Lesson 1: Introduction to the MATLAB Environment

MATLAB is a programming platform designed specifically for engineers, scientists, and applied mathematicians.

The MATLAB application focuses on providing a specialized language* for scientific and technical computing.

The matrix-based MATLAB language is intended to provide several advantages to users who want to develop, analyze, or improve mathematical models of real-world phenomena that require computation.

* Remark: when we say that MATLAB is a specialized language we mean that it's major purpose for existence is to support numerical computation. In contrast, a general purpose programming language is designed to write software in a much wider variety of domains. Examples of general purpose languages include C, C++, Python,

- The name MATLAB is a portmanteau (a blend of words) in which the first part of the word MATrix is combined with the first syllable of the word LABoratory.

- In otherwords, you can think of the name MATLAB using the following formula:

MATLAB = MATrix LAB oratory

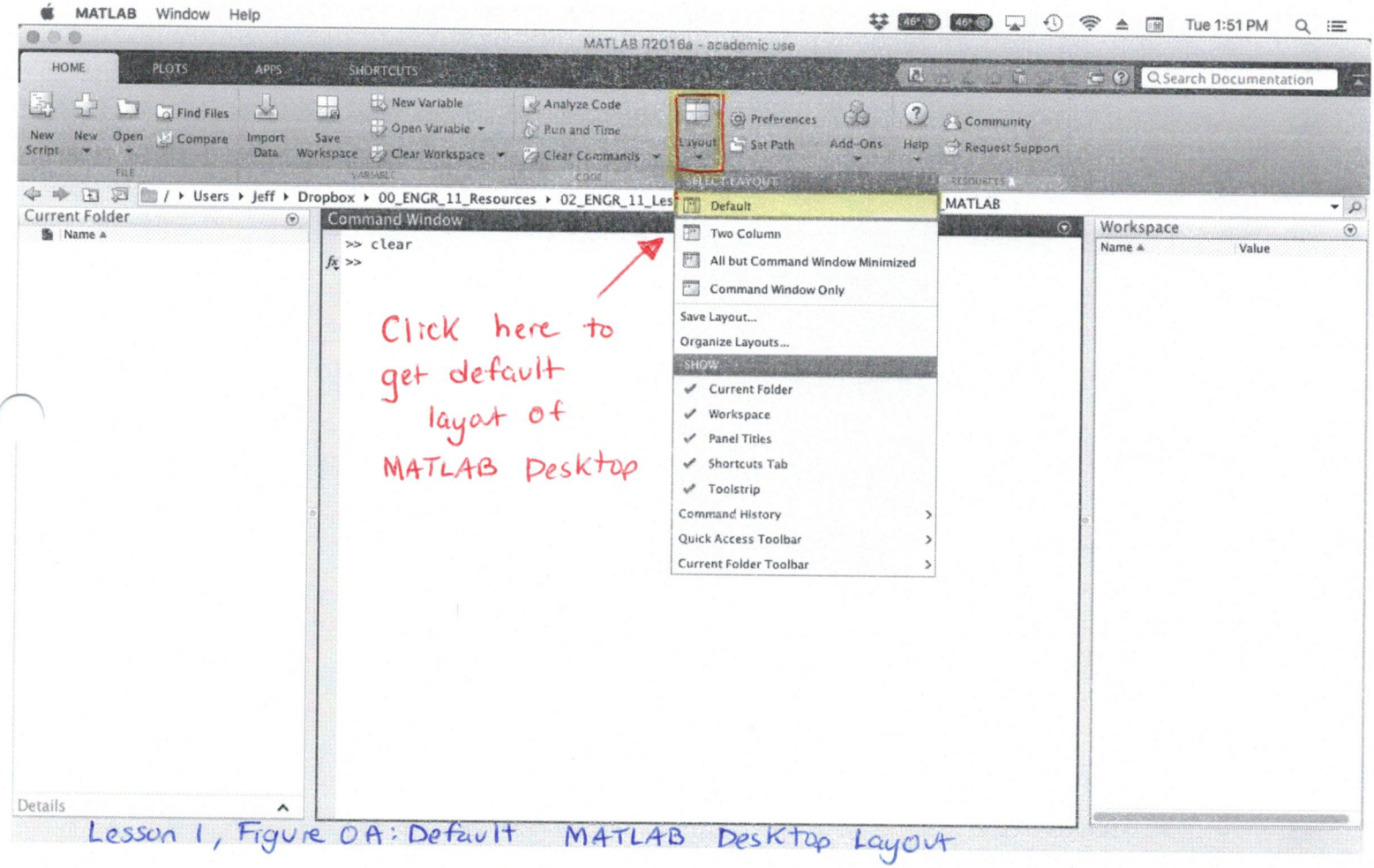
This naming convention follows from the fact that MATLAB has been designed as a Matrix-based language. In other words, while most general purpose programming languages usually work with numbers one at a time, MATLAB is designed to operate on whole matrices and arrays.

☐ However the default layout for the MATLAB Desktop contains three main windows:

① The command window

② The Workspace window

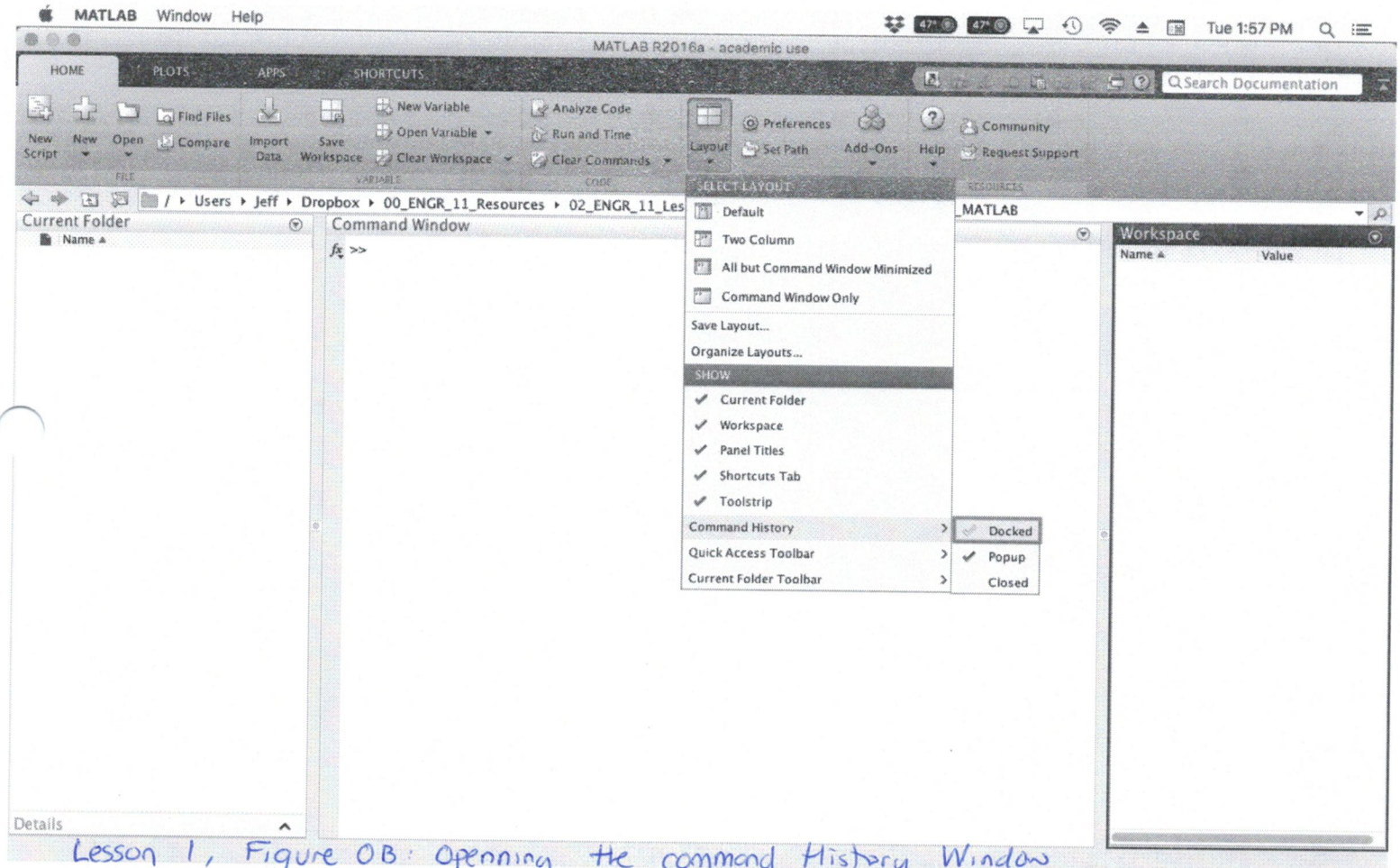
③ The current folder window



Lesson 1, Figure 0A: Default MATLAB Desktop Layout

☐ To ensure you begin with the default layout, click on the Layout button. On the 'Select Layout' menu that appears click the Default item.

