WEEK 1: INTRODUCTION TO MATLAB 🎉

OBJECTIVES:

- Get familiar with MATLAB's interface and basic commands.
- Learn how to think like a programmer (algorithmic thinking 🧠).
- Start creating your own mini-programs to solve cool problems!

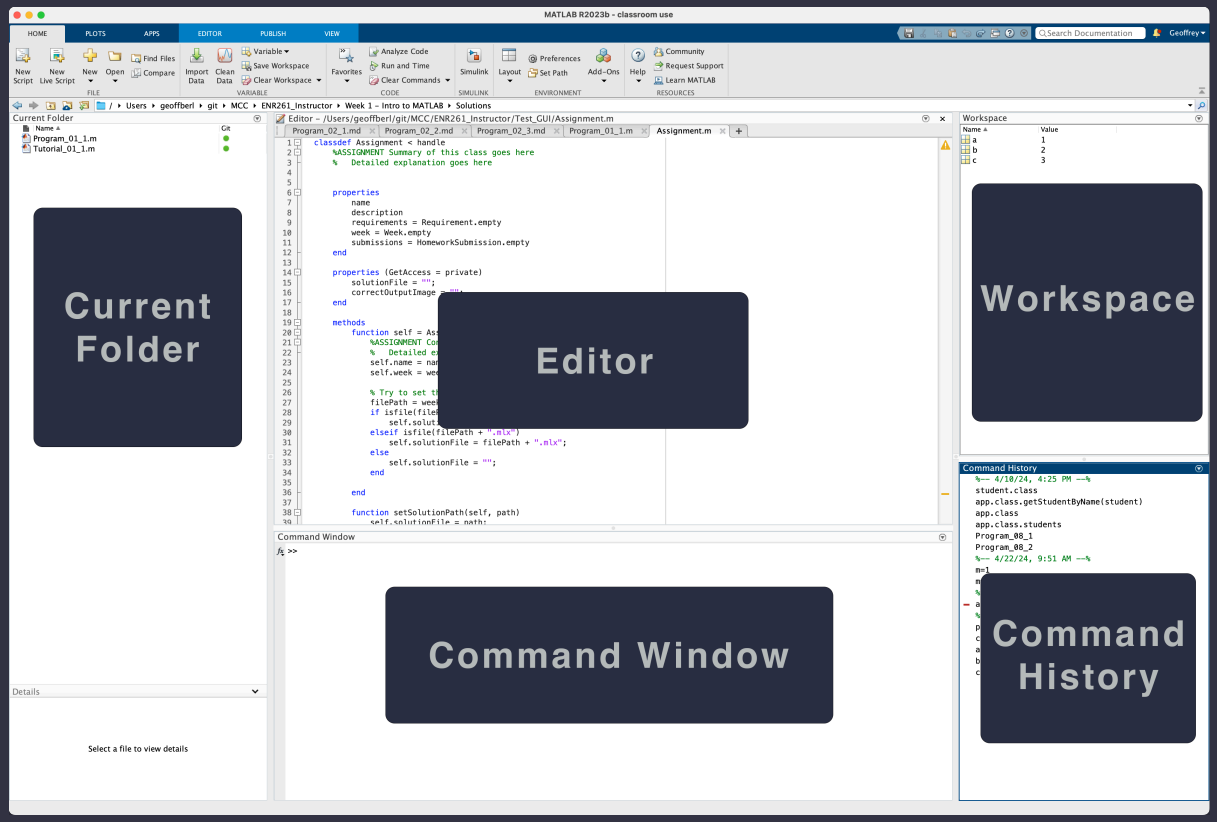
INTRODUCTION TO THE MATLAB INTERFACE

WHAT IS MATLAB?

- MATLAB was originally developed for matrix calculations.
- It is useful to engineers and scientists for data analysis and solving complex problems.

