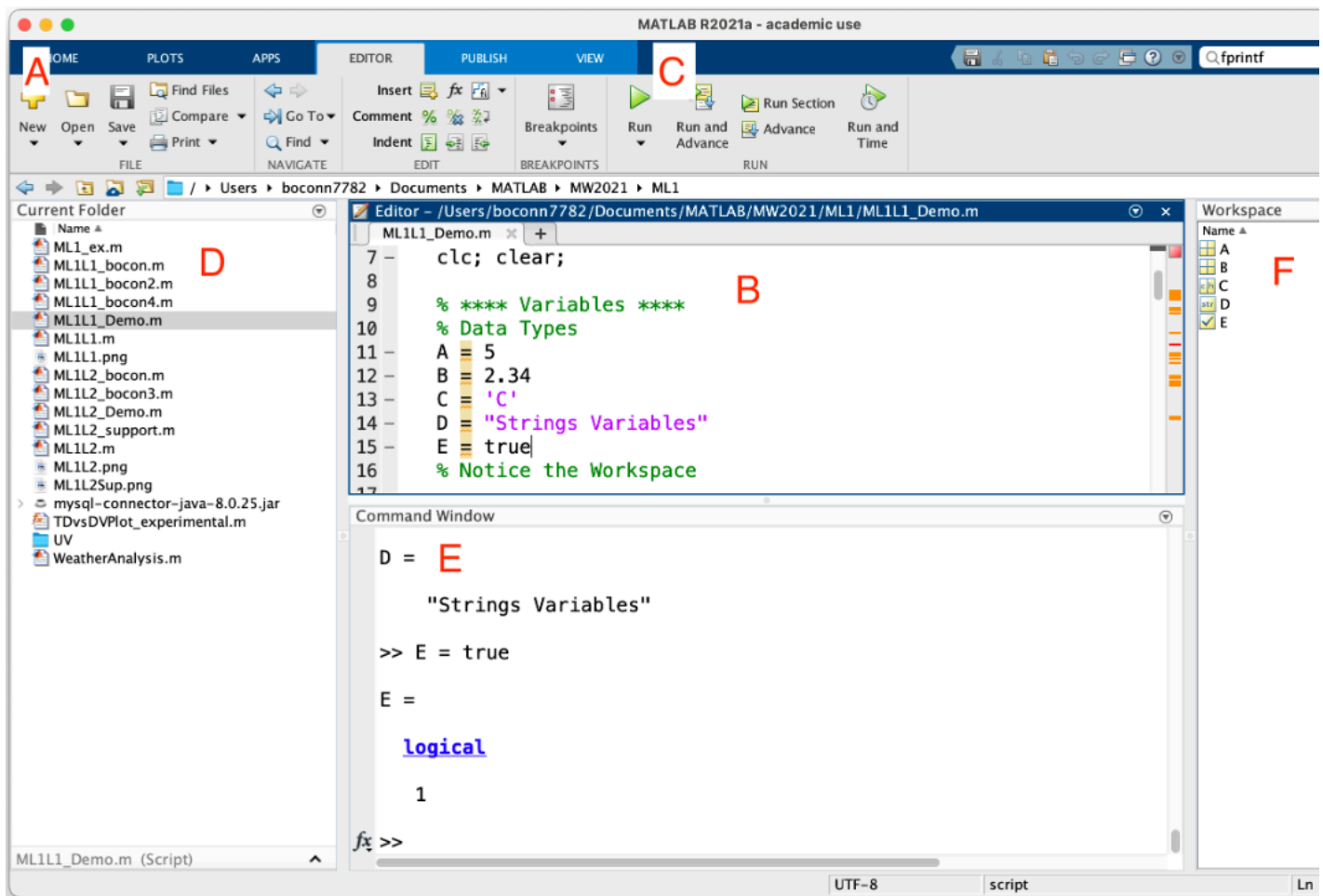


Preferences

Next to the Layout button is the **Preferences** button. I won't get into details about the preferences here but if you want to update the workspace, color scheme, or other aspects of the MATLAB interface, you can do so here. For instance, I prefer coding with white text on black so that is how I have it set up in my personal settings. For images in the pre-labs and in class though, I use the defaults layout and color schemes.

Getting Started MATLAB

Some of this is repeated from the **User Interface** section above. That section talked about what those elements are. This will be more about the basics about how to use them, in particular for following along with the rest of this Pre-Lab, focusing on the sections of the UI you need to interact with and pay attention to while using MATLAB. There will be some repetition throughout when necessary, particularly since the UI does change somewhat and some elements are important in multiple contexts.



The above marks each of the important areas for general use with letters

A - File Panel

The File Panel contains tools for file management. Here you can start, open, and save scripts. Primarily, you'll engage with it to

Select **new** to start a new script. There are 2 types, *script* and *live script*. You will always work with a *script* an m-file type. When you don't have the editor window(B) open, there will be icons for creating a *new script* or a *new live script*. You will typically be creating a **New Script (.m file)**

The first time you save a script, a window will open for you to name it properly. To rename it, you can select *Save as...* under the Save icon's drop down.

B - Editor Window

Your setup won't currently have this. It only appears when you open or start a script.

The editor window displays any scripts you're currently working on. You typically want to be doing your work in a script so that you may more easily manage multiple lines of code and include comments to describe what they are doing.

When you open a MATLAB file, it will appear here. This is where you can edit that file and then save it. There is a tab across the top that shows what files are open. These are typically your MATLAB script(s).

It's common practice to have a test script open, to allow you to experiment with larger portions of code without making major edits to your main program. For larger scripts, it's not uncommon to have a dedicated development script for experimental purposes before committing code to the main script.

C - Run Panel

The run panel contains all commands for running your scripts. You will largely only deal with **Run**. This command compiles and runs your script, sending any outputs to the command window(E) and saving any variables created in the workspace(F).

WARNING: **Run** will queue up multiple runs if you hit it multiple times. If your script seems to be just restarting on its own, you've likely hit run too many times. To cancel the current execution of a run, you can hit **CTRL+C** or **Ctrl+Break**. On a Mac, you also can also use **Command+.** (the Command key and the period key). You will likely have to hit it multiple times to get back to a neutral status with nothing attempting to run (When you just see just the command prompt (**>>**) in the last line of the command window).

D - [Current Folder](#)

The current folder window displays the folder on your computer that you are currently working in and all the content of it. Any new scripts created will default to be created in that folder. You can only run scripts, other than those part of built-in MATLAB functions, that are located in that folder.

E - [Command Window](#)

The command window allows for the entry of individual commands. It should **only** be used for testing a few commands. The command window is not for you to create your entire program in. Think of this as a notepad or scratch paper for your to try out ideas or calculations before adding them to your script in the editor window(B).

The command window is also where any outputs from your scripts will display.

F - [Workspace](#)

The workspace displays any information you've created and stored from the editor window or the command window. Both have access to this information. It shows the name and value of any variable created as well as graphical indicator of the type of data stored. If you double click on it, more detailed or complete information will display in the editor window.



MATLAB Unresponsive

[Show Correct Answer](#)[Show Responses](#)

You've been using MATLAB, running commands and scripts for a bit. It stops responding and you no longer see the command prompt `>>` as the last line in the command window. What do you do?