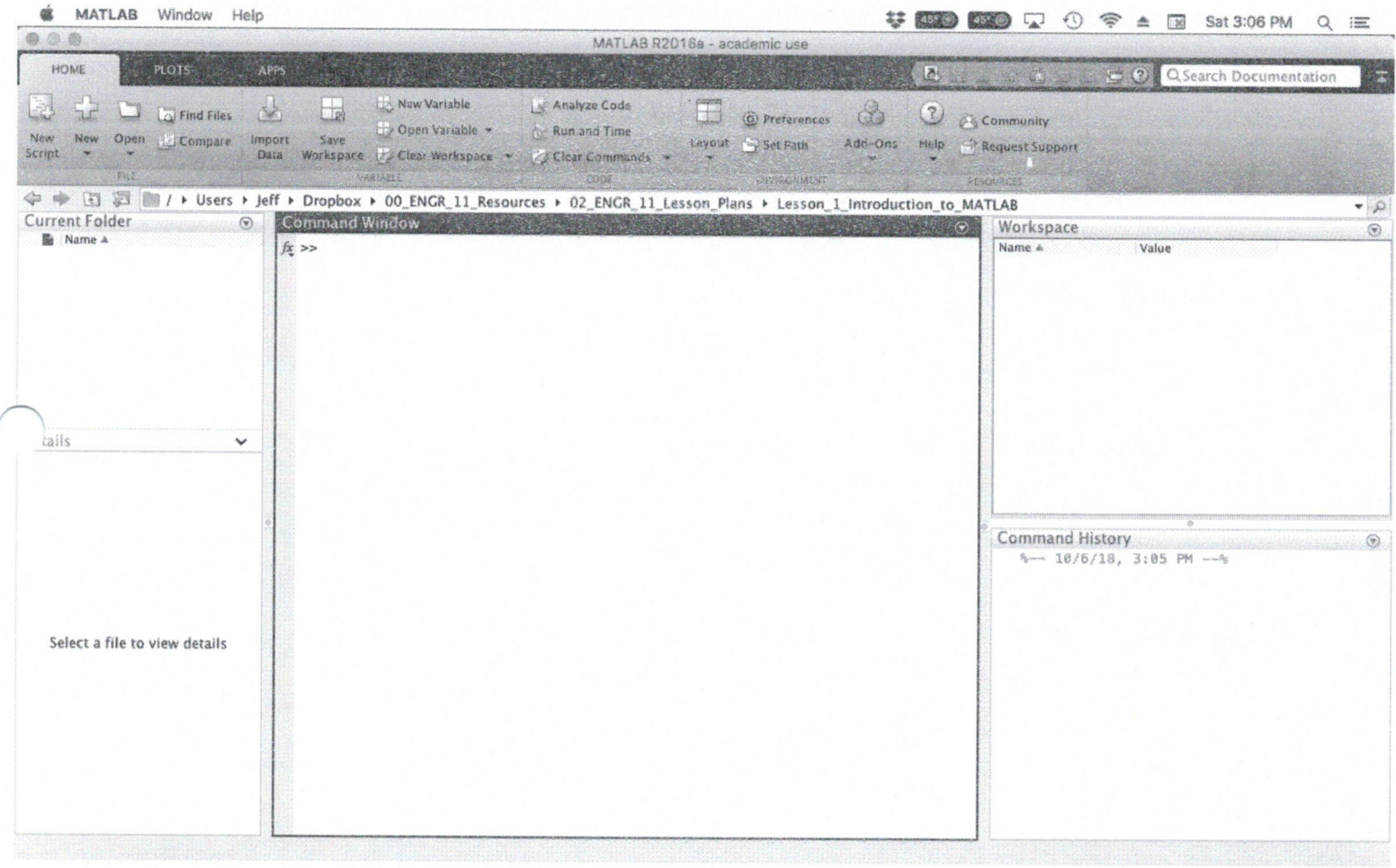
- For our purposes in this lesson, we will use one more window known as the Command History window.
- To access this window, click on the layout button, hover over the 'command History' item and click on 'Docked'.



Lesson 1, Figure 0B: Openning the command History Window

- Now, we can rearrange our screen so that we match the screen presented in this lesson (see next page).
- Notice that we can click and drag to rearrange the windows as we like.

□ When we open our MATLAB software, a new window will appear on our screen (as seen below in Figure 1). This window is the MATLAB Desktop and appears for the first time in its default layout.

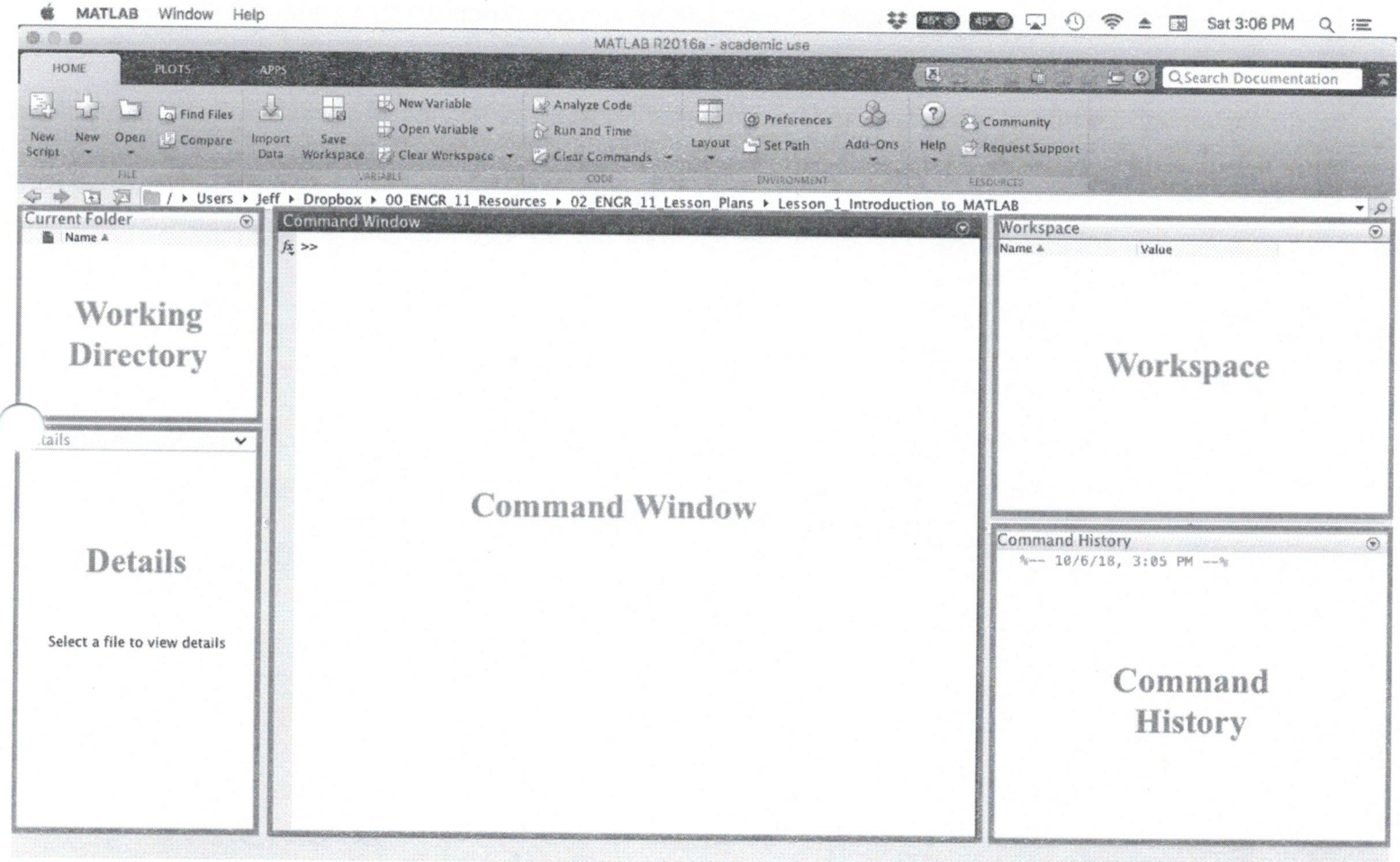


Lesson 1, Figure 1: The MATLAB Desktop Window

□ The exact appearance of the basic MATLAB Desktop window depends on which version of MATLAB we've installed on our machine.

☐ Luckily, all versions of MATLAB share commonalities. For example, the MATLAB Desktop always includes a selection of windows that can be organized, moved, and hidden by us as users. Let's investigate three such windows viewed in the MATLAB Desktop below:

① Command Window
② Workspace Window
③ Command History Window



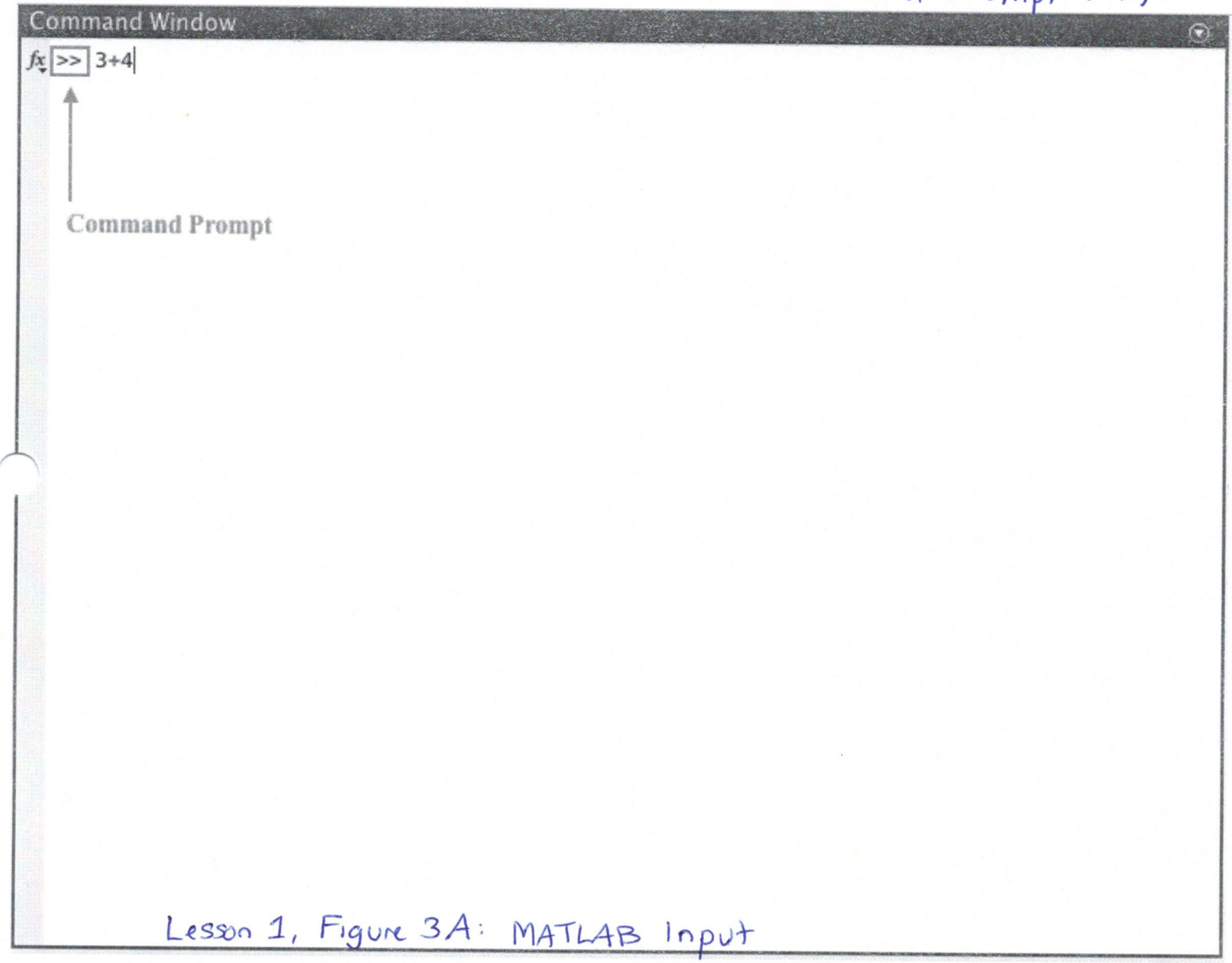
Lesson 1, Figure 2: Basic Windows of the MATLAB Desktop

☐ The Command Window:

This panel will be used to enter MATLAB commands as individual statements. Upon doing so, MATLAB will immediately execute our command and display any results in the same command window.