THE MATLAB DESKTOP INTERFACE

- **Command Window**: Where you type commands and see immediate results.
- **Workspace**: Displays variables you've created during your session.
- **Command History**: Keeps a record of the commands you've entered.
- **Editor**: Used for writing scripts (M-files) and functions.
- **Current Folder**: The current working directory (`cwd`).



INTERACTIVE DEMONSTRATION:

- Open MATLAB and take a tour of the interface.
- Feel free to open MATLAB and follow along.

HANDY SHORTCUTS ⌨️

- Command Window history: ↑ / ↓
- Auto-complete: tab
- Stop execution: ctrl + c
- Editor: Run section or file via toolbar (or *Run* button)
- Comment/Uncomment selection (Editor toolbar)

CREATING A SCRIPT (.M) 📝

1. In MATLAB, click "New Script" (Home tab → New → Script)
2. Type your code in the Editor; add comments with `%`
3. Save as `my_script.m` (avoid names that shadow functions, e.g., `mean.m`)
4. Run via the green Run button (or right-click → Run)
5. Keep the script in the Current Folder or on the MATLAB path

Tips:

- End lines with `;` to suppress output
- Use code sections with `%%` to run blocks independently
- Scripts share the base workspace; functions have their own workspace

EXAMPLE SCRIPT: HELLO_WORLD.M

```
% HELLO_WORLD  Demo script that creates data and plots it
clc; clear; close all;

%% Parameters
t = 0:0.1:10;           % time vector
A = 2; f = 0.5;        % amplitude and frequency

%% Compute signal
y = A * sin(2*pi*f*t);

%% Plot
plot(t, y, 'LineWidth', 1.5); grid on;
xlabel('Time (s)'); ylabel('Amplitude');
title('Sine Wave');
```

Common errors to avoid:

- Script name conflicts (e.g., `plot.m` will break the built-in `plot`)
- Saving the script outside the Current Folder (not on path)
- Accidental workspace reuse; if behavior seems odd, try `clear` and re-run

THE COMMAND WINDOW 💻

USED FOR REAL-TIME INTERACTION:

- MATLAB allows you to interact directly with your data. For example, typing `2 + 3` and pressing `enter` will immediately return `ans = 5`.

COMMON COMMANDS

COMMON COMMANDS

`clc`: Clears the command window, but doesn't clear the workspace.

COMMON COMMANDS

`clc`: Clears the command window, but doesn't clear the workspace.

`clear`: Clears all variables from the workspace.

COMMON COMMANDS

`clc`: Clears the command window, but doesn't clear the workspace.

`clear`: Clears all variables from the workspace.

`whos` (`who`): Displays detailed information about variables in the workspace.

COMMON COMMANDS

`clc`: Clears the command window, but doesn't clear the workspace.

`clear`: Clears all variables from the workspace.

`whos` (`who`): Displays detailed information about variables in the workspace.

💡 Tip:

`clc` → `Cl`ear `C`ommand Window

`clear` → `Cl`ear `E`nvironment

INTERACTIVE EXERCISE:

1. Open MATLAB.
2. Try the following commands and observe:

```
2 + 3
clc
whos
```

INTERACTIVE EXERCISE:

1. Open MATLAB.
2. Try the following commands and observe:

```
2 + 3
clc
whos
```

3. What happened after each command?

💡 Tip: More on Commands

- Use ↑ to cycle through previously executed commands. (works better without the history window)
- Double-click or drag and drop from the command history window (may need to be enabled).
- Start typing a previously used command and use ↑ to filter
- Start typing a known command and use tab to see possible solutions
- ctrl + c: Stops the execution of a command or script if it runs too long or seems to be stuck.

DEBUGGING BASICS 🐞

- Set a breakpoint: click the left gutter next to a line in the Editor
- Run your script: execution stops at the breakpoint
- Step Over / Step In / Continue using the Debug toolbar
- Inspect variables in the Workspace; hover in the Editor to see values
- Stop debugging or remove breakpoints when done

Tip: Add a temporary `disp()` to print variables while you iterate.