A

Shut down MATLAB and restart

B

Hit 'Ctrl+C'

C

Hit "enter" or 'return' several times

D

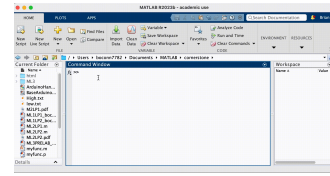
Hit 'Ctrl + R'

MATLAB Basic Command Inputs

We are going to go over a lot of commands in a short time. Not all will be used in the deliverable for this pre-lab. We're just looking for a general familiarity of a lot of these basic elements. When we do get to using them, there will be some review, but it's just useful that they're not 100% new when some come up in class or future assignments. A lot of them are also so when you look at standard examples, you recognize a lot of the common elements. So don't feel the need to memorize and reread this part a dozen times but don't speed skim it either. Try out some of the commands to see what they do in MATLAB. You can always go back if something comes up in the walkthrough activity that you think you'd need a little more review on before implementing.

You can enter commands directly into the MATLAB Command Window simply by typing in that field. The “**>>**” is the command prompt for the window, meaning that this is where you will be typing information for entry into MATLAB. This is specifically for attempting the commands below. You may also put them in a script file then run that script.

Type or copy the following commands into MATLAB's



command window, pressing enter after each line:

1. **5+5**
2. **ans**

NOTE: Due to restrictions in Top Hat, copy-paste is not an option in all browsers. When it's too much typing, I will include a link to content you can copy-paste. I have recently checked though and this bug seems to have been solved, but I want to give that warning just in case since the combination of operating systems and browsers is greater than I can check.

If you're struggling with understanding, I have also found that typing through the commands helps rather than relying on copy-paste. It slows you down a bit, allowing you a little more time to observe and get the concepts.

Another NOTE: I also use a numbered list not because it's necessary for MATLAB but because, in Top Hat, they display as single spaced. Otherwise, with larger code examples, it would become unwieldy to view. It also allows for reference of specific lines. If you copy paste or retype the code, don't include the numbers.

MATLAB will run the simple mathematical equation of 5+5 and output an answer for you. It will also store the answer in **ans**. This is MATLAB's catch-all variable. You'll now see it in the workspace and the next time you run a command, it'll get overwritten with whatever the output of that command is.

Type the following commands:

1. **A=5*5**

2. **ans**
3. **A**
4. **5-1**
5. **ans**
6. **A**

This creates a new variable to store the information. Note that **ans** still contains the information from the first equation we entered and we can still call that.

The “**A=5*5**” is called an **assignment**. It is where you are assigning a value to variable. We established the variable A and then assigned it the result of the expression **5*5**. Let’s take something you may be more familiar with in how it looks, **x=2x+1**. You may look at that as an algebraic equation and could solve it for $x = -1$. What MATLAB does with this is reassign the value of x to $2*x + 1$ based on whatever the last value of x is.

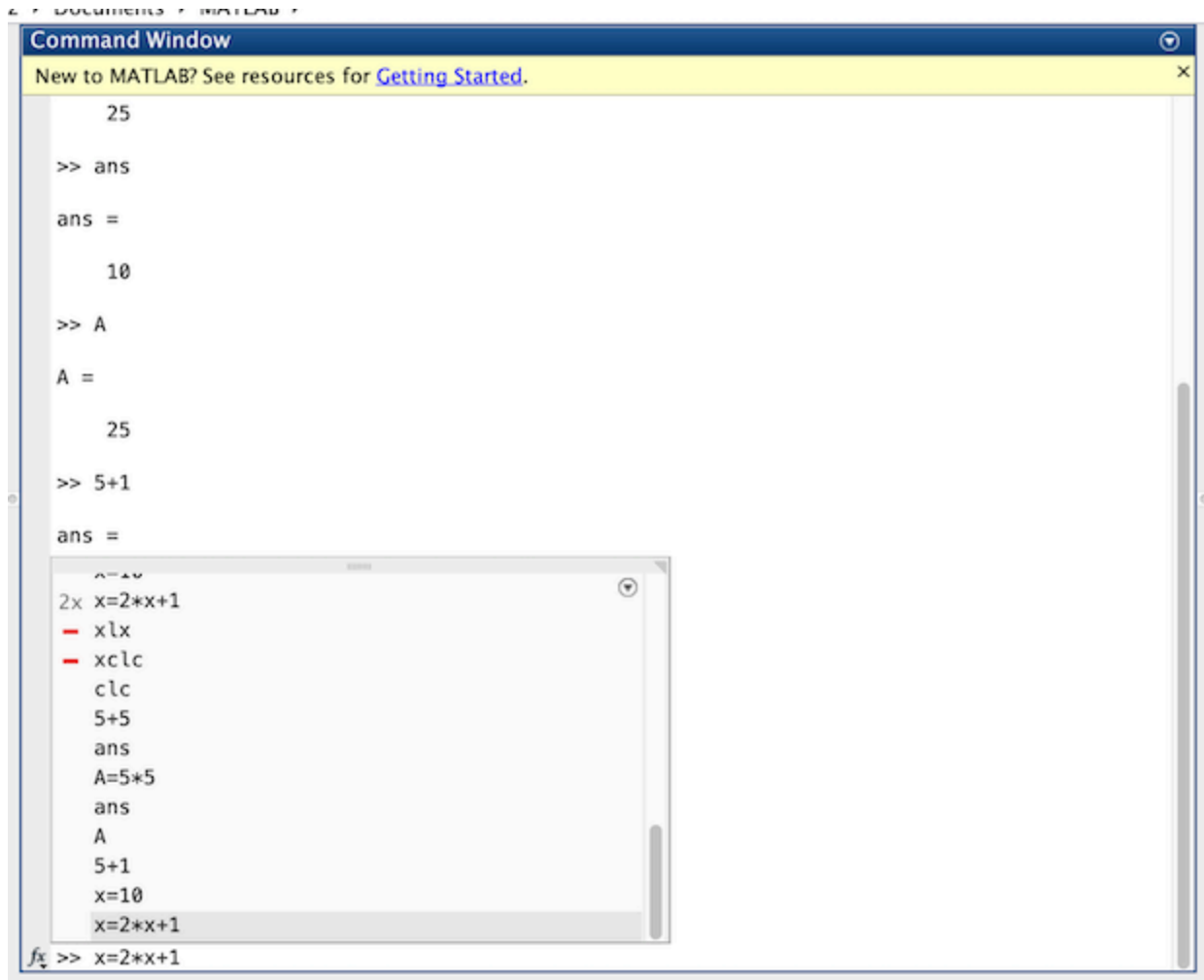
Run the following:

1. **x=10**
2. **x=2*x+1**

If you run **x=2*x+1** again, it’ll reassign x using the new x value solving for $2*21+1$.

