Using MATLAB's Elementary Built-In Functions

- In addition to the basic arithmetic operations we discussed, MATLAB offers a large library of built-in "functions"* that are native to the MATLAB environment

* Notes: More about built-in functions

- In the description above, we are abusing the word function because we are actually using it to represent two different ideas:

  ① "function" as a mathematical object which relates to your understanding from Math class and all the functions you studied in math (i.e. think $y = f(x)$)

  ② "function" as a programming object which relates to a specific computational procedure: such functions take in input, execute an algorithm to manipulate said input and return a specific output...

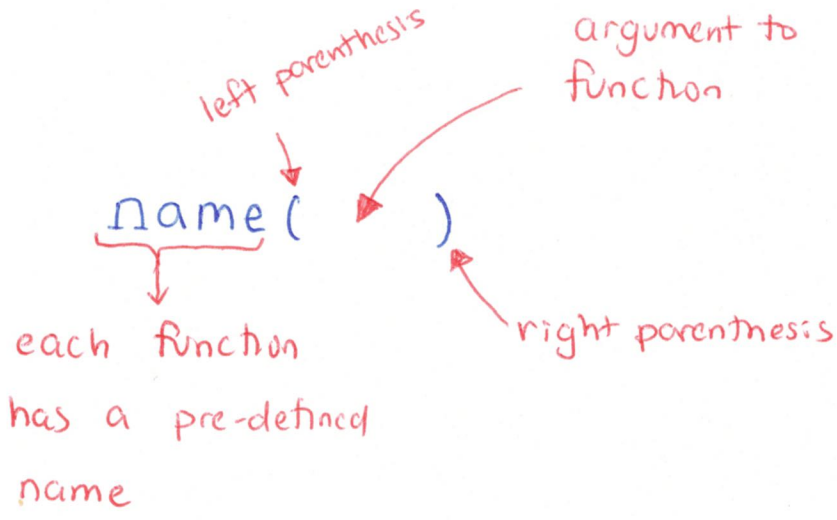
*Notes: More about built in functions, continued...

- When discussing functions in programming, ultimately we are talking about the manipulation of electric signals in hardware. In contrast functions in mathematics are abstract relationships between sets... We'll see more about this later in the course.

- When we say MATLAB offers a library of built-in functions that our "native" to the MATLAB environment, we mean that no special packages need to be imported in order to access these built-in functions. Contrast this with general purpose languages (like C or python) in which we must import special libraries of functions to use these in our work. Remember: MATLAB is designed for numerical computation.

The Anatomy of Functions in MATLAB

- When working with MATLAB's library of built-in math functions, we will use a familiar pattern to calculate values using these functions:



- When calling a function in the Command Window, the argument of the function can take multiple forms including:

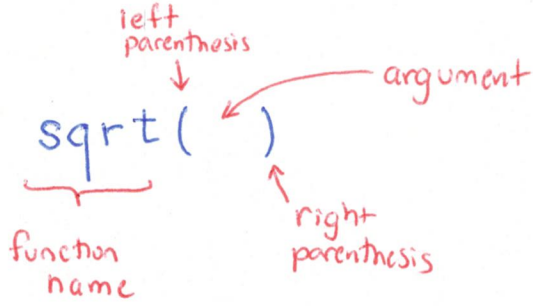
  A. A scalar (number)