ANALOGY: DEBUGGING 🎬

- Breakpoints are pause points in a video
- Step Over = advance one frame; Step In = open the scene (function)
- Workspace is your on-screen overlay showing live values

LEARNING AND EXECUTING SIMPLE COMMANDS

COMPUTATION NOTATION:

- MATLAB uses standard mathematical operators: `+`, `-`, `*`, `/`, `^`.
- Try to guess the output for the following:

LEARNING AND EXECUTING SIMPLE COMMANDS

COMPUTATION NOTATION:

- MATLAB uses standard mathematical operators: `+`, `-`, `*`, `/`, `^`.
- Try to guess the output for the following:

```
3 - 2
```

LEARNING AND EXECUTING SIMPLE COMMANDS

COMPUTATION NOTATION:

- MATLAB uses standard mathematical operators: `+`, `-`, `*`, `/`, `^`.
- Try to guess the output for the following:

```
3 - 2
```

```
3 * 2
```

LEARNING AND EXECUTING SIMPLE COMMANDS

COMPUTATION NOTATION:

- MATLAB uses standard mathematical operators: `+`, `-`, `*`, `/`, `^`.
- Try to guess the output for the following:

```
3 - 2
```

```
3 * 2
```

```
3 / 2
```

LEARNING AND EXECUTING SIMPLE COMMANDS

COMPUTATION NOTATION:

- MATLAB uses standard mathematical operators: `+`, `-`, `*`, `/`, `^`.
- Try to guess the output for the following:

```
3 - 2
```

```
3 * 2
```

```
3 / 2
```

```
3 ^ 2
```

LEARNING AND EXECUTING SIMPLE COMMANDS

COMPUTATION NOTATION:

- MATLAB uses standard mathematical operators: `+`, `-`, `*`, `/`, `^`.
- Try to guess the output for the following:

```
3 - 2
```

```
3 * 2
```

```
3 / 2
```

```
3 ^ 2
```

```
1 + inf
```

VARIABLES 🥣

WHAT ARE VARIABLES?

- Variables are like labeled jars: the *label* is the variable name and the *contents* are the value.
- Replacing the contents does not change the label (name stays the same).
- The Workspace is your pantry where all the jars (variables) live. Clearing the workspace empties the pantry.

VARIABLES 🥣

WHAT ARE VARIABLES?

- Variables are like labeled jars: the *label* is the variable name and the *contents* are the value.
- Replacing the contents does not change the label (name stays the same).
- The Workspace is your pantry where all the jars (variables) live. Clearing the workspace empties the pantry.

TYPE:

- Each jar has a content type (number, text, table, etc.).
- MATLAB is dynamically typed: you don't declare types ahead of time.

VARIABLES 🥣

WHAT ARE VARIABLES?

- Variables are like labeled jars: the *label* is the variable name and the *contents* are the value.
- Replacing the contents does not change the label (name stays the same).
- The Workspace is your pantry where all the jars (variables) live. Clearing the workspace empties the pantry.

TYPE:

- Each jar has a content type (number, text, table, etc.).
- MATLAB is dynamically typed: you don't declare types ahead of time.

VALUE:

- The value is what's inside the jar; you can change it anytime.
- New values can even change the content type (e.g., from scalar to vector).

VARIABLES 🥣

WHAT ARE VARIABLES?

- Variables are like labeled jars: the *label* is the variable name and the *contents* are the value.
- Replacing the contents does not change the label (name stays the same).
- The Workspace is your pantry where all the jars (variables) live. Clearing the workspace empties the pantry.

TYPE:

- Each jar has a content type (number, text, table, etc.).
- MATLAB is dynamically typed: you don't declare types ahead of time.

VALUE:

- The value is what's inside the jar; you can change it anytime.
- New values can even change the content type (e.g., from scalar to vector).

"This jar is a **double**, and it currently contains **42**."