Example 1: Use MATLAB Command Window as a basic calculator to add the numbers 3 and 4

Solution: □ As discussed previously, we can execute commands in the Command Window. To do so, we type our commands after the MATLAB Command Prompt (>>):



Lesson 1, Figure 3A: MATLAB Input

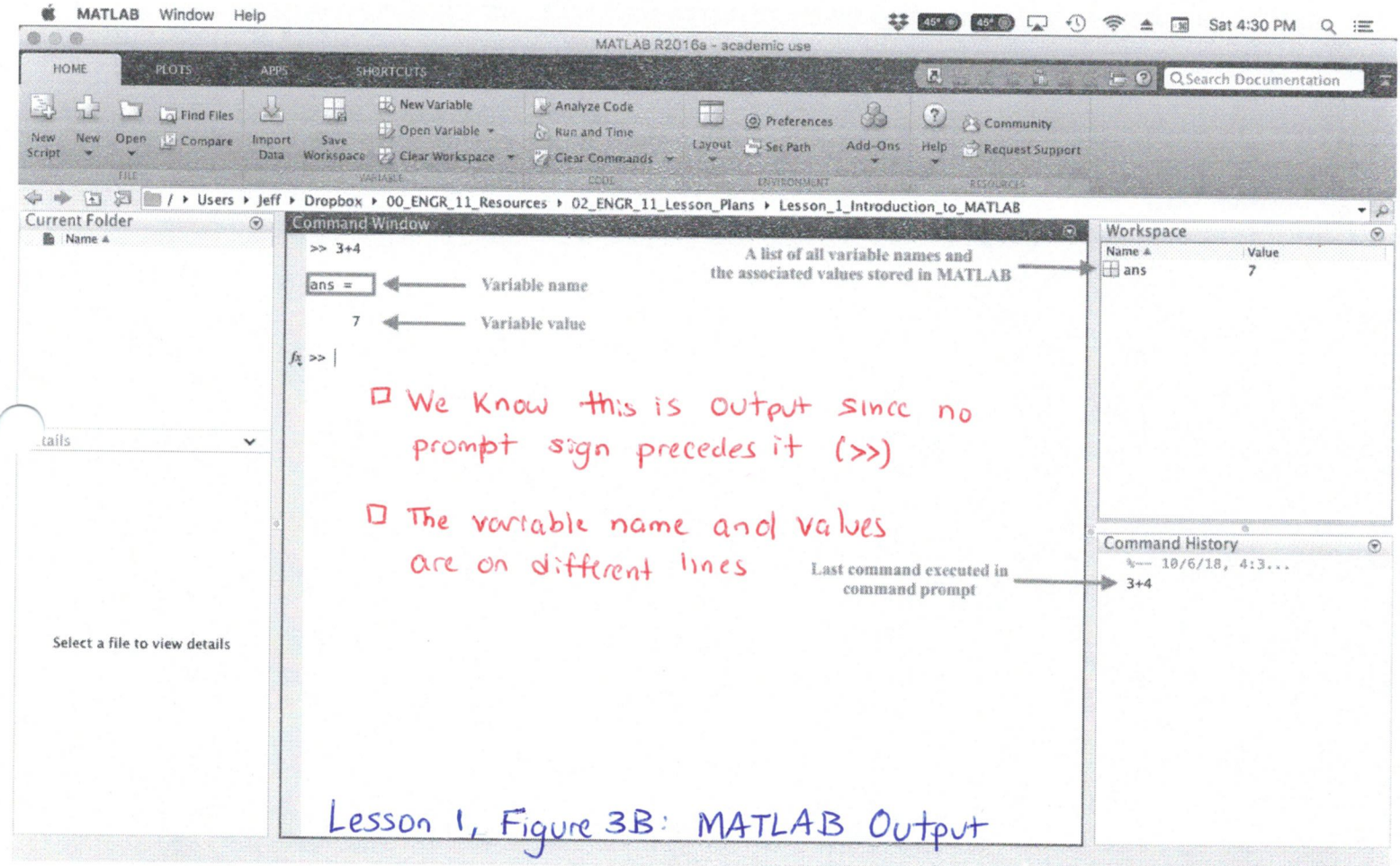
□ Once we've typed the command we want to execute, we press the Enter key.

□ In the Command Window, only the command(s) typed after the command prompt are executed when pressing Enter. All previously executed commands remain unchanged.

Example 1 continued...

- After we press the Enter key to execute our command, MATLAB will display our result in the command window.

- MATLAB Input is always preceded by the Prompt sign >> while output immediately follows and takes format seen below



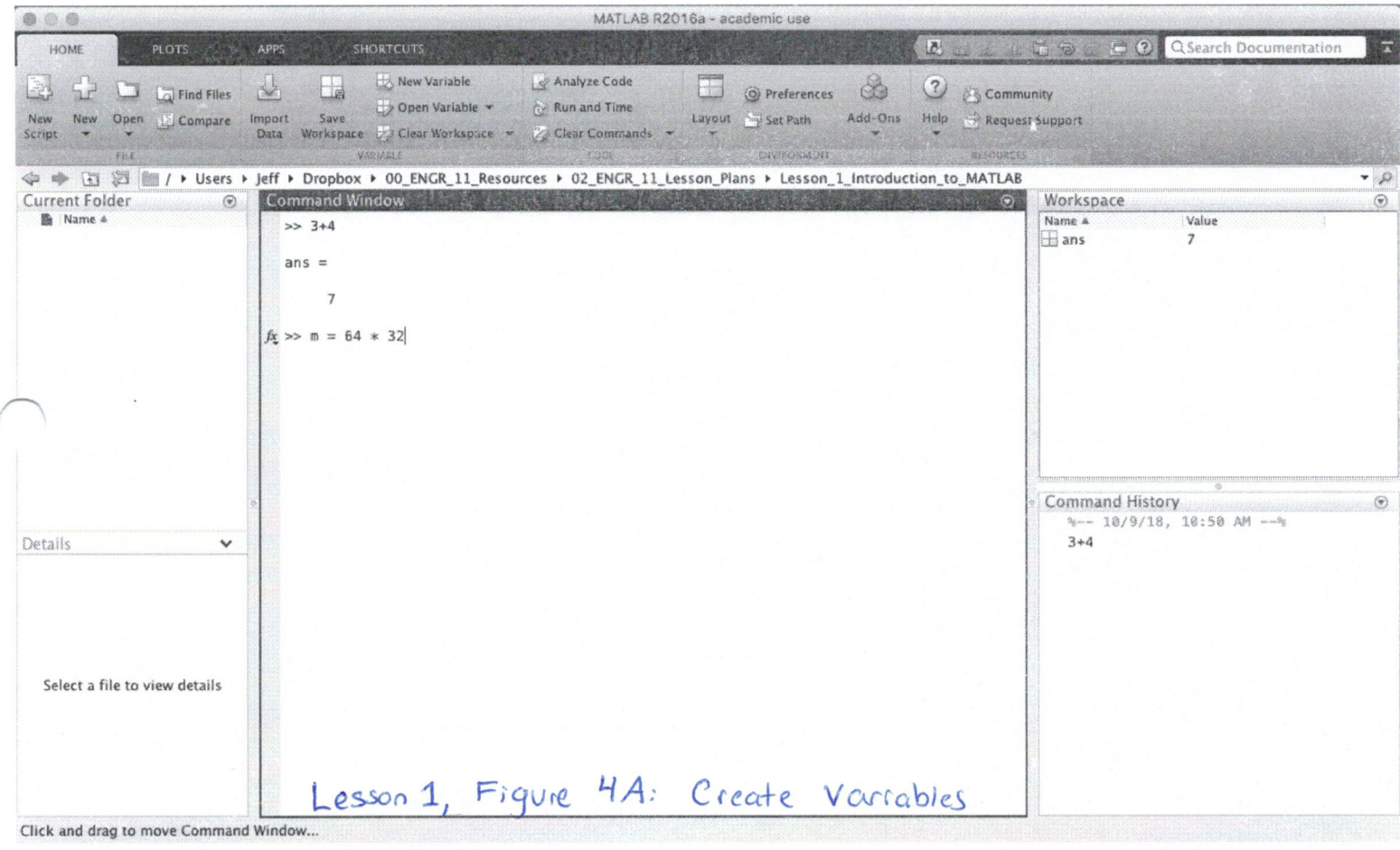
Lesson 1, Figure 3B: MATLAB Output

- We know this is output since no prompt sign precedes it (>>)

- The variable name and values are on different lines

- Unless otherwise specified, MATLAB stores calculations executed in the command prompt in a variable named ans

Lesson 1, Example 2: Use the MATLAB command window as a basic calculator to multiply 64 by 32. Then, assign the result of this operation to a variable m.