  B. a variable that has been assigned a numeric value

  C. a computable expression made up of numbers or variables

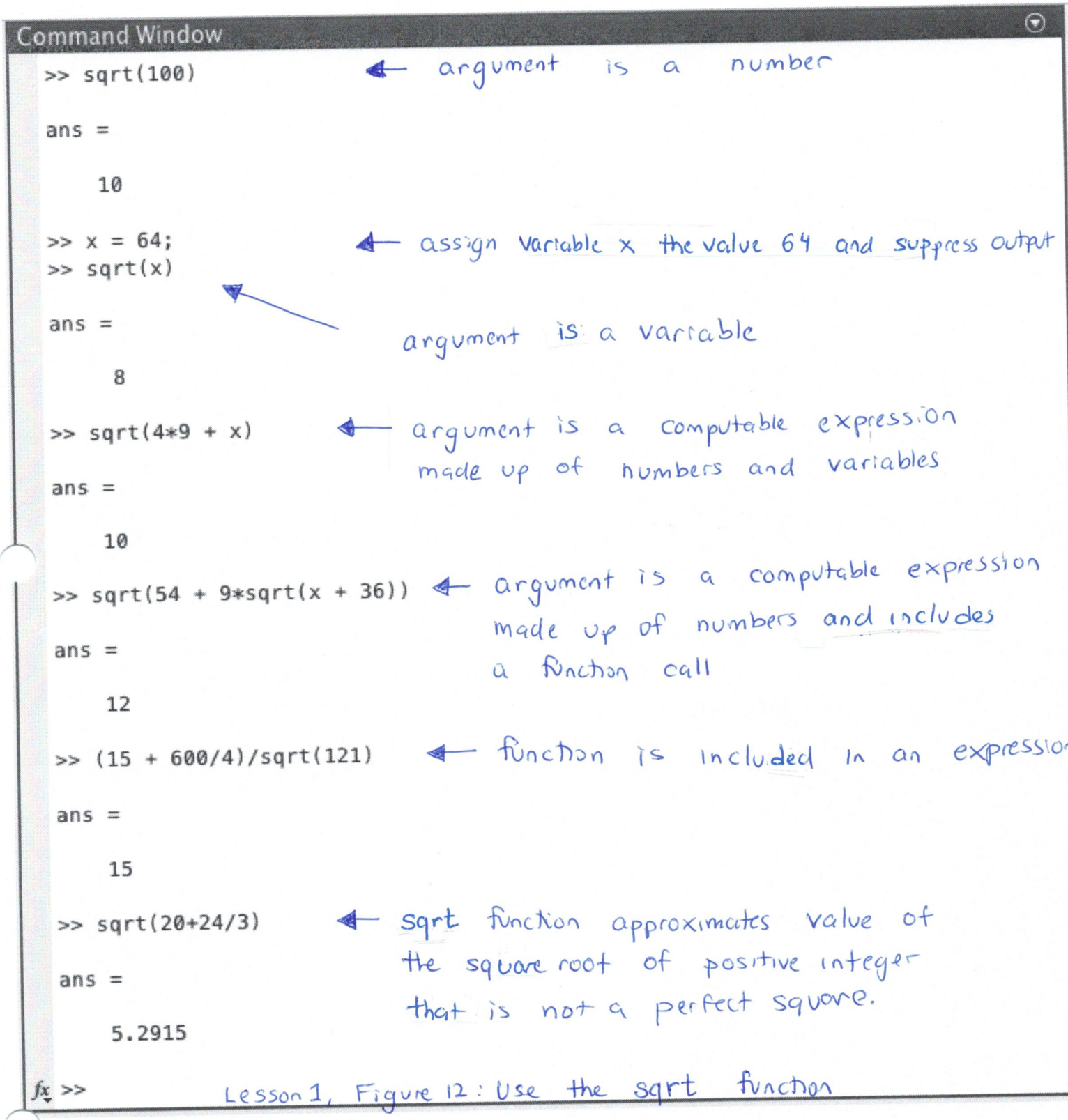
  D. a computable expression that includes functions

☐ To begin our investigation of MATLAB's Elementary Math Built-In Functions, we focus on the



function which is designed to calculate the square root of a (complex) number.

Note:



Lesson 1, Figure 12: Use the sqrt function

☐ Below is a list of some commonly used elementary math functions that are built into MATLAB.

TABLE 1-3:

Elementary math functions

Function	Description	Example

`sqrt(x)`	Square root.	`>> sqrt(81)`
		`ans =`
		`9`

`nthroot(x,n)`	Real *n*th root of a real number *x*. (If *x* is negative *n* must be an odd integer.)	`>> nthroot(80,5)`
		`ans =`
		`2.4022`

`exp(x)`	Exponential (e^x).	`>> exp(5)`
		`ans =`
		`148.4132`

`abs(x)`	Absolute value.	`>> abs(-24)`
		`ans =`
		`24`

`log(x)`	Natural logarithm.
	Base *e* logarithm (ln).
		`>> log(1000)`
		`ans =`
		`6.9078`

`log10(x)`	Base 10 logarithm.	`>> log10(1000)`
		`ans =`
		`3.0000`

`factorial(x)`	The factorial function *x*!
	(*x* must be a positive integer.)
		`>> factorial(5)`
		`ans =`
		`120`

TABLE 1-4:

Trigonometric math functions

Function	Description	Example

`sin(x)`
`sind(x)`
	Sine of angle x (x in radians).
	Sine of angle x (x in degrees).
		`>> sin(pi/6)`
		`ans =`
		`0.5000`

`cos(x)`
`cosd(x)`
	Cosine of angle x (x in radians).
	Cosine of angle x (x in degrees).
		`>> cosd(30)`
		`ans =`
		`0.8660`

`tan(x)`
`tand(x)`
	Tangent of angle x (x in radians).
	Tangent of angle x (x in degrees).
		`>> tan(pi/6)`
		`ans =`
		`0.5774`

`cot(x)`
`cotd(x)`
	Cotangent of angle x (x in radians).
	Cotangent of angle x (x in degrees).
		`>> cotd(30)`
		`ans =`
		`1.7321`

The inverse trigonometric functions are `asin(x)`, `acos(x)`, `atan(x)`, `acot(x)` for the angle in radians; and `asind(x)`, `acosd(x)`, `atand(x)`, `acotd(x)` for the angle in degrees. The hyperbolic trigonometric functions are `sinh(x)`, `cosh(x)`, `tanh(x)`, and `coth(x)`. Table 1-4 uses `pi`, which is equal to π (see Section 1.6.3).