STANDARD NAMING CONVENTIONS

- `camelCase`
- `snake_case`
- `ALL_CAPS`, `SCREAMING_SNAKE_CASE`

STANDARD NAMING CONVENTIONS

- `camelCase`
- `snake_case`
- `ALL_CAPS`, `SCREAMING_SNAKE_CASE`

Example:

```
MAX_FRUIT_ALLOWED = 15
numApples = 5
numOranges = 10
fruitCount = a + b
withinLimit = fruitCount <= MAX_FRUIT_ALLOWED
```

SCRIPT VS FUNCTION 👩‍🍳⚙️

- Script = Recipe in the main kitchen (base workspace)
  - Uses whatever is on the counter unless you clean first (`clear`)
  - No inputs/outputs by signature; affects current Workspace
- Function = Kitchen appliance with its own chamber (private workspace)
  - Put inputs in, get outputs out; does not see your counters
  - Safer, reusable, testable

IMPORTANT CONCEPTS:

- **Case Sensitivity**: `Variable` and `variable` are different.
- **Naming Conventions**: Stick to clear, descriptive names to make your code easier to understand.

RESERVED WORDS/VARIABLES:

- MATLAB has predefined variables, known as constants, like `pi` and functions like `inf`.
- MATLAB also has predefined functions, such as `sin`, `cos`, `clear`, `clc`, `sqrt`, etc.

Warning! Avoid using reserved names for your variables when possible.

ANALOGY: MATRICES & TABLES 📊

- Matrix = Excel grid (rows × columns). Example: `A(2:4,1:3)` ↔ cells A2:C4
- Table = Spreadsheet with headers (named columns) and typed values
- Add a column = add a new header; add a row = add a new record
- Use tables when names/units matter; use matrices for pure numeric work

ALGORITHMS

WHAT IS AN ALGORITHM?

ALGORITHMS

WHAT IS AN ALGORITHM?

Put simply, an algorithm is a set of instructions to complete a task or solve a problem.

ALGORITHM DEVELOPMENT PROCESS 🗺️

1. **Analyze the problem** – Just like planning a trip, figure out where you want to go.
2. **Define inputs** – Pack your bags! What do you need to go on your trip?
3. **Perform Manipulations** – Map out your route, how are you going to get there?
4. **Produce Output(s)** – Arrive at your destination!

ALGORITHM DEVELOPMENT PROCESS 🗺️

1. **Analyze the problem** – Just like planning a trip, figure out where you want to go.
2. **Define inputs** – Pack your bags! What do you need to go on your trip?
3. **Perform Manipulations** – Map out your route, how are you going to get there?
4. **Produce Output(s)** – Arrive at your destination!

BEST PRACTICES:

- **Commenting**: Use comments (%) to explain your code. This helps others (and future you) understand what your code does.
- **Incremental Development**: Write and test small pieces of code before integrating them into a larger program.

PROBLEM SOLVING EXAMPLE: VOLUME OF A CONE

APPLYING WHAT WE'VE LEARNED:

1. **Analyze the problem**: Explain the formula $v = \frac{\pi r^2 h}{3}$
2. **Define inputs** (in this case variables):

PROBLEM SOLVING EXAMPLE: VOLUME OF A CONE

APPLYING WHAT WE'VE LEARNED:

1. **Analyze the problem**: Explain the formula $v = \frac{\pi r^2 h}{3}$
2. **Define inputs** (in this case variables):

```
radius = 6
height = 12
```

Note: *We don't need* `pi` *as that's provided for us*

PROBLEM SOLVING EXAMPLE: VOLUME OF A CONE

APPLYING WHAT WE'VE LEARNED:

1. **Analyze the problem**: Explain the formula $v = \frac{\pi r^2 h}{3}$
2. **Define inputs** (in this case variables):

```
radius = 6
height = 12
```

Note: *We don't need* `pi` *as that's provided for us*

3. **Perform the manipulation**:

```
volume = (pi * radius^2 * height) / 3
```

PROBLEM SOLVING EXAMPLE: VOLUME OF A CONE

APPLYING WHAT WE'VE LEARNED:

1. **Analyze the problem**: Explain the formula $v = \frac{\pi r^2 h}{3}$
2. **Define inputs** (in this case variables):

```
radius = 6
height = 12
```

   Note: *We don't need* `pi` *as that's provided for us*

3. **Perform the manipulation**:

```
volume = (pi * radius^2 * height) / 3
```

4. **Produce Output/Result**:

   In this case, we are simply outputting to `volume` and the console (done by MATLAB)

INTERACTIVE EXERCISE:

LET'S COMPUTE THE VOLUME OF A CONE:

INTERACTIVE EXERCISE:

LET'S COMPUTE THE VOLUME OF A CONE:

```
radius = 7
```

```
radius =

        7
```

INTERACTIVE EXERCISE:

LET'S COMPUTE THE VOLUME OF A CONE:

```
radius = 7
```

```
radius =

        7
```

```
height = 10
```

```
height =

        10
```

INTERACTIVE EXERCISE:

LET'S COMPUTE THE VOLUME OF A CONE:

```
radius = 7
```

```
radius =

        7
```

```
height = 10
```

```
height =

        10
```

```
volume = (pi * radius^2 * height) / 3
```

```
volume =

        513.1268
```

INTRODUCTION TO M-FILES

WHAT ARE M-FILES?

- M-files are scripts or functions you can save and reuse. They help organize code and are essential for more complex projects.

USED FOR:

- Saving your programs (scripts).
- Executing a list of commands.
- Saving your work while you work through a problem.
- Re-opening and modifying your program at any time.

CREATING A BASIC M-FILE:

1. Open the MATLAB Editor.
2. Choose "New Script" from the menu
3. Type the following:

```
% Compute and display the volume of a cone
radius = 6
height = 12
volume = (pi * radius^2 * height) / 3
```

CREATING A BASIC M-FILE:

1. Open the MATLAB Editor.
2. Choose "New Script" from the menu
3. Type the following:

```
% Compute and display the volume of a cone
radius = 6
height = 12
volume = (pi * radius^2 * height) / 3
```

4. Save as `volume_of_cone.m`.

CREATING A BASIC M-FILE:

1. Open the MATLAB Editor.
2. Choose "New Script" from the menu
3. Type the following:

```
% Compute and display the volume of a cone
radius = 6
height = 12
volume = (pi * radius^2 * height) / 3
```

4. Save as `volume_of_cone.m`.
5. Run the script by typing `volume_of_cone` in the Command Window.

MORE ON SCRIPTS:

- To run a MATLAB script call the filename without `.m`
- The script must be in the current working directory `cwd`
- Alternatively, you can provide a path (`C:\my\script\location.m`)

MORE ON SCRIPTS:

- To run a MATLAB script call the filename without `.m`
- The script must be in the current working directory `cwd`
- Alternatively, you can provide a path (`C:\my\script\location.m`)

OUTPUT SUPPRESSION:

- MATLAB, by default, prints command results to the command window.
- When someone asks for the volume of a cone, they probably don't want to see the radius, and height displayed.
- We can use the semicolon `;` operator to *suppress* output.

UPDATING OUR SCRIPT

- We'll add some semicolons to suppress the output of redundant values.

```
% Compute and display the volume of a cone
radius = 6;
height = 12;
volume = (pi * radius^2 * height) / 3
```

SUPPRESSING OUTPUT & DISPLAYING RESULTS

In MATLAB, a semicolon `;` at the end of a line suppresses output. But what if you want to display something?

- **disp(variable)**: A quick way to show the value of `variable` in the Command Window.
- **fprintf(format, variables...)**: For formatted output, similar to `printf` in C.

```
% Example of using disp vs. fprintf

radius = 6;
height = 12;
volume = (pi * radius^2 * height) / 3;  % Volume of a cone

% Quick display:
disp(volume)

% Formatted display:
fprintf('The volume of the cone is %.2f cubic units.\n', v
```

CREATING QUALITY WORK:

- Write clear and understandable code.
- Add comments to document your code.
- Include a set of header comments with the program's objective, your name, anyone who assisted you.