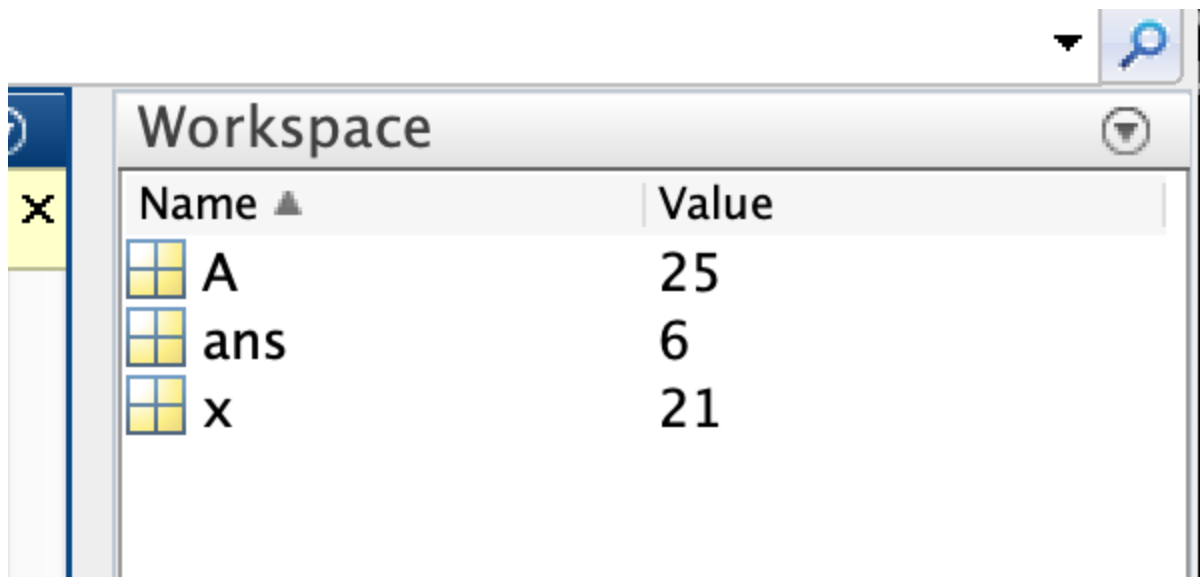
Command History

This is a good opportunity to teach you a useful MATLAB feature, **the Command History**, and how to navigate it. If you want to re-run a new command, you can just hit the **up-arrow** key. This pops up the Command History and starts you selecting the last command. You can cycle through them using the arrow keys. If you hold down the shift key while cycling through commands, you can select multiple commands. When you hit enter, MATLAB will run the selected commands. This is one of the most useful features that students often forget about.



Workspace Stores Variables

If you want to see what variables you currently have created, you can look to the workspace. All active variable in session are stored and displayed there. You can double click on them to view them and even alter them. They open up in a new window as a spreadsheet. Right now we're only dealing with single values so you'll see these in the 1st row and 1st column. We'll get into arrays and matrices later, which will populate more fields in that **Variable - <variable name>** window.



The image shows a screenshot of the MATLAB Workspace window. The window has a title bar with a search icon and a close button. Below the title bar is a table with two columns: 'Name' and 'Value'. The table lists three variables: 'A' with a value of 25, 'ans' with a value of 6, and 'x' with a value of 21. Each variable name has a small icon to its left, which is a yellow square with a blue grid pattern. The 'Name' column has a small upward-pointing triangle next to the header. The 'Value' column has a small downward-pointing triangle next to the header.

Name ▲	Value ▼
A	25
ans	6
x	21

Hovering over the icon to the left of the name will display the variable type.

Built-In Functions

Here we'll go over some of the basic built-in functions and their commands.

Basic Math

We've already shown that MATLAB can handle addition and multiplication. It can also handle subtraction, division, and power. You can use all of these using their standard symbols: +, -, /, *, and ^. For more advanced trigonometric and algebraic functions.

Built-in Trig Functions

You should recognize most of this from your previous courses. Some have modified function names but you can make some inferences

- The usual trig functions - Use radians as an input by default
 - `sin(0)`
 - `cos(0)`
 - `tan(130)`
- Inverse Trig Functions
 - `asin(0)`
 - `acos(1)`
 - `atan(90)`
 - `acos(0)/(pi/180)` - Includes the conversion to degrees with `/(pi/180)`
- Trig Functions but degrees as an input

- `acosd(0)`
 - `sind(270)`
 - Just add a d to the end of the function name for the rest
- Logarithmic Functions
 - The inverse of the natural log
 - `exp(2)` - same as e^2
 - The Natural Log
 - `log(100)`
 - The base-10 logarithm
 - `log10(100)`
- Common functions
 - Absolute Value
 - `abs(-3.1)`
 - Square Root
 - `sqrt(81)`
 - `sqrt((3^2) + (4^2))`
 - Hypotenuse
 - `hypot(3,4)`

MATLAB is by no means limited to the above. These are just some common functions you're likely familiar with.

The above are just examples for mathematical functions you are likely familiar with. You may try running some equations of your own or the provided example code below.

I recommend trying them one or a few at a time. It'll be easier to follow that way since MATLAB will process all copied lines in first then output all the results after.

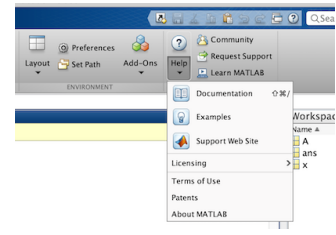
Built-in help

MATLAB does an extremely good job of supporting their product. Their documentation is one of the most well-kept in the industry. They have also integrated into their product well. If you ever need more information about a command type “doc” then that command. This will run the help documentation function with the input of the command you typed.

Run the following command:

1. `doc sin`

A help window should pop up already on the documentation page for the command 'sin'. You can use this for any command. You



can also manually open up the built-in Documentation support and even Examples. You can also go to the Support Web Site as well.



With a little help from MATLAB 1

[Show Correct Answer](#)

[Show Responses](#)

Use the search field in MATLAB to find function for performing a polynomial evaluation. Try using the search term for the activity you want to engage in, "polynomial evaluation." The command is _____(p,x)

Clear Commands

Finally, 2 of the most important commands are `clc` and `clear`.

`clc` will clear all information that is currently displayed in the command window. It doesn't clear the command history though.

`clear` will clear out all variables in the workspace.

To clean up your window and workspace, you will want to occasionally run these commands. For any script you create, you will want to include clear commands at the beginning.

Variables

You've already used a few above but let's get into more detail. Variables store information to be referenced and manipulated later. Think of them as information containers.

Scalars, Arrays, and Matrices

In MATLAB, every variable is a matrix. We will deal with them as scalars, arrays, and matrices, but on the backend, MATLAB stores them as matrices. It's **important** to start thinking about program information in terms of its size and arrangement because most other languages and development environments are not as forgiving or flexible as MATLAB is. MATLAB will allow for expansion, change, and modification in general regarding a variable in its datatype and arrangement. Other languages will crash if that happens.

- A **Scalar** is a single value. MATLAB stores it as a 1 row by 1 column matrix.
 - `X=1` sets a single variable as a 1x1 matrix
- An **Array** is a 1-dimensional series of values. MATLAB stores them as 1 row by multiple columns matrix.
 - `X=[1 2 3 4]` creates an array as a 1 x 4 array
- A **Matrix** is a 2-dimensional series of values. MATLAB stores them as multiple rows by multiple columns matrix.
 - `X=[1 2 3 4; 5 6 7 8]` creates a matrix as a 2 x 4 matrix

More examples:

- `a = 42 % 1x1 scalar`
- `b = [2 , 3 , 4 , 6, 8] % 1x5 Array`
- `c = [1 , 2 , 3 , 4 ; 6 , 7 , 8 , 9; 10, 11, 12, 13] % 3x4 Matrix`

Data Types

There are many data types available in MATLAB. We're going to use a few in MATLAB and will get into more details as move along on.