Lesson 1, Figure 4A: Create Variables

Solution: The equals sign (=) in MATLAB is used to assign values to user defined variables. Hence we refer to this symbol (=) as the assignment operator

☐ The basic anatomy of a command involving the assignment operator looks as follows

assignment operator

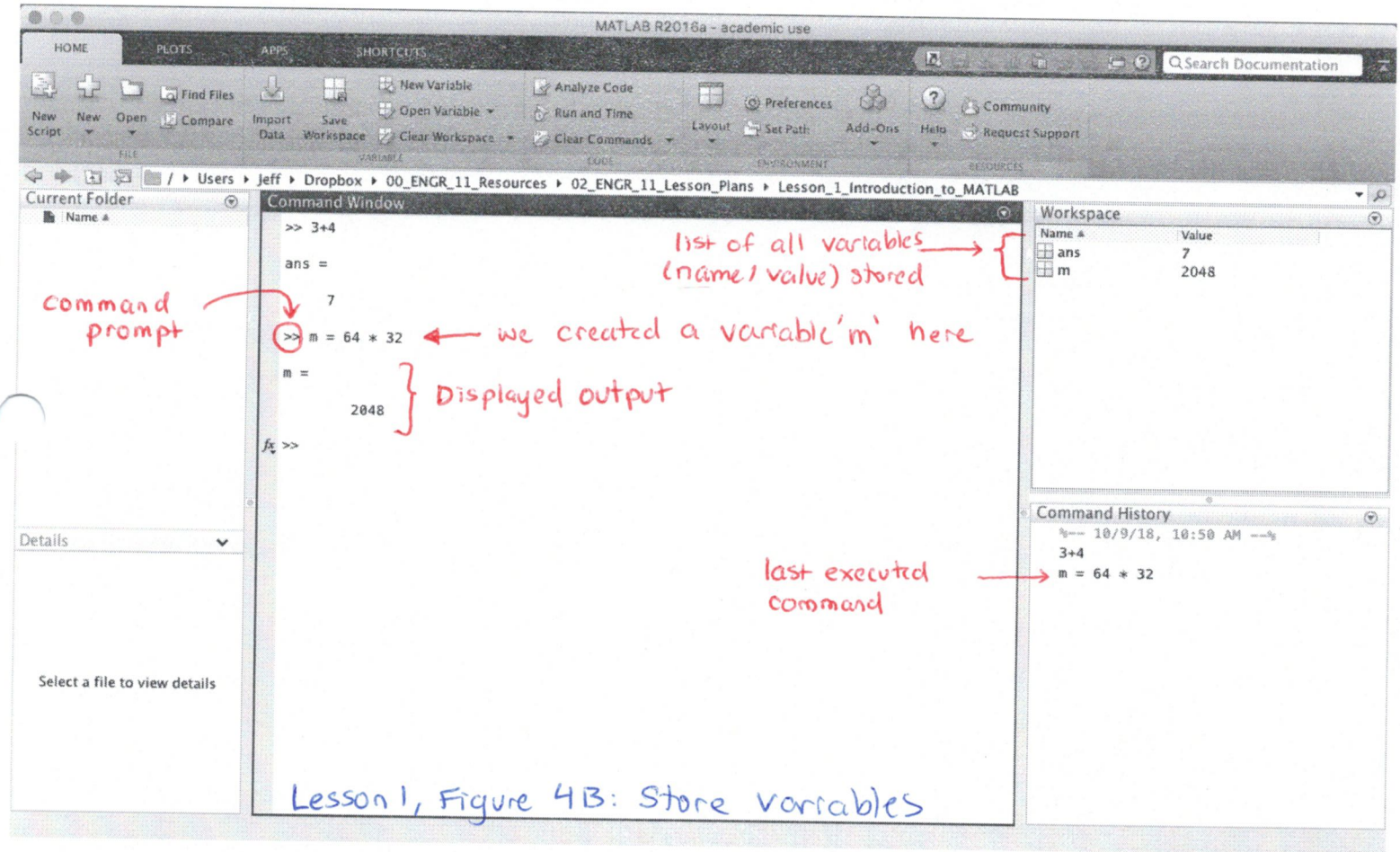
variable = expression

left-hand side | right-hand side

In our case, we have

m = 64 * 32

variable | expression



Lesson 1, Figure 4B: Store variables

☐ When we type this command into MATLAB, the expression* on the right-hand side of the = sign is assigned to the variable on the left-hand side.

* As we will discuss later in this course, MATLAB operators are assigned precidence levels which is why the output of 64 * 32 is stored as m ...

☐ Notice when we typed the command

```
m = 64 * 32
```

and then pressed the Enter button,

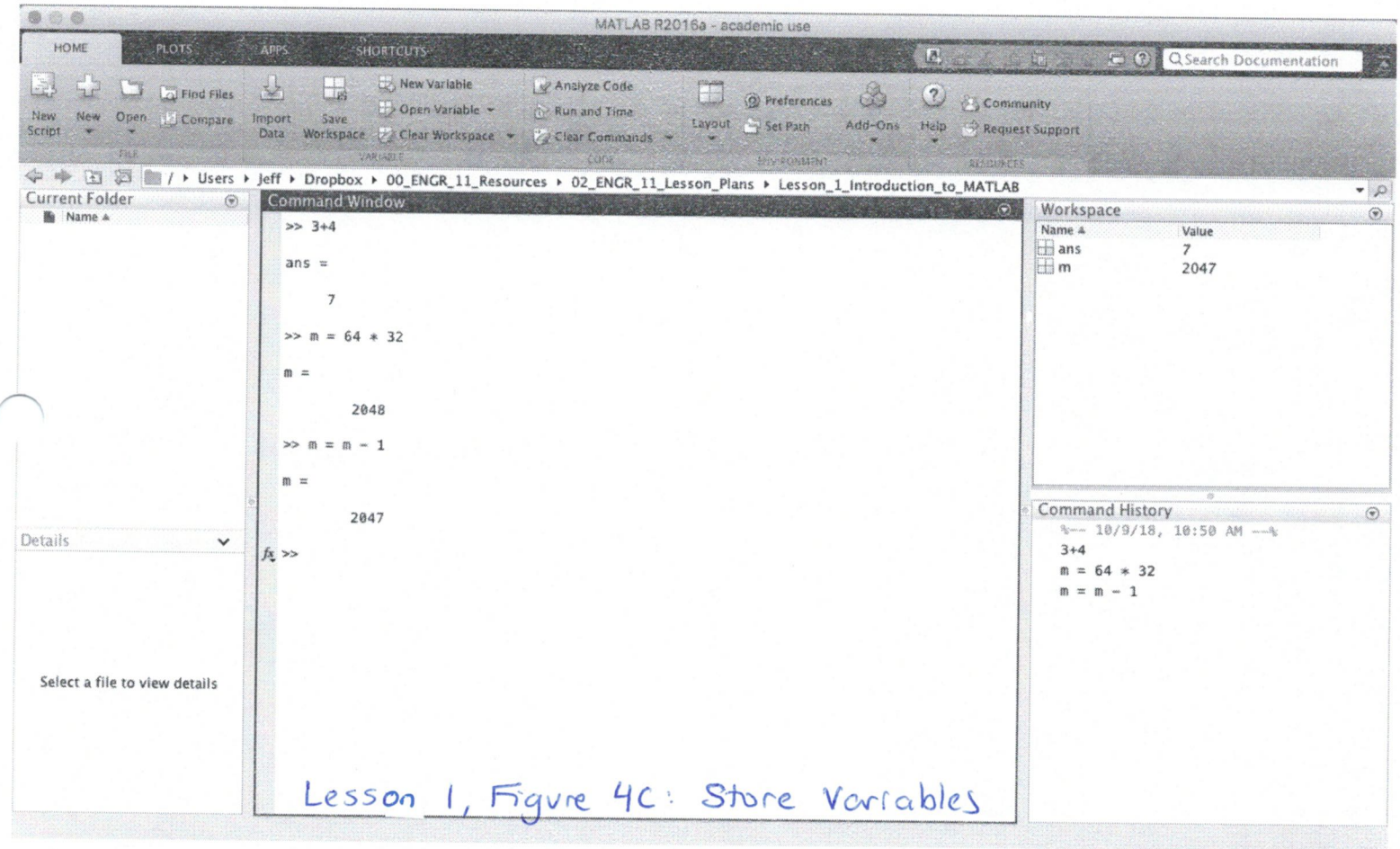
MATLAB first evaluated the product

```
64 * 32
```

and then assigned the result (output)

to the variable m.

- Because the equals sign (=) is used to assign expressions to variables, we will want to think of this assignment operator differently from the meaning of the = sign in our other math classes.

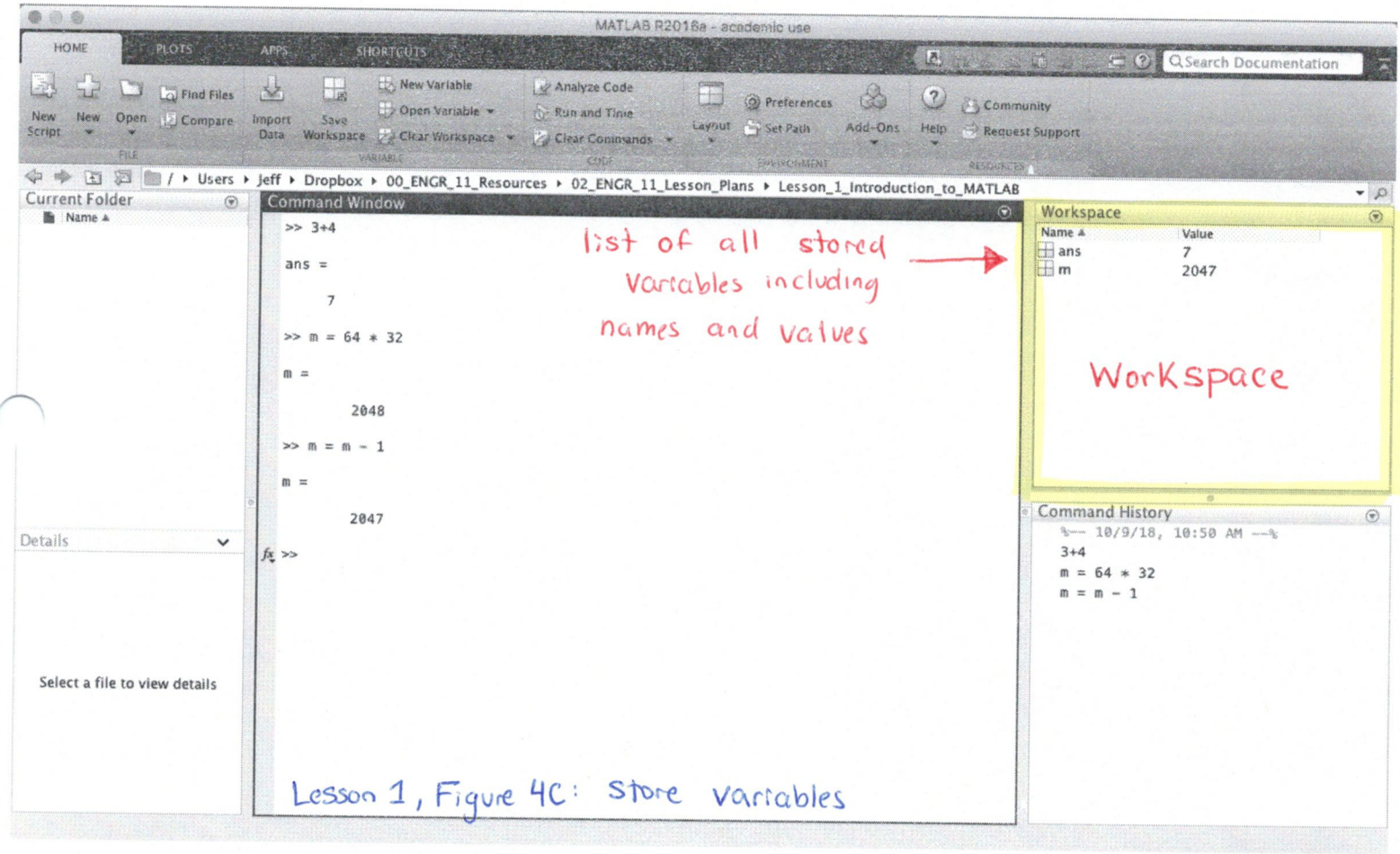


Lesson 1, Figure 4C: Store Variables

- In the example above, we overwrite the pre-existing value of m with a new value. In english, we might read the code m = m - 1 as: "take the previous value stored in variable m, subtract 1 from this value and assign the result to variable m (thus overwriting the previous value).