CREATING QUALITY WORK:

- Write clear and understandable code.
- Add comments to document your code.
- Include a set of header comments with the program's objective, your name, anyone who assisted you.

TIP! Write your comments first, then write code to honor your comments.

EXAMPLE PROGRAM WITH EXPECTED FORMAT

```
% Compute and display the volume of a cone
%
% Filename: Volume_of_cone
% Developer: Your Name
% Assisted By:
% Date: 08/04/2024

% Always start with a clean output and workspace
clc
clear

% Declare variables
radius = 6; % Radius of the cone (inches)
height = 12; % Height of the cone (inches)

% Compute the volume of a cone
volume = (pi * radius^2 * height) / 3;
```

ERROR HANDLING AND DEBUGGING 🕵️

BE THE DETECTIVE!

- **Syntax Errors**: Oops, you mistyped something! Let's fix it.
- **Runtime Errors**: Something went wrong during the run – let's figure out why!

INTERACTIVE EXERCISE:

Run `clear` and introduce an error by changing `radius` to `radiuss` in the M-file and let's observe the error message.

INTERACTIVE EXERCISE:

Run `clear` and introduce an error by changing `radius` to `radiuss` in the M-file and let's observe the error message.

TIP! Start fresh to avoid confusion.

- Whenever you run into a problem like this it is a good idea to `clear` your workspace of any possible bad values.
- You should include `clear`, and sometimes `clc` in your script to ensure you are starting fresh.

USING THE HELP SYSTEM

ACCESSING MATLAB HELP:

- **In Command Window**: Type `help` enter to get information on a specific command.
- Right-click on a function and select "Help on `functionName`".
- **Help Browser**: Access via the  icon.

Example:

- Type `help sin` to explore how MATLAB handles trigonometric functions.

GENERAL SHORTCUT TIPS

KEYBOARD NAVIGATION:

- home and end keys jump to the beginning and end of the line (cmd + ← / → on Mac).
- Arrow keys move a single character (← / →) or row (↑ / ↓).
- ctrl + ← / → jumps by word (option + ← / → on Mac).
- Holding shift while moving the cursor (using any of the aforementioned shortcuts), selects text.
- ctrl + c to copy, or ctrl + x to cut, ctrl + v to paste. (cmd + c, cmd + x, cmd + v on Mac)
- Double-click highlights a word, triple-click highlights a row.

LECTURE RECAP

- Variables are used for storing data for re-use.
- Use clear, and informative names.
    - Others and future-you will thank you
- M-files allow you to save and organize your work.
- The command window is immediate and can allow you to troubleshoot, or incrementally develop your code.
- Use Help whenever you're unsure of a command's syntax or usage.
- **Always**:
    - Write clear, well-documented code.
    - Include **comments**. (This is your documentation)
    - **Suppress** what doesn't need to be displayed.
    - Start with a `clear` workspace and `clc`d command window.
    - Learn keyboard shortcuts for efficient coding.
    - Save often, commit often.

GOTCHAS

- Variables **are** case sensitive, do not try to use this to your advantage.
- Don't forget `;` to suppress undesirable output.
- Be careful not to overwrite (mask) reserved variables/names.
- Copy and paste is your frienemy.

ANALOGIES:

- **MATLAB Interface**: Think of MATLAB as a sophisticated calculator with a built-in notebook where you can save your calculations and notes.
- **Command Window**: Imagine MATLAB M-files (scripts) as a solution to a math problem. The Command Window is like your calculator where you check your values and help you develop your scripts.
- **Variables**: Think of variables as labeled containers. You store information in them, which you can use later.
- **M-Files**: M-files are like recipes; they list the steps needed to perform a task. You can save them, modify them, and share them with others.

SOFTWARE ENGINEERING EXAMPLE