- [Logical](#)
 - Logicals are also called booleans in other programming languages. These are binary integer representations of True (1) and False (0).
 - I'm starting with this since, although we don't always use it directly, MATLAB outputs booleans for many functions and comparisons, some of which we'll use here.
- [Float](#)

- Floating point numbers are those with decimal places. Its name is a reference to the decimal changing places, ie the decimal point can float.
- This is the default format that MATLAB uses.
- Example using function to check data type
 - `x = 25.783;`
 - `isfloat(x) % Command to ask if a variable is a float`
- [Integer](#)
 - These are whole numbers. Be careful when using them because integers create integers so if you divide them and it results in a non-whole number, it rounds to the nearest whole number.
 - MATLAB defaults to floats so you don't end up using them unless you force MATLAB to. This will not be a necessity for how we plan to use MATLAB but understanding the difference will be important in other programming languages.
 - But just to provide an example of integers in MATLAB, see below. I purposely left out `;` so you can see MATLAB's displaying it as a whole number but treating it as a float.
 - `a = 1; % Will print with no decimal places`
 - `b = 3; % Looking like integers`
 - `c = a/b % Result is not though`
 - `isfloat(a) % Because they were floats`
 - `d = int8(a) % this command forces the number to be an 8-bit integer`
 - `e = int8(b)`
 - `f = d/e % Different result`
 - `isfloat(f) % Because they're integers now`
- [Character](#)
 - This is one of the smallest addressable units in code. For now, it's best to think about it as a single letter. We'll get into what actually defines a character later. For now, you can use it fairly interchangeably with Strings.
 - To create a character array, you need to use single quotations when creating it:
 - `C = 'Hello, world'`
 - `length(C) % returns number of elements in an array. In this case, it will count each character as one element, so 12.`
- [String](#)
 - Strings are what you are more familiar with. These are whole words or phrases. The difference between a string and a character array is that a string can be a scalar while anything more than 1 character will always be an array.
 - To create a string, you must use double quotations when creating it:
 - `str = "Hello, world"`

- `length(str)` % You'll notice this is just 1 as opposed to the character array.

Naming Variables

Naming variables is important. First of all, there are some rules on what's allowed, but we'll get into those soon. When you choose a name, you want to make sure you choose one that is descriptive enough that you'll know what it's for but you don't want it to be so big it's unwieldy. You'll figure out what works for you with experience and particularly as you learn and establish useful abbreviations. For instance if you were going to create an array to store projectile heights at points in a time series, you could use variable names like:

- `Projectile_Heights`
- `Prjctile_Hghts`
- `Projectile_heights_in_a_time_series`
- `X`

All are perfectly valid. Some are more descriptive than others while some are very descriptive but so big they're unwieldy. 'X' is also usable as a variable name. There's nothing about the name itself that violates MATLAB's rules. It gives you no information about what the variable is for though. Now this could be clear in context or with some comments but right now we're only talking about variable names.

Case also matters. Try running the follow series of commands:

1. `clear % Clears the current workspace`
2. `x=1;`
3. `X=5;`
4. `y=7;`
5. `x,X,y,Y`

You will get an output that looks like this:

```
x =  
    1  
X =  
    5  
y =
```

Undefined function or variable 'Y'.

You will get an error for 'Y' because it was never defined before trying to call it, only a lower case 'y' was previously defined.

There are rules to variable names that must be followed. It's one of the few things that MATLAB isn't that forgiving about. Here are the rules for naming variables:

- Capitalization matters.
 - 'x' and 'X' are two different variables.
- Variable names start with a letter.
 - If you start with a numerical value, MATLAB will not recognize it as a variable
 - You can use numbers as any other character.
 - '1A' will not work. 'A1' will work.
- Letters, numbers, and underscore ONLY
 - Other characters cause errors.
 - You can't use characters like #, @, \$, and other special characters.



Variable names

[Show Correct Answer](#)[Show Responses](#)

Multiple answers: Multiple answers are accepted for this question

Identify all of the BAD variable names:

A

passGoCollect\$200

B

Two_nums

C

2BR02B

D

Mult2

E

Is_this_Valid?

F	What_about_this_one
G	GClefSign
H	Or this_one

Be careful about overwriting built-in functions. MATLAB doesn't protect itself from that. For instance, `pi` is a built-in function in MATLAB that returns the floating-point number nearest the value of π . You can accidentally overwrite any built-in function though. If you assign a value to `pi` and store that in your workspace, the next time you run the command `pi`, MATLAB will not run the built-in function but rather reference the variable in your workspace.

Run the following commands to illustrate this point. You don't need to type the part after the percentage mark, that's just a comment to explain that command. I will explain comments when we get to m-files:

1. `X=pi*2`
2. `pi=2`
3. `X=pi*2`
4. `clear pi % To remove the overwritten 'pi'`
5. `X=pi*2`

Arrays

Arrays are a useful way to organize a series of data. In general, arrays will keep the same data type for all elements in it.

Creating Arrays

To create an array, you use square brackets and either a space or comma between the elements. Try running the command:

1. `X = [1 2 3 4]`
2. `Y= [5,6,7,8]`

MATLAB has some built-in functions for creating arrays as well.

Colon Operator

It uses a [colon operator](#) to generate Row vectors. Try the following commands and see if you see the descriptions.

- **D=[1:8]**
 - Generates all #s between 1 and 8, incrementing by 1 as integers.
- **E=[0:2:10]**
 - Generates all #s between 0 and 10, incrementing by 2 as integers.
- **E=[10:-0.5:3]**
 - Generates all #s between 10 to 3, incrementing by 0.5 as floats. Using the decimal format for the increment drives the format and using a negative value causes the array to increment in a decreasing order.
- **G=[0:pi/2:10]**
 - The end value doesn't have to be an exact multiple of the increment. It will use the first value to start but will stop generating and inputting numbers once they're over the maximum.

Linearly Spaced Vectors

Another means is using the command **linspace()**. These also generate row vectors but instead of controlling the incremental unit, it controls the spacing.

- **Z = linspace(-5,5,7)**
 - Generates 7 elements between -5 and 5, evenly spacing them between that min and max.

Colon operator controls the starting value and the spacing between elements.

Linspace() controls the minimum and maximum value and the number of elements between them.

Experiment a bit with COLON OPERATOR and Linspace(). They are incredibly useful so I want to ensure you understand them. Try some examples, read the MATLAB help if you need, and then describe how they work and what makes them difference in the discussion.



Match the command with what it does

Premise

1	Colon operator
---	----------------

**Response**

A	Controls the starting value and ending value as well as the spacing between elements
---	--

2	linspace
---	----------



B	Controls the starting value and maximum value as well as spacing between elements
---	---

C	Controls the starting value and ending value as well as number of elements
---	--

Calling Array values

To call an array value, you need to call the variable and include the index for the value you want. You use the variable name and then the index value. For instance, run the following commands.