- Let's revisit L1, Fig 4C: Store variables. Notice that in the Workspace window, we can view a list of all variables we create and store in MATLAB.

- We can think of MATLAB's workspace as the working memory of MATLAB.



- In otherwords, as we go about entering commands and storing variables, MATLAB's workspace will track and record all of our work to give us access to all variables we've stored. With this in mind, let's take a look at the Workspace Window.

☐ The Workspace Window :

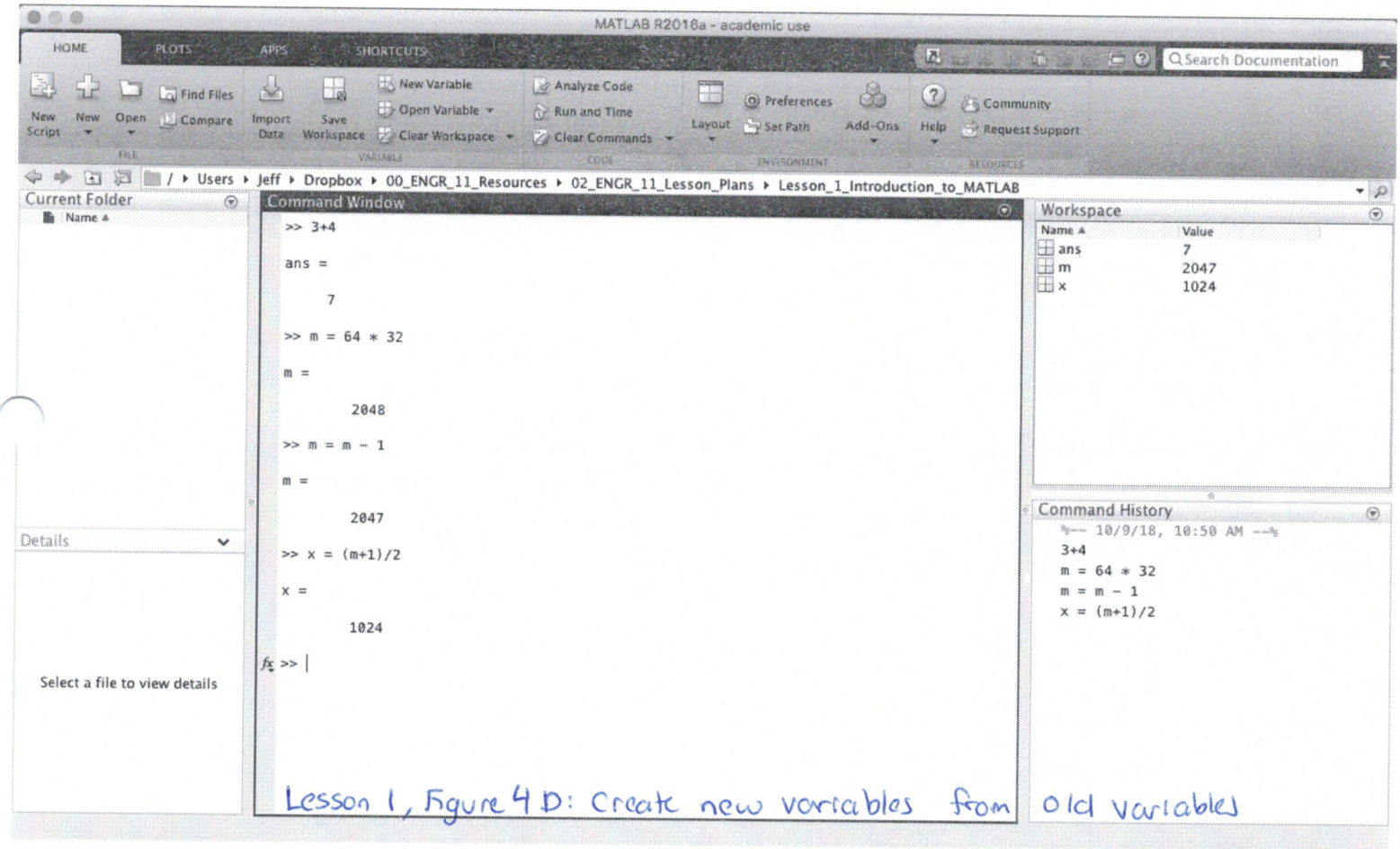
- The workspace browser enables us to view and interactively manage the variables or objects stored in the MATLAB workspace.

- The MATLAB workspace consists of all variables we create and store in memory during a particular MATLAB session*

  * For our current purposes, we can think of a MATLAB session as our entire experience on MATLAB between the time we open a new instance of the MATLAB program and the time we quit the program (assuming our new instance is the only one running on our machine).

Lesson 1, Example 3: Create new variables from old variables

Solution: Once we have stored a variable in MATLAB's workspace, we can refer back to this variable to define new variables



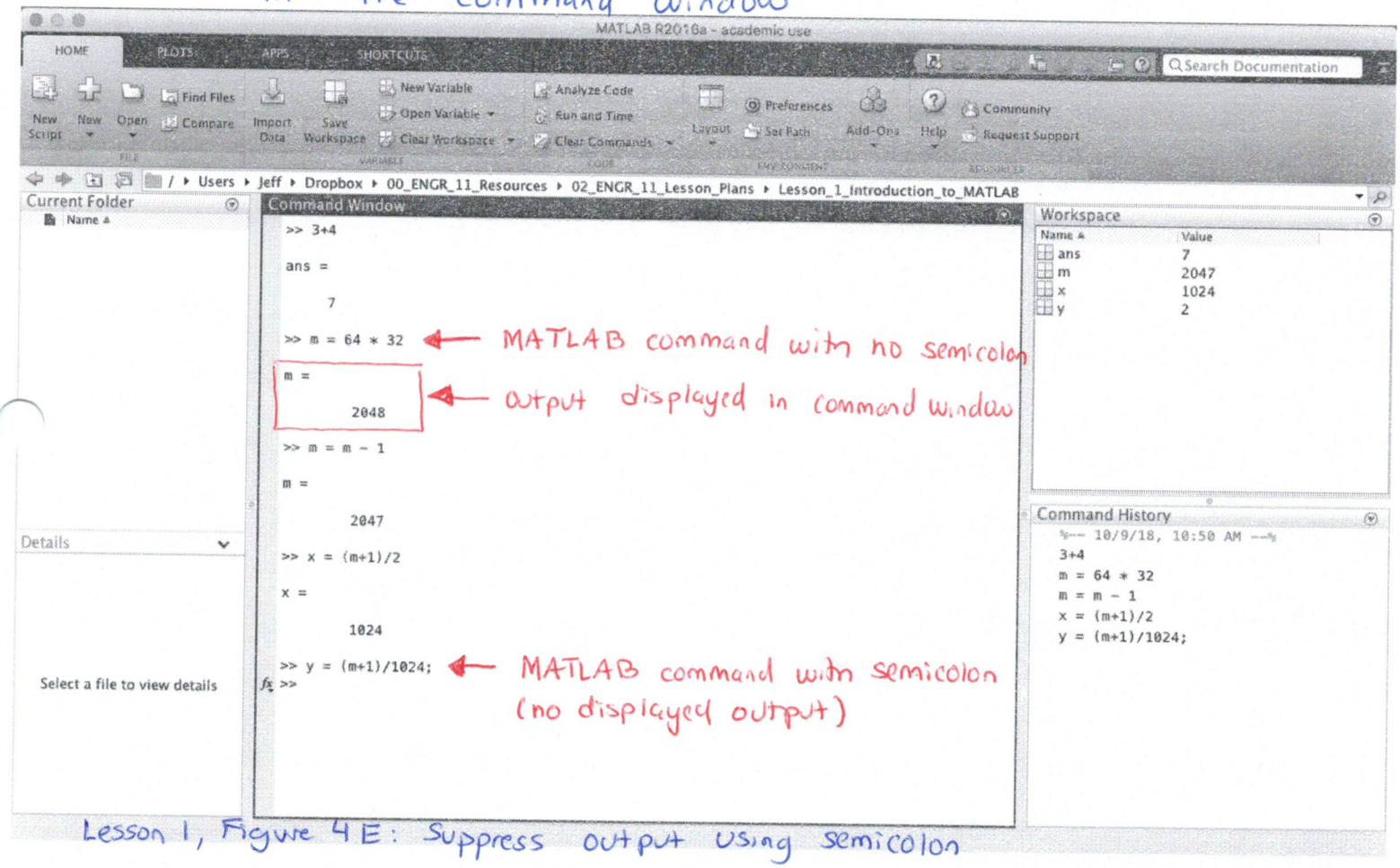
Lesson 1, Figure 4D: Create new variables from old variables

☐ In this example, we create a new variable named x that stores the result of the calculation $(m+1) \div 2 = \frac{(m+1)}{2}$

(again: operator precidence matters here... think PEMDAS for now).

Lesson 1, Example 4: Surpress output in the Command Window

Solution: As we noticed in previous examples, when we enter a command into the command Window without placing a semicolon at the end of our command, MATLAB displays the result of such a command in the command window



Lesson 1, Figure 4E: Suppress output using semicolon

- By adding a semicolon to the end of our command, we can suppress our output in the command window.

Thus, the command `y = (m+1)/1024;` executes the command and stores the result in the workspace as variable y but does not display the output in the command Window.