1. `G=[0:pi/2:10]`
2. `G(1)`
3. `G(3)`
4. `G(1) + G(3)`

`G(1)` calls the first number from the array while `G(3)` calls the third one in the array. You can use these indexed array calls in any situation where you would call a scalar variable. This function by using the index of the array value to call it. In most other programming language, the index value starts at 0. In MATLAB, it starts at 1.

This is also a good opportunity to showcase the difference between a character array and a string. Run the following commands:

1. `C='Hello World'`
2. `S="Hello World"`
3. `C(1)`

4. **S(1)**
5. **C(2)**
6. **S(2)**

That last command, **S(2)**, should have given you an error because there is no second element in **S**. It's just the single scalar string. That wasn't the case for the character array since that is the same information as in the string just constructed from an array of characters instead.



MATLAB Arrays

[Show Correct Answer](#)

[Show Responses](#)

If you ran the command `RED = [6 : 2.3278 : 24]`, What is the value of `RED(6)`?

We will work on the creation and manipulation of matrices in future assignments. There are some used below just to really illustrate the differences between how MATLAB handles Array and matrix Math differently than you would have done so in your previous math courses.

Array Math

There are two types of math in Array Math; standard matrix math which you

may be familiar with and element by element which you are probably less familiar with but could probably figure it out by its name. Don't worry, I'll still go over it.

Matlab's mathematic abilities

• Traditional Matrix Math

Symbol	Description	Symbol	Description
*	Multiplication	/	Right Division
^	Exponentiation	\	Left Division

• Element by element Matrix Math

Symbol	Description	Symbol	Description
.*	Addition	.-	Subtraction
.*	Multiplication	./	Right Division
.^	Exponentiation	.\	Left Division

Note the **•** before each operator!!!

Matrix Math

This is the standard version you should have come across in your math classes. You multiply or divide a row of elements by a column of elements then sum them. This uses the basic `*` or `/` operator.

Element by Element

In element by element math, using the examples of X and Y , $Z = X + Y$ would result in $Z(1) = X(1) + Y(1)$, $Z(2) = X(2) + Y(2)$, ... $Z(n) = X(n) + Y(n)$. Each element is added/subtracted/multiplied/divided by its corresponding element. For addition and subtraction, MATLAB only uses element by element math. For multiplication, division, or exponentiation, you have to alter the standard symbol to be $*$, $./$, or $.^$. That additional period before the symbol lets MATLAB know you want to do element by element.

$$A = [A_1 \ A_2 \ \dots \ A_n] \quad B = [B_1 \ B_2 \ \dots \ B_n]$$

Match

$$A.*B = [A_1 * B_1 \ A_2 * B_2 \ \dots \ A_n * B_n]$$

You can see how this works by running the following and experimenting:

- $A = [1 \ 3 \ 5;$
 $2 \ 4 \ 7]; \% \text{ 2x3 matrix}$
- $B = [-5 \ 8 \ 11;$
 $3 \ 9 \ 21;$
 $4 \ 0 \ 8]; \% \text{ 3x3 matrix}$
- A/B

That uses traditional matrix math that requires # of columns of the first to match the number of rows of the second to create a matrix that has the same number of rows as the first and columns of the second. If you ran $A./B$ then you'd get an error because the rows and columns do not match.

- $C = 2; \% \text{ 1x1 Scalar}$
- $A*C$

The scalar, C , will automatically be treated as element by element by default, with or without the $.$.

- $A*A$
- $A.*A$

The first, 2x3 matrix times a 2x3 matrix, will fail because the number of rows do not equal number of columns. The second will perform using element by element math because they are the same size, with each corresponding element multiple across matrices.

- `D = [.5;2]; % 2x1 Array`
- `A.*D`

Also goes element by element but since the number of columns do not equal rows but there's only 1 column, it just takes that 1 column array D and applies the calculation across each individual column in the matrix A.

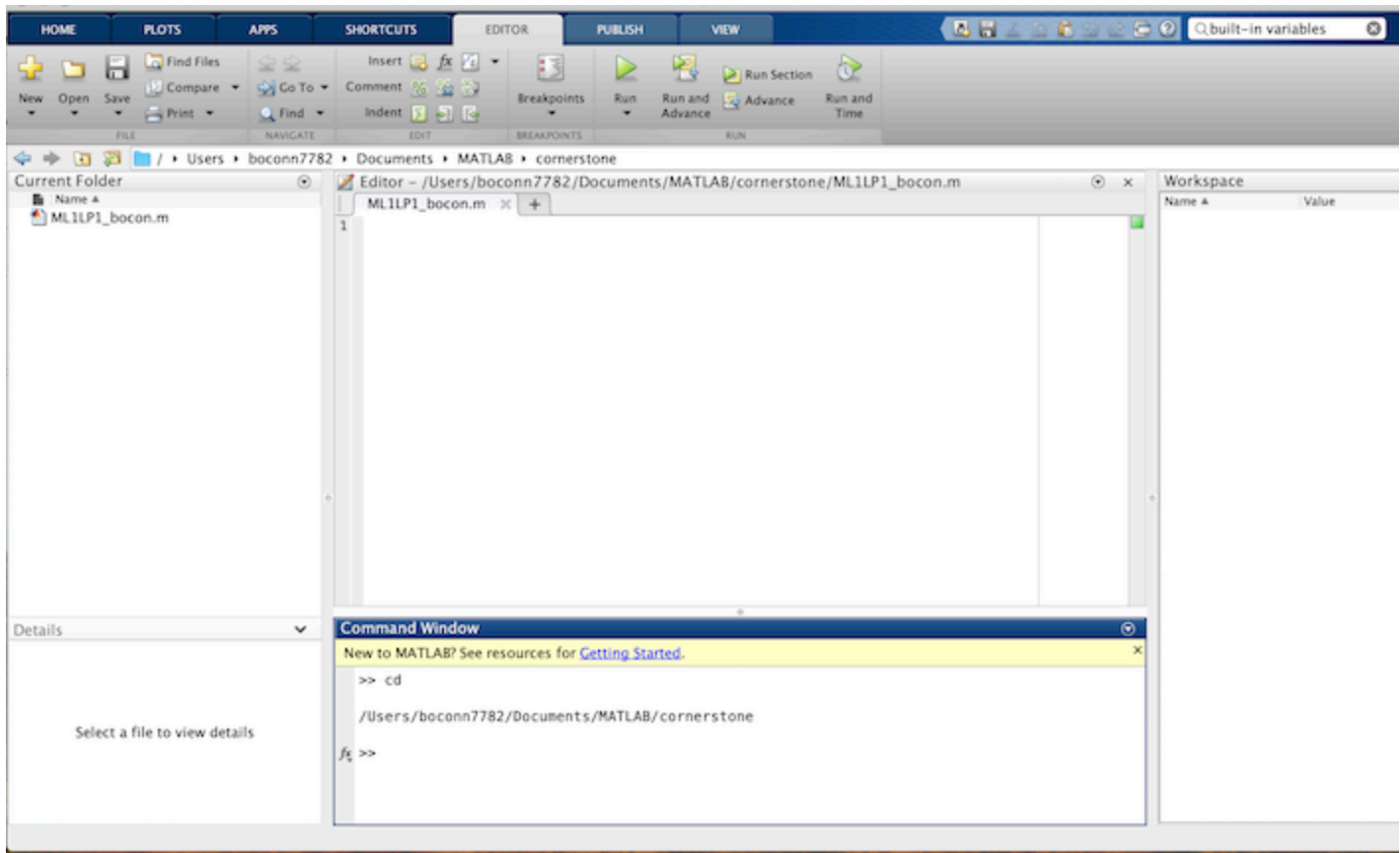
Try an assortment of combinations. Some will fail so read the warning to see why then try again. We will do more work with array math in class.

Programming in MATLAB

M-Files

M-files are just the standard file type for MATLAB scripts. We will create one soon for the deliverable of this pre-lab. When working just in the command window, you can run a few commands at a time before it becomes hard to manage. An m-file will allow you to run a full program and keep that code intact from session to session. Get into a habit of using the command window to test out your ideas, running a line or 2 of code to see what it will do, but do the majority of your actual work in save-able and easily repeatable full script kept as an m-file.

When you create an M-file, by default it's created in the current folder, on the left side of the UI and you can see its file path just below the Ribbon. To run a script using the Run command, it must be visible in that current folder for MATLAB to identify it. There are ways to call scripts in other folders but we won't get into that. You can use the command `cd` to display the current working directory. Your user interface should look similar to this:



Comments

This is a requirement for all of your assignments. Someone without any programming knowledge should be able to look at your program and at the very least know what you're trying to do. They may not understand how you're going based on the syntax of the code but should come across understandable comments throughout that lets them know what you are attempting to achieve with each block of code.

Comments start with a **%**. Anything on the line after the **%**, outside of any other function, will be ignored by MATLAB but still readable by a human observer. To comment out a block of code, use **% {** and **}%** to bookend the lines you would like commented out. You can also use **ctrl-/** or **command-/** to comment whatever line or group of lines you have selected.

Debugging

This is a topic we'll come to a lot but it is a skill. There are mountains of tips and tricks for debugging but there's one in particular that is pertinent with MATLAB. Its use of a command window for single line command input makes it very easy to try a few commands before including them in your final program. You do not have to have a complete program before you try running it.

You can run an incomplete program or copy paste a snippet into the command window to just run that. It'll help you figure out if it's a small problem in an isolated part of your program. You'll be amazed at how often one line will be the issue for an entire program.

There is also the workspace. It lets you see exactly what variables you're working with as well as what the stored information is in there. It's not uncommon for the issue to be not in the codes ability to run but that it's running perfectly and you're not working with the information or type of information you think you're working with.

M-File Benefits

How does the use of an M-file in MATLAB assist in the scripting and debugging process?


- A It allows running of a few commands at a time.
- B It helps maintain code from session to session.
- C It allows running of complete program immediately.
- D It displays all variables currently used.

 HINT



 EXPLANATION



 Show Responses

 Show Answer

Input/Output Commands

For your programs, you'll have to use a variety of input/output commands. These take information as well as present it from your running program to the Command Window.

;

[Adding a semi-colon](#) to the end of a line with suppress its command window output. Trying typing a few equations into the command window then try it again adding a semi-colon to the end.

Notice that the second time the command isn't repeated.

disp(X)

Displays an output in the command window. Whatever you input into the function, shown as **X** in **disp(X)**, will get displayed. You can either place a variable in there and it will display the content of that variable or you can directly enter a number or a string.

With functions that require inputs, you can enter them in the parenthesis. This function only requires one input but others will require more. They can even be a different amount depending on the circumstances. When there are multiple inputs, they're typically separated by commas.

input(X)

This function will display a string to prompt the user of required input and assigns that user to a variable. Whatever you put in the parenthesis will be the function input for the function Input(). I'm trying to be clear since the language can be confusing here. This function will print to the command window that string input.

Try running the following commands:

1. `prompt = 'What is the original value? ';`
2. `x = input(prompt)`
3. `y = x*10`

We start by storing the question into a variable, prompt. When we call input() we use 'prompt' as that function's input. It will then display the character array and then wait for the user to type a number as an input and hit return. It then loads that value into the variable x. If you want to ask the user for a string, you need to include a second input of the character 's' so that MATLAB knows to expect a string as an input instead of a number.

1. `z = input(prompt, 's');`

Demonstration of Input command

Video

Please visit the textbook on a web or mobile device to view video content.

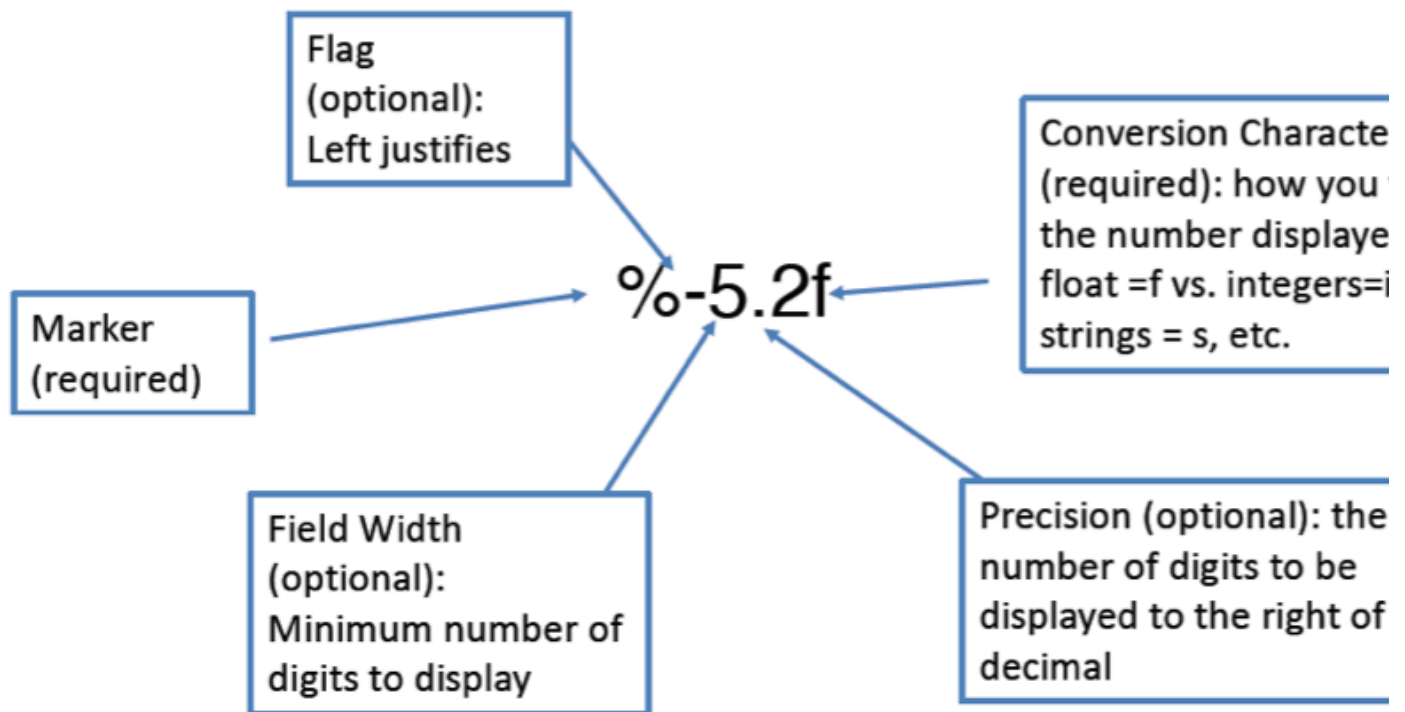
[fprintf\(formatSpec,A1,...,An\)](#)

This function is a more powerful form of the `disp()` function, but it is typically conceptually more difficult. If the below is not enough, try the MATLAB help documentation for the command. They go into far more detail and have more examples. We will also cover this in class likely a few times.