☐ We can use semicolons for many reasons including the following:

- Often times, we will execute commands in which the results are obvious. In these cases, we may not need to see the output of our work and thus we can surpress the results using (;)

- When designing algorithms to solve problems, we will sometimes creak commands that generate very large output. Displaying massive amounts of data can be confusing and distracting. In this case, we can surpress this output using (;)

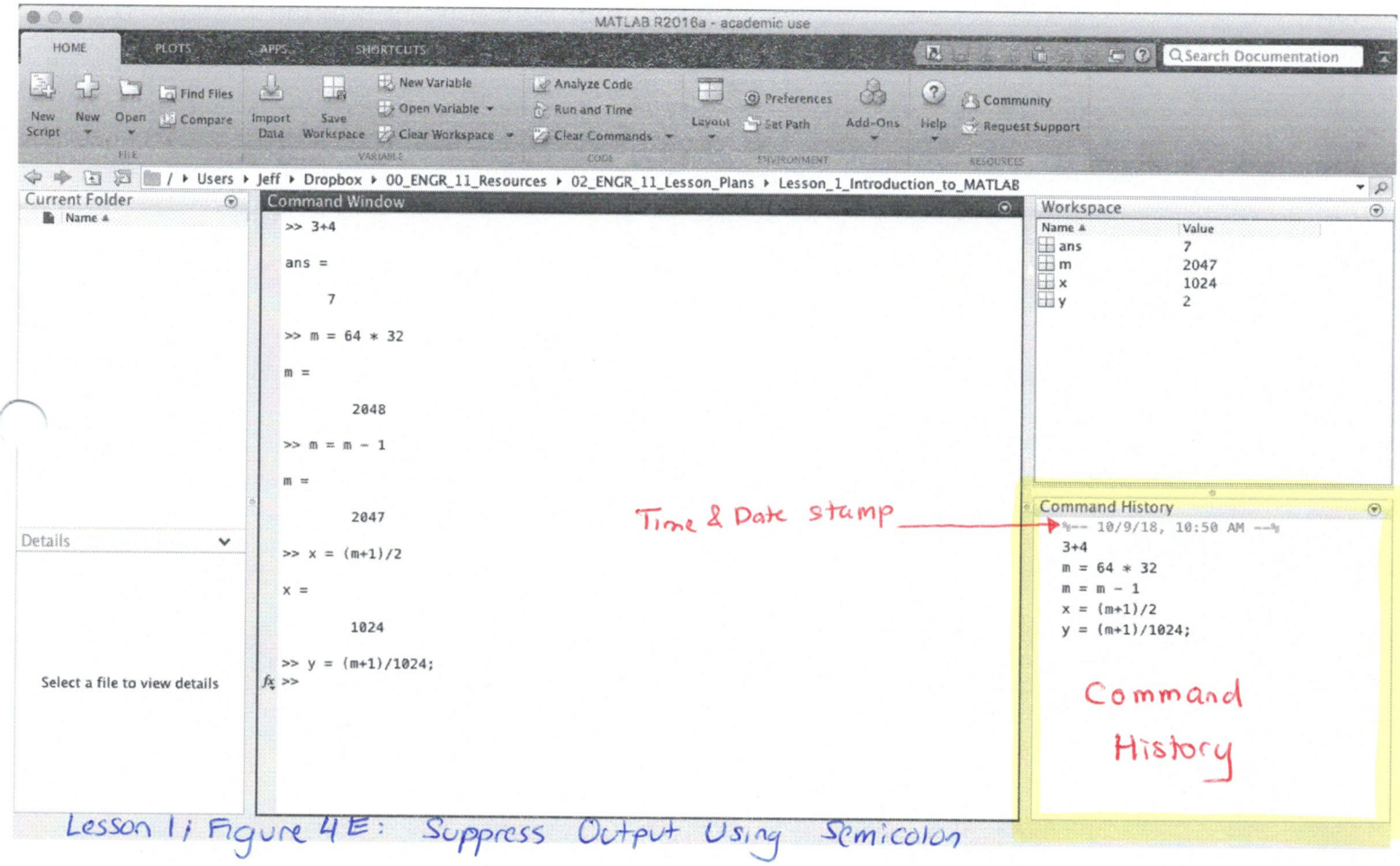
- The command history window is quite helpful because we cannot go back to a previous line in the command window itself.

- Instead, we can think about the Command Window as displaying a read-only log of all previous commands executed during our session.

- Thus, if we make a mistake and execute an erroneous command, we cannot go back to a previous line in the command window, make a correction, and re-execute our command. We can, however, skillfully use the Command History to recall and edit previous commands.

Lesson 1, Example 5: Use the command History to recall previously executed commands

Solution: Notice that in the Command History Window, MATLAB displays a log of statements that we've run in our current and previous MATLAB sessions.



Lesson 1, Figure 4E: Suppress Output Using Semicolon

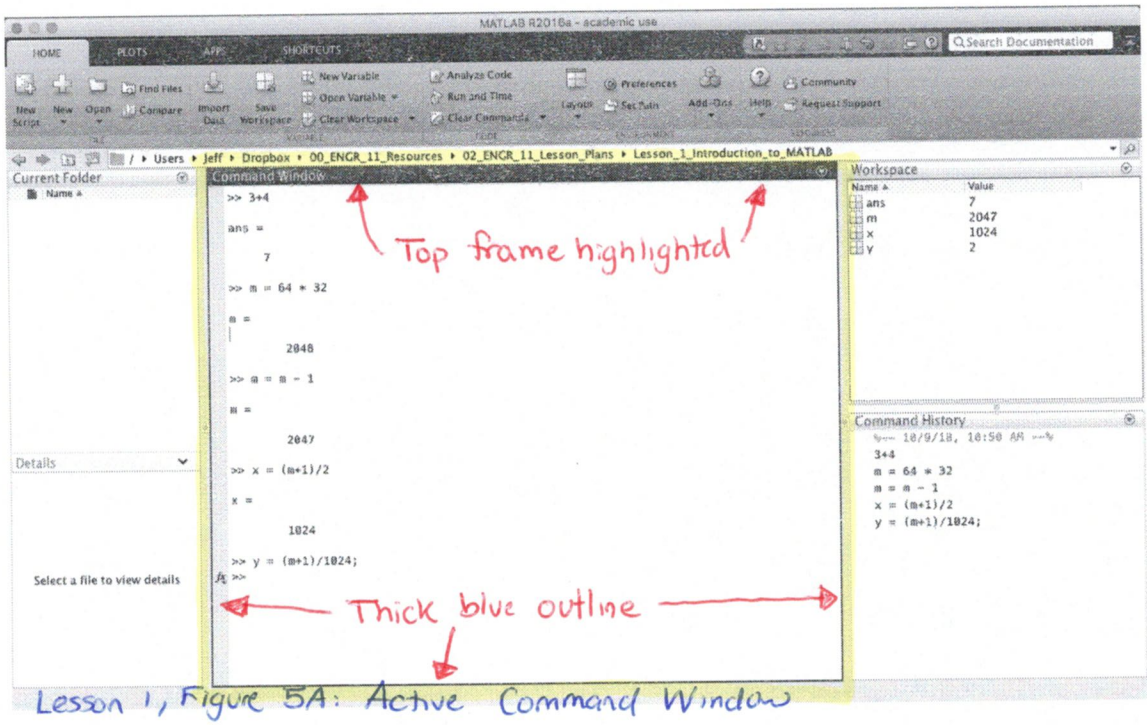
☐ The Command History records all commands that we execute and organizes these commands by the time and date of each session followed by all commands entered in that session.

- We can use the fact that MATLAB records all statements we run in the command window to recall previous commands.

- For example, if we activate* the command window by clicking on the command prompt (>>), we can then press the Up arrow (↑) on our keyboard to scroll up through previous commands. Similarly, the down arrow (↓) allows us to scroll down through our command history.

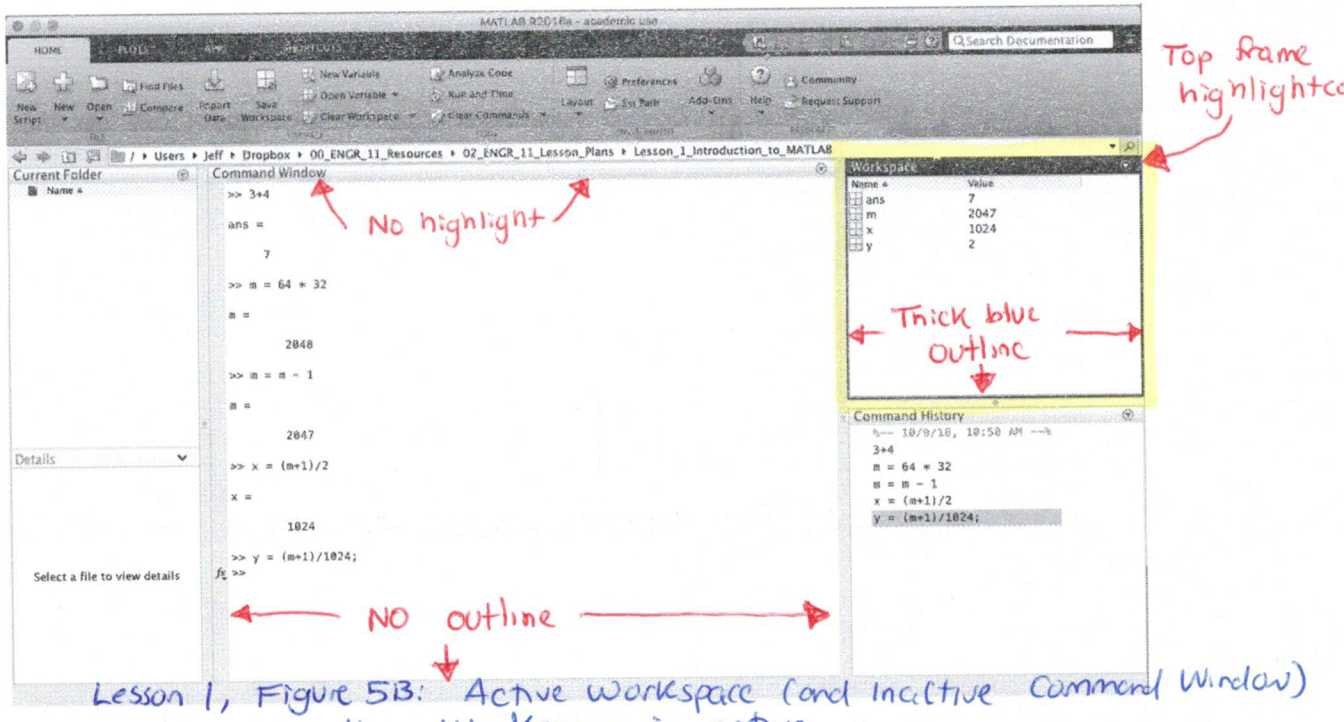
  * to ascertain whether or not the command window is active, see next page

- When the command is displayed at the command prompt (>>), we can modify the command before we re-execute it.