This command writes formatted data to a file. If you don't give it a file, it prints the information to the command window. We'll deal with printing to a file later, for now we'll just use it as a print to command window tool.

formatSpec is a combination of ordinary text and special formatting operators. It prints that information with each of the formatting operators replaced with **A1, ..., An** in order of their appearance. When replaced the operators in the text, how **A1, ..., An** displays is defined by the information in the formatting operators.

formatting operators always start with % symbol. MATLAB describes how each of the following characters much better than I can and in more detail here



This is the basics. You have the characters in this order

1. Marker `%`
2. Justification Flag
3. Field Width – How many spaces is this going to take up?
4. Precision – To how many significant digits should we display?
5. Conversion – What data type format should it be in?

Here's an example that uses a Float, an Integer, and a String:

1. `x = 2.582489;`
2. `y = 3.000;`
3. `names = "Peter, Paul, and Mary";`
4. `fprintf('The average family has %-9.3f children. Mine has %7d. Their names are %s. \n', x, y, names);`

If you try running it with `..., x, x, names);` instead of `..., x, y, names);`, it'll print the scientific format of `x` instead of rounding it to the nearest integer. It is a known issue but has yet to be fixed so if you want to print an integer, you'll have to handle it as such or convert it separately. Also take note of how much space the variables take up in the output. I specifically chose 9 characters and 7 characters to illustrate the point. Try running it again with smaller field widths.

One last point about `fprintf()` is that, unlike `disp()`, it does not automatically include a new line after printing. Include a `'\n'` in your text to include a newline character in the print out.

Video

Please visit the textbook on a web or mobile device to view video content.



fprintf specifications

[Show Correct Answer](#)

[Show Responses](#)

What command would be behind the orange box to get the shown output (Hint: It starts with the formatting operator, not a space.):

Displaying Data

The most common plot used by engineers is likely the simple 2-dimensional XY plot. MATLAB handles this very easily with the command `plot(X,Y)` where Y is the dependent variable along the vertical axis and X is the independent variable along the horizontal. It does not plot an equation for you, it plots a series of points and connects them. Try running the command:

1. `x = 0:pi/100:2*pi; % Notice colon operator use to quickly create a range of values to use for calculation`

2. `y1 = sin(x);`
3. `plot(x,y1)`

You created two separate arrays of the same size then called plot with them. It plots a point, using an XY Coordinate system, at `(x(1),y1(1))`, then at `(x(2),y2(2))`, and then it connects them. It does this for all 31 points. If the arrays were not of the same size, it would give you an error and not plot.

Try running the following:

1. `y2 = sin(x-0.25);`
2. `y3 = sin(x-0.5);`
3. `plot(x,y1,x,y2,x,y3)`

This plots three sets of lines. When you give multiple sets of X and Y, using the command as `plot(X1,Y1,X2,Y2,...Xn,Yn)`, it will plot multiple times on the same figure. You can even include line specifications in the command, `plot(X1,Y1,LineStyle1,...,Xn,Yn,LineStylen)`, to specify how you want the lines to look. The **linespecs** involve enough options that I will leave you to utilizing MATLAB help center to learn more about them and see some examples.

You can also modify the plot afterwards by adding a legend, labels, and titles. Enter the following commands, assuming the figure is still open. If not, rerun the last plot command. I figure you can infer what they do by their function names and what you see happen in the figure. If not, you can always use the doc command.

1. `legend('line 1','line 2','line 3');`
2. `xlabel('My X Label');`
3. `ylabel('My Y Label');`
4. `title('Example Graph');`

Here's some useful other command when working with plots.

- [hold on](#) or [hold off](#)
 - MATLAB overwrites the figure window every time. When hold is on, it does not do this, just drawing the plot over previous plots.
- [figure\(n\)](#)
 - You can change what figure you're working on. By default, MATLAB has you working on Figure 1 but you can be working on multiple plots.
- [clf](#)

- This clears the currently active figure window. It leaves it open though but just clears all graphical and label information displayed.
- [close](#)
 - This closes all currently open figures.



plot linespecs

[Show Correct Answer](#)

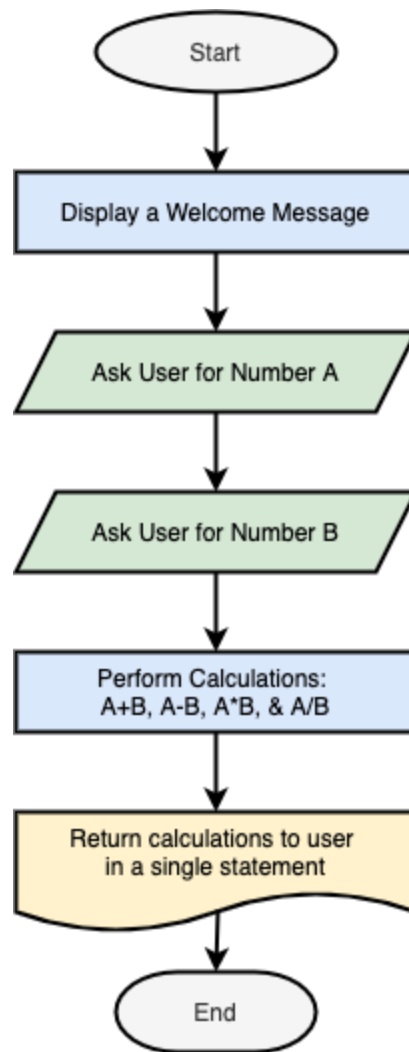
[Show Responses](#)

What is the linespec input for a green dashed line with diamond markers?

This one is intentionally not supported in the pre-lab to force you to use the MATLAB help center to a greater extent. Start by looking through the help center entry for 'plot' and scroll through the documentation. Make sure to use the short name symbols.

P5L1 - Simple Calculator

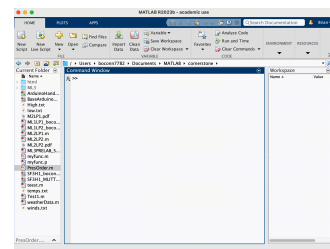
For your first MATLAB program, you will be writing a simple calculator that meets the requirements of the provided flow chart:



You are welcome to and challenged to try and create this from scratch. There is minimal guidance in the instruction below. There is a more detailed video walkthrough available as well.