Lesson 1, Figure 5A: Active Command Window

☐ Here, the command window is active. We can tell because the top frame of the command window is highlighted in blue and the body of the window is outlined with a thick blue line



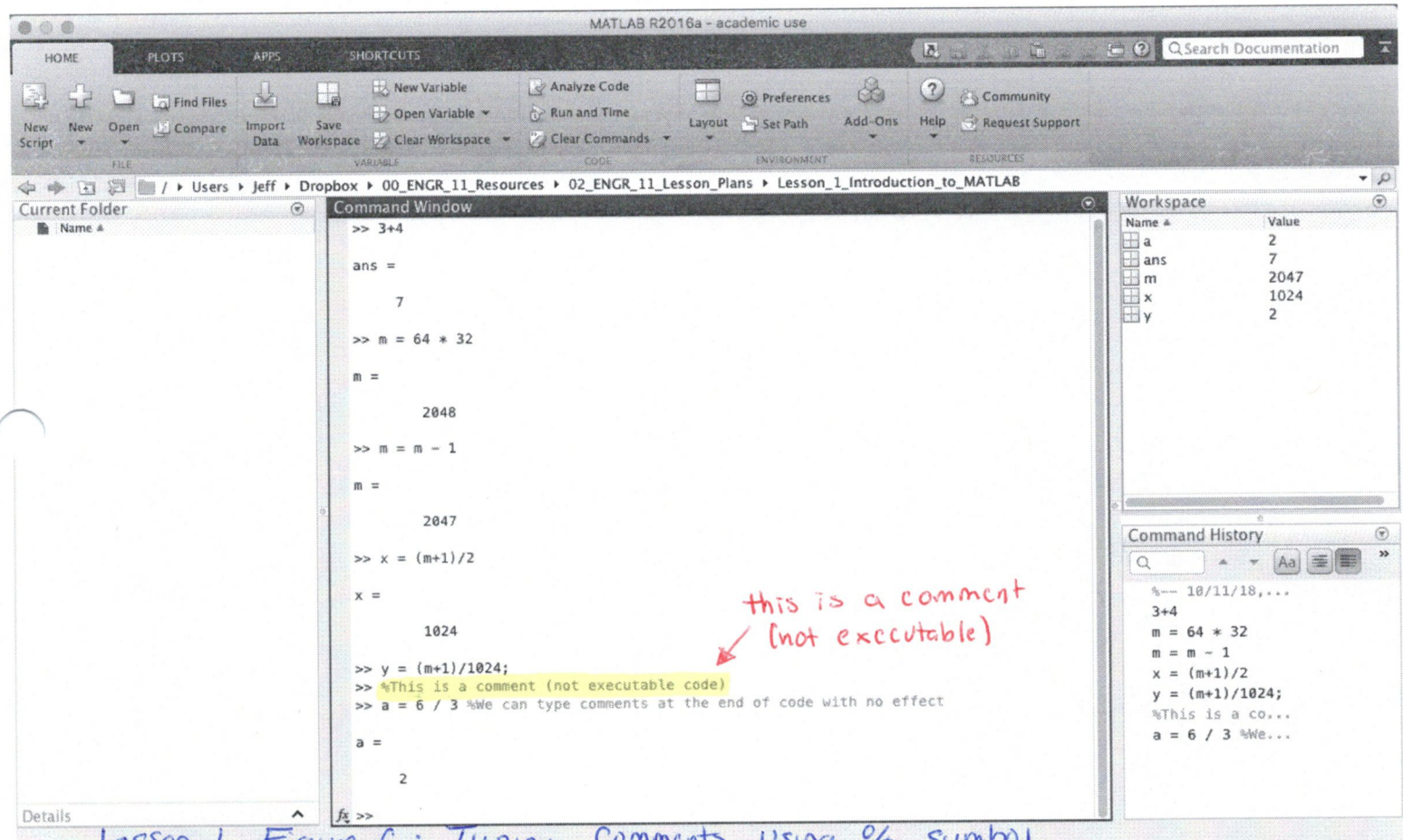
Lesson 1, Figure 5B: Active Workspace (and Inactive Command Window)

☐ In this second image the Workspace is active

Special Symbols and Commands

The % Symbol

- We can type the % symbol (percent symbol) at the beginning of a line to designate that line as a comment (as opposed to an executable command). When we press the Enter button on a line beginning with %, no command is executed. We will usually not type comments into the command window. Instead we will use comments later to document script and function files that we create to solve problems.
- MATLAB automatically colors comments green.
- Typing the percent symbol at the end of a line in the command prompt has no effect.



Lesson 1, Figure 6: Typing Comments using % symbol

Special Symbols and Commands, continued...

The clc command

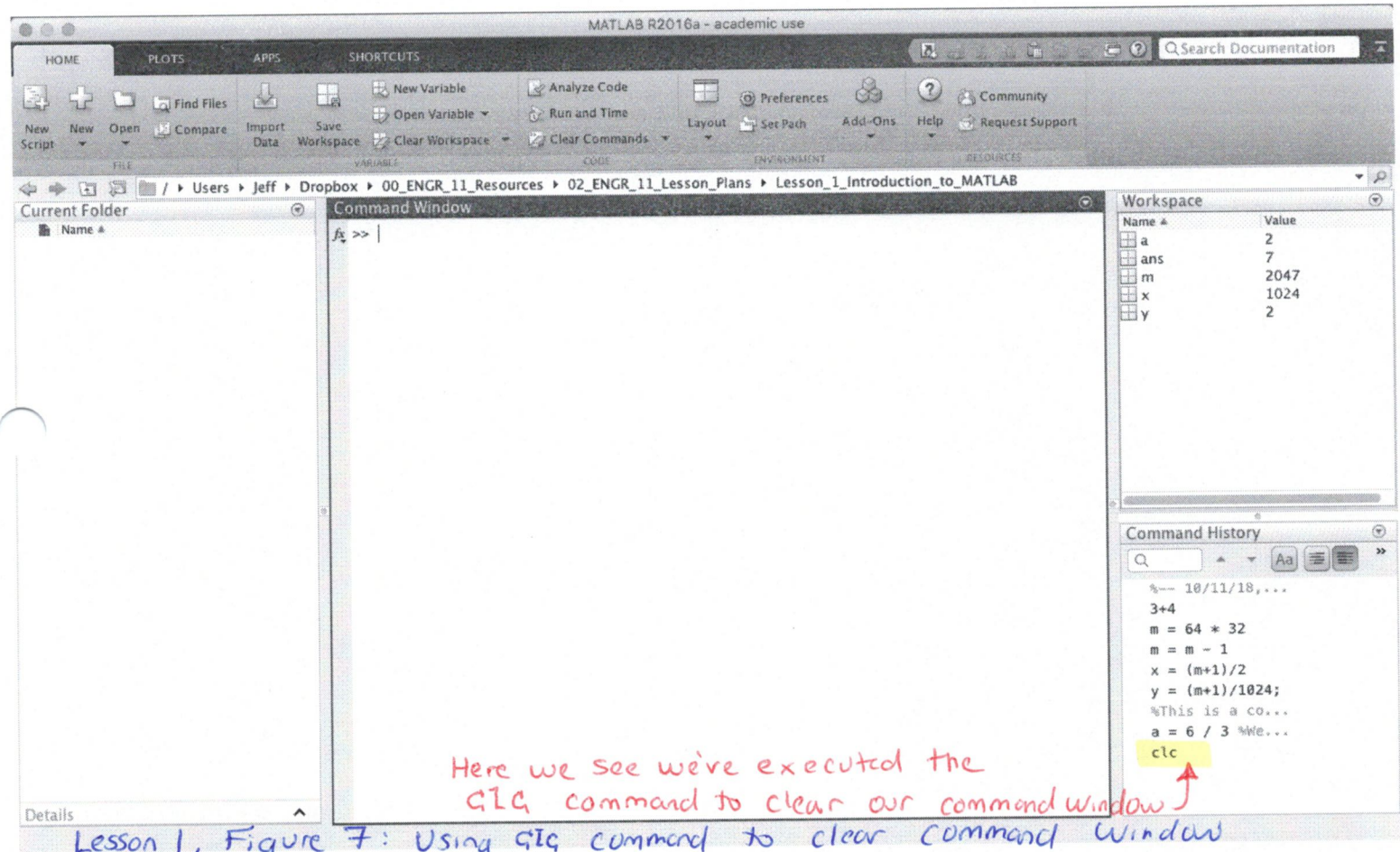
- We can use the clc command (type lower case clc into command window and then press Enter) to clear the command window.

- Once the clc command is executed, we see a clear command window on screen

- We can use the cld command to clean up our command window: after working in the command window and executing many commands, the displayed output may become very long and cumbersome... The clc command can help us keep our workspace clean.

The clc command, continued...

- Notice that the clc command has no effect on any variables stored in the workspace nor does it alter the command history



Lesson 1, Figure 7: Using clc command to clear Command Window

- All variables we had stored prior to executing clc still exist in the workspace and can be used

- Our Command History remains intact and we can still scroll up through previous commands to recall code we executed before

Overview of our work in Command Window

- To type a command, we need to activate the command window a place the cursor next to the command prompt (>>)

- When we type a command in the Command Window and press the Enter key, the command is executed. Any output generated by our command will be displayed in the Command Window

- If we type a semicolon (;) at the end of our command and execute the command, the output of the command will not be displayed. This is useful to surpress obvious or large output.

Overview of our work in the Command Window, continued ...

- We can use the % sign to type comments (nonexecutable code) in MATLAB. Everything typed after the % symbol will not effect the execution but instead be used as documentation. We will use comments most frequently in script and function files to properly document our work.

- We use the `clc` command to clear all previous input/output displayed in the command window and start fresh with a blank command window.

Performing Basic Arithmetic with Scalars

- Now that we have some basic familiarity with the MATLAB Desktop, let's investigate how we can use MATLAB to do arithmetic operations on scalars.