Start a new m-file

- Select **New Script** from the **Home** ribbon across the top of the MATLAB UI
- Save and name your script using the class naming convention
 - **P5L1_ ...**
 - **AS ALWAYS SAVE EARLY AND SAVE OFTEN**



NOTE: These gifs are a new thing I'm trying. This seems to run at a slower speed than the actual video. I will add some survey questions later about these after I've used them in a few

assignments. Feel free to give me feedback before hand though and I'll try different things, speeds, lengths, sizes, etc before formally surveying everyone for feedback.

- Put in a title block

```
% MATLAB Lab #1 Lesson Part 1
```

```
% <First Initial>.<Last Name>
```

```
% <Date of creation>
```

```
% <Name of Script – You have some freedom to decide this>
```

```
% <Description of what the program will do. You're welcome to...
```

```
% come back to this after going through the rest of this and edit it>
```

- Add the clean up commands to the top of your code
 - These ensure that you start with a clean command window and workspace everytime you run your script
 - `clc; clear;`

Use comments to scaffold your code.

So the logical process is already laid out in your requirements and the

Let's break that down into a pseudocode and even use that as our comments in the code. We want to:

1. **PROGRAM P5L1:**
2. **Display a welcome message**
3. **Ask the user for Number A**
4. **Ask the user for Number B**
 1. (We don't have to do any error checking here because we haven't been asked to do that. Don't overthink it then, just ask the user for the calculation inputs.)
5. **Perform Calculations: +, -, *, /**
6. **Return calculations in a single statement**
 1. We've even been given an example below (See the **Example Command Window Output:**)
7. **END**

We can put that directly into our code as comments and meet that requirement:

1. `% Display a welcome message`

2. % Ask the user for Number A
3. % Ask the user for Number B
4. % Perform Calculations
5. % Return calculations in a single statement

Start filling in the code

We can start filling in the code where we can. We want to display a welcome message so we need to print a welcome message to the command window. We have a function for that. We need to take some user inputs and we have a function for that too. We also need to store that information so we'll have to create some variables and we should make sure they're named in a useful way too. So let's give the hard part a go since nothing says we have to complete this code in order:

1. % Display a welcome message
2. % Ask the user for Number A
3. `A = input("What is the value to be used for A?");`
4. % Ask the user for Ask the user for Number B
5. < Repeat for Number B >
6. % Perform Calculations
7. % Return calculations in a single statement

The video demo below walks through creating this program in detail including some debugging practices. I put it at the end though so you can take a short time to challenging yourself by trying this on your own for a bit before following along with the video though. Set a short timer to keep yourself from spending too much time before switching to the video though. This is a learning exercise so you want to stay in a zone of productive struggle rather than prolonged frustration.

Finish your code

So, we've handled the input of information. All that's left is some math and displaying it.

% Display a welcome message

Include a simple message at the beginning of the program telling the user what the program is supposed to do. This will be a static statement that doesn't deal with any special formatting so `disp()` is fine here.

% Perform Calculations

You'll need to perform those 4 calculations and therefore need to store the results of each. Therefore you'll need to establish variables to store the result of the calculations and use the basic MATLAB math syntax to perform those calculations: **+**, **-**, *****, & **/**.

% Return calculations in a single statement

This means you want a single command that provides you a WELL FORMATTED output. See the example for more details below. So pick a command that will allow you to **FORMAT** how you **PRINT** out the information. This is admittedly the hardest part conceptually. Challenge yourself but don't wait too long before referring to the videos.

Requirements:

- Use all of the input/output commands where appropriate:
 - **disp()**
 - **input()**
 - **fprintf()**
- Display requirements:
 - display the sum and difference as integers
 - display the product and dividend as floats
- Use 2 integers as your inputs when you run the script.
 - Floats will cause some funky outputs. We'll talk about those in class, but go ahead and try it to see what happens with the integer formats.

Reminders:

- If you're using single quotes to indicate text then use a double apostrophe to show possession: Bob's Hat --> 'Bob's Hat'
- Make sure you suppress the echo (Where it prints the result of the last command) with semicolons!
- You can use **doc fprintf** or MATLAB Help search bar for more options!
 - **'\n'** will add a new line to the output and the example output uses tabs for organizing the outputs. How would you incorporate those to your output?

This will help with your debugging and make it easier for the TA to run your program and check it.

Example Command Window Output:

This is just an example. Your

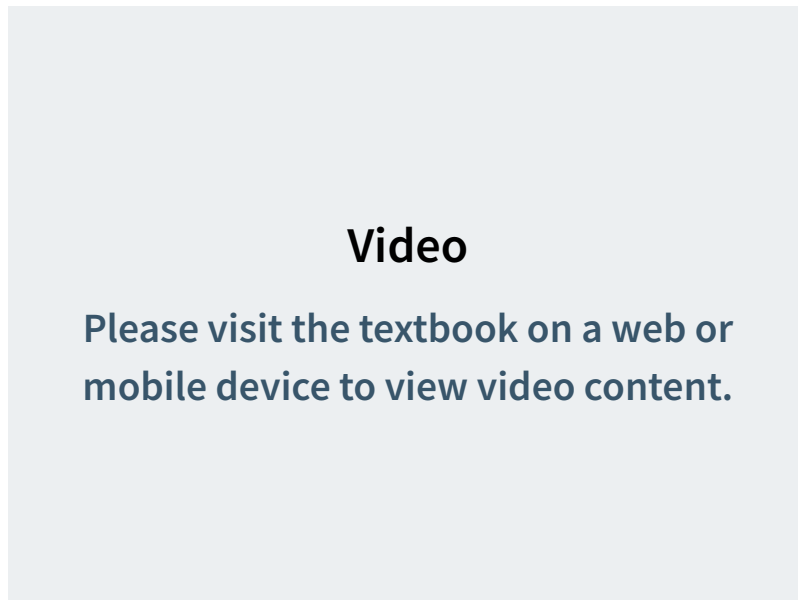
display output
does not have to
be arranged
exactly as shown,
just provide

```
----- Simple Calculator -----
This program takes in 2 values and
performs the following calculations:
    A+B    A-B
    A*B    A/B
What is the value to be used for A? 23
What is the value to be used for B? 65
The calculations are:
    A+B= 88      A-B=-42
    A*B=1495.00  A/B= 0.35

fx >>
```

generally the same information and layout. For instance, you don't need to match the spacing of the output exactly for the calculations, like it doesn't have to account for really long numbers or anything like that. Just some tabs between outputs.

Video Demo/Walkthrough



Video

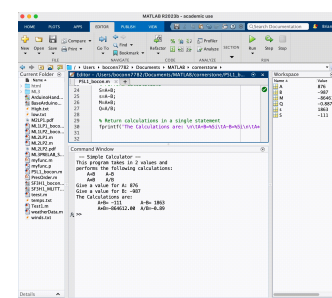
Please visit the textbook on a web or
mobile device to view video content.

End of Part 1

Run your code once using any 2 integer inputs.

Preparing your submission

Since this is your
first MATLAB
assignment, let's
talk about
preparing your
submission. For



MATLAB assignments, you need the raw M-file and then a pdf of your command window output. You've already created the M-File.