- We can execute arithmetic operations in two ways

  1. Calculate directly without explicitly assigning the output result to a variable (by default MATLAB will assign the output of such direct calculations to variable ans)

  2. Assign the output of arithmetic calculations to variables and store the results for later use

- Let's start by considering (1).

□ We begin by focusing on a limited set of aritmetic operations:

Operation name	symbol	Example
Addition	+	8 + 4
Subtraction	−	8 − 4
Multiplication	*	8 * 4
Right division	/	8 / 4
Left division	\	8\4 = 8^-1 . 4 =
Exponentiation	^	8^4 = 8^4
Negation	−	−4

□ Notice that all symbols except for left division are identical to a standard graphing calculator.

□ Left division is mostly used for solving Linear-Systems Proble and if we write A \ b we can think of this like $A^{-1} \cdot b$.

□ Below, we execute each of our arithmetic operations between the numbers 8 (left) and 4 (right)

Command Window

```
>> 8+4

ans =

    12

>> 8-4

ans =

     4

>> 8*4

ans =

    32

>> 8/4

ans =

     2

>> 8\4

ans =

    0.5000

>> 8^4

ans =

        4096

fx >>
```

□ When using MATLAB as a calculator, we do not explicitly assign our expressions to variables. Thus, MATLAB automatically stores these calculations in the variable ans

□ In this work, we do not use the semicolon to suppress output. Thus, after we execute each command, MATLAB displays the output in the Command Window.

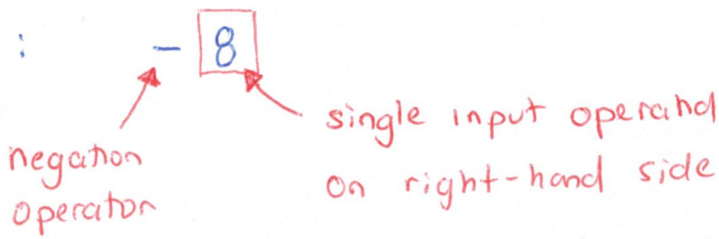
Lesson 1, Figure 8: Basic Arithmetic Operations

- Looking back at our basic arithmetic operations we see two types:

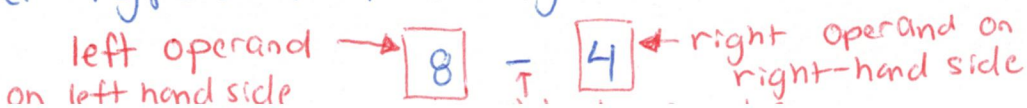
  Unary operator: an operator that operates on a single value (also known as an operand)

  binary operator: an operator that operates on two operands (two input values).

- For example, the negation operator (-) expects it's single operand typed on the right-hand side as follows:



- On the other hand, the subtraction operator (-) expects two operands, one typed on the left-hand side and the other typed on the right-hand side:



- To better understand how MATLAB executes arithmetic commands, lets recall our knowledge of order of operations (PEMDAS):

Should be written
P
E
MD
AS

Eg: $-5^2 + 14 \div 7 \cdot 2$

$= -1 \cdot (5^2) + \left[\frac{14}{7}\right] \cdot 2$

$= -1 \cdot 25 + 2 \cdot 2$

$= -25 + 4$

$= -21$

- In MATLAB, the idea of order of operation is known as Precedence levels and determines the order in which MATLAB evaluates expressions

Operator Precedence for Basic Arithmetic

- MATLAB executes arithmetic operations based on pre-defined precedence levels associated with each operation.

- For our six basic arithmetic operations, the precedence rules used by MATLAB are listed below ordered from highest precedence level to the lowest precedence level:

Precedence level	Mathematical Operation
First (highest)	Parenthesis For nested parenthesis, the innermost command is executed fi
Second	Exponentiation
Third	Negation
Fourth	Multiplication, left/right division (have equal precedence levels)
Fifth (lowest)	Addition and subtraction (have equal precedence levels)

- When executing a typed command (expression) that includes several operations, MATLAB follows the following rules:

  1. higher-precedence operations are executed before lower-precedence operations.

  2. if two (or more) operations have equal precedence in an expression, these operations are evaluated from left to right.

- Notice we can use parenthesis strategically to change the order of operations and increase the precedence levels of lower-level operations.

☐ Let's take a look at some examples of how MATLAB uses precedence levels to execute basic arithmetic

Command Window

```
>> 6+10/2

ans =

    11

>> (6+10)/2

ans =

     8

>> 2 + 4/8 + 6

ans =

    8.5000

>> 4^3/2

ans =

    32

>> 4^(3/2) + 16^0.25

ans =

    10

>> -5^2 + 14/7*2

ans =

   -21

fx >>
```

The right division 10/2 is executed first since it has higher precedence. Then the result $\frac{10}{2} = 5$ is added to 6.

The expression inside the parenthesis (6+10) is evaluated first since parenthesis have highest priority.

The right division 4/8 is executed first

The exponentiation 4^3 is executed first followed by the right division /2 (exponentiation has higher precedence than division)

The expression inside the parenthesis is executed first with 3/2 = 1.5. Then the exponents 4^1.5 and 16^0.25 are executed. Finally, addition is executed.

First the exponentiation is executed, then the negation operator −25, followed by the division 14/7 = 2, then the product 2*2 = 4. The last operation to be executed is the addition since it has lowest precedence.

Lesson 1, Figure 9: Precedence Levels for Arithmetic Operations

- Sometimes, we will type long commands that include nested parenthesis. When doing so, here are two useful practices to cultivate:
  - type left and right parenthesis at the same time
  - use ellipsis in command window

Command Window

```
>> ( 12*(12-7)+ 8*5) / (2^4+3^2)

ans =

     4

>> ( 12*(12-7)+ 8*5) / (2^4+3^2) + -5^2 + 14/7*2 + ...
4^(3/2) + 16^0.25

ans =

    -7

fx >>
```

← for nested parenthesis, the innermost command is executed first. Here, we type (then) and use arrows to fill command

← ellipsis (also called the continuation operator)

Lesson 1, Figure 10: Ellipsis for typing command on next line

- In the second command executed above, we typed three periods (...) (called an ellipsis) and then pressed Enter to continue to type our command on the next line in the command window prior to executing our command. The ellipsis is useful to type long commands that do not fit nicely on one line.

- When typing commands in the command Window, we can continue to type our command, line after line, using ellipsis. In fact, our command can be up to 4096 characters long.

see wikipedia article on "Software bug"

Note: Develop habits that minimize bugs

- As we develop our skills in MATLAB, we will begin to take on larger problems that will require us to write lots of code. When we do this, we will inevitably need to debug our work (to fix errors, flaws, failures, or faults in our code).

- While it is unrealistic to assume we can write code that contains no errors, here are some very useful habits of thought and action we can develop to help minimize the number of errors we make (and thus these tricks will make us more effective and efficient).

Note: Develop habits that minimize bugs, continued...

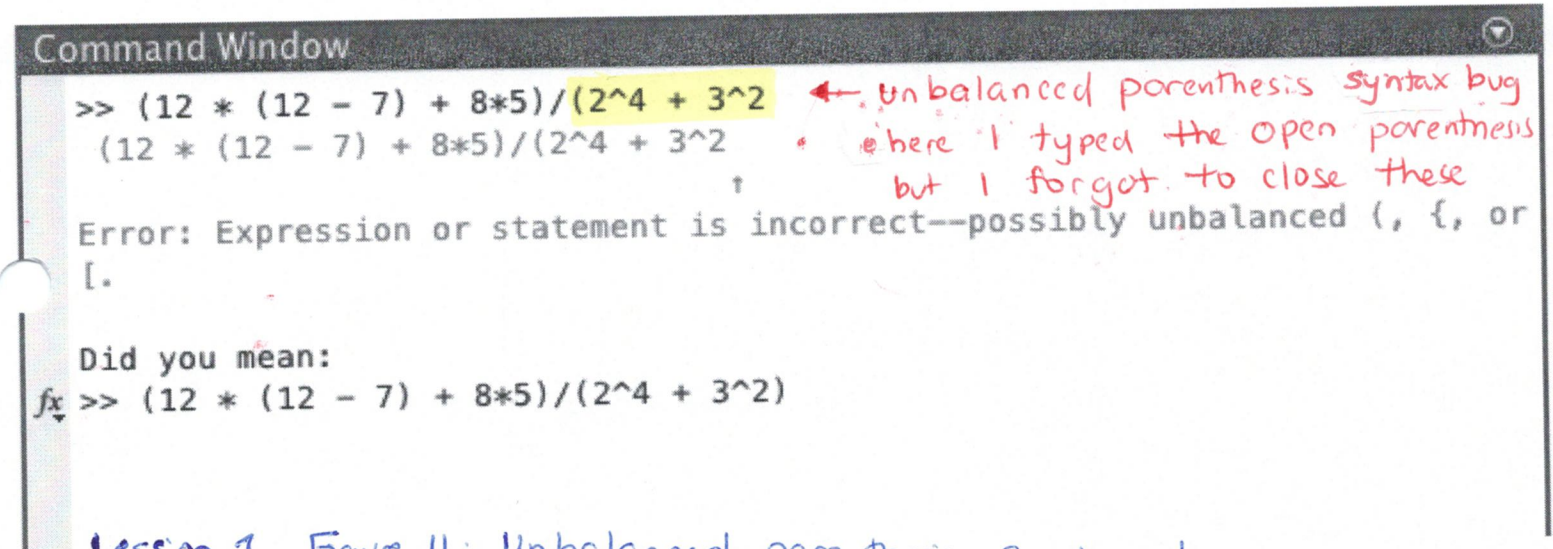
- Throughout the course, I will guide you to develop some habits that I find useful to avoid eroneous bugs. I learned these through personal experience (pain and suffering). Hopefully you can benefit from these habits to avoid re-experiencing my pain.

- The first debugging trick we will discuss is designed to help you catch and avoid unclosed parenthesis, braces, brackets, and quotes.

  - When typing commands in MATLAB, all parenthesis, brackets, curly braces and quotes are expected to come in pairs

  - We can think of the two items in a pair as an openning (left) parenthesis and a closing (right) parenthesis.

☐ As demonstrated below, typing only one of the two items in the pair leads to an unbalanced Universe and will cause MATLAB to mal function



Lesson 1, Figure 11: Unbalanced parenthesis syntax bug

☐ One way to avoid this problem is as soon as we type the openning character, we immediately type the closing character to match / balance the pair. Then, we move our curser back between the characters and continue typing.