TABLE 1-5:

Rounding functions

Function	Description	Example

`round(x)`	Round to the nearest integer.	`>> round(17/5)`
		`ans =`
		`3`

`fix(x)`	Round toward zero.	`>> fix(13/5)`
		`ans =`
		`2`

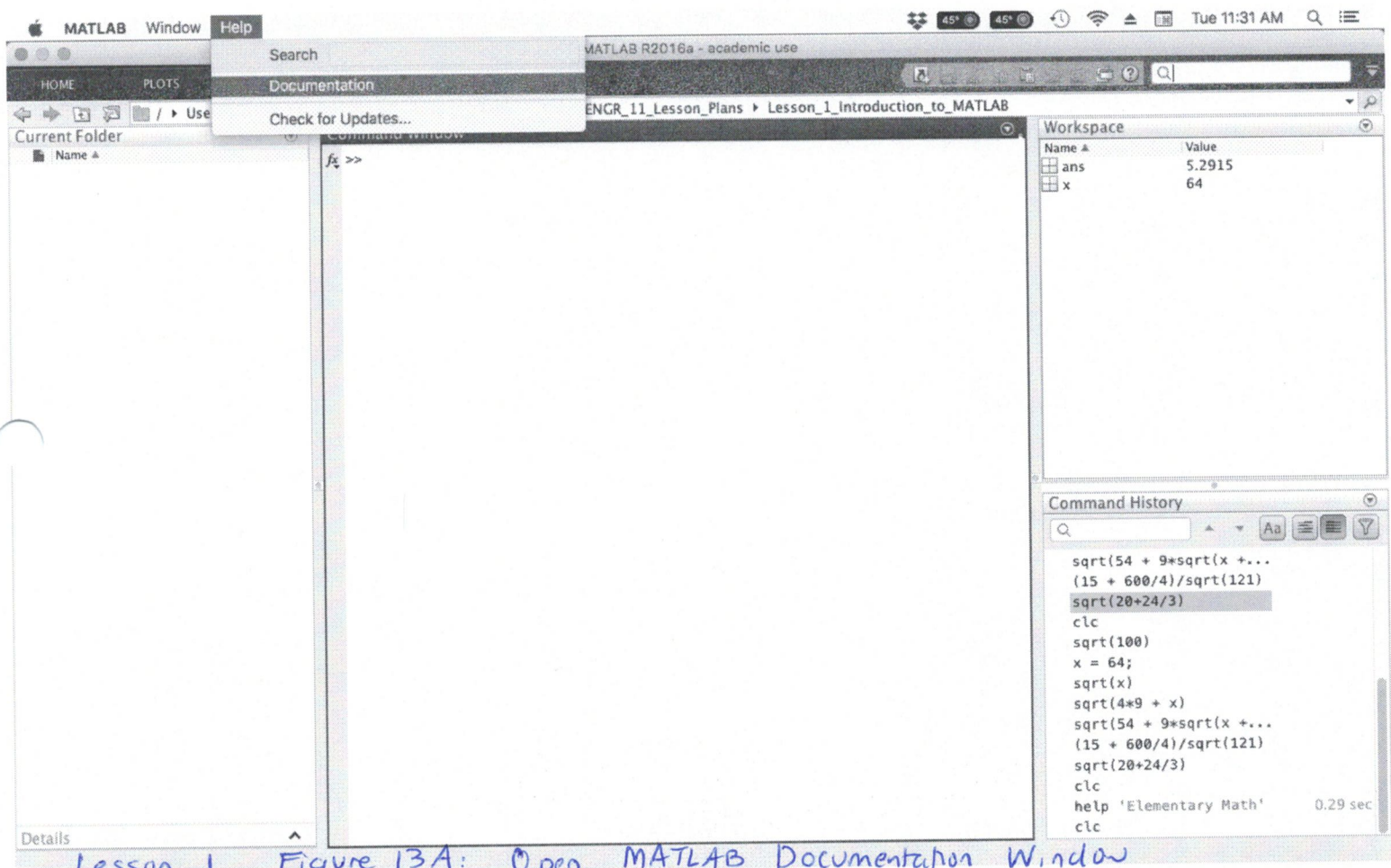
`ceil(x)`	Round toward infinity.	`>> ceil(11/5)`
		`ans =`
		`3`

`floor(x)`	Round toward minus infinity.	`>> floor(-9/4)`
		`ans =`
		`-3`

`rem(x,y)`	Returns the remainder after x is divided by y.	`>> rem(13,5)`
		`ans =`
		`3`

`sign(x)`	Signum function. Returns 1 if $x > 0$, -1 if $x < 0$, and 0 if $x = 0$.	`>> sign(5)`
		`ans =`
		`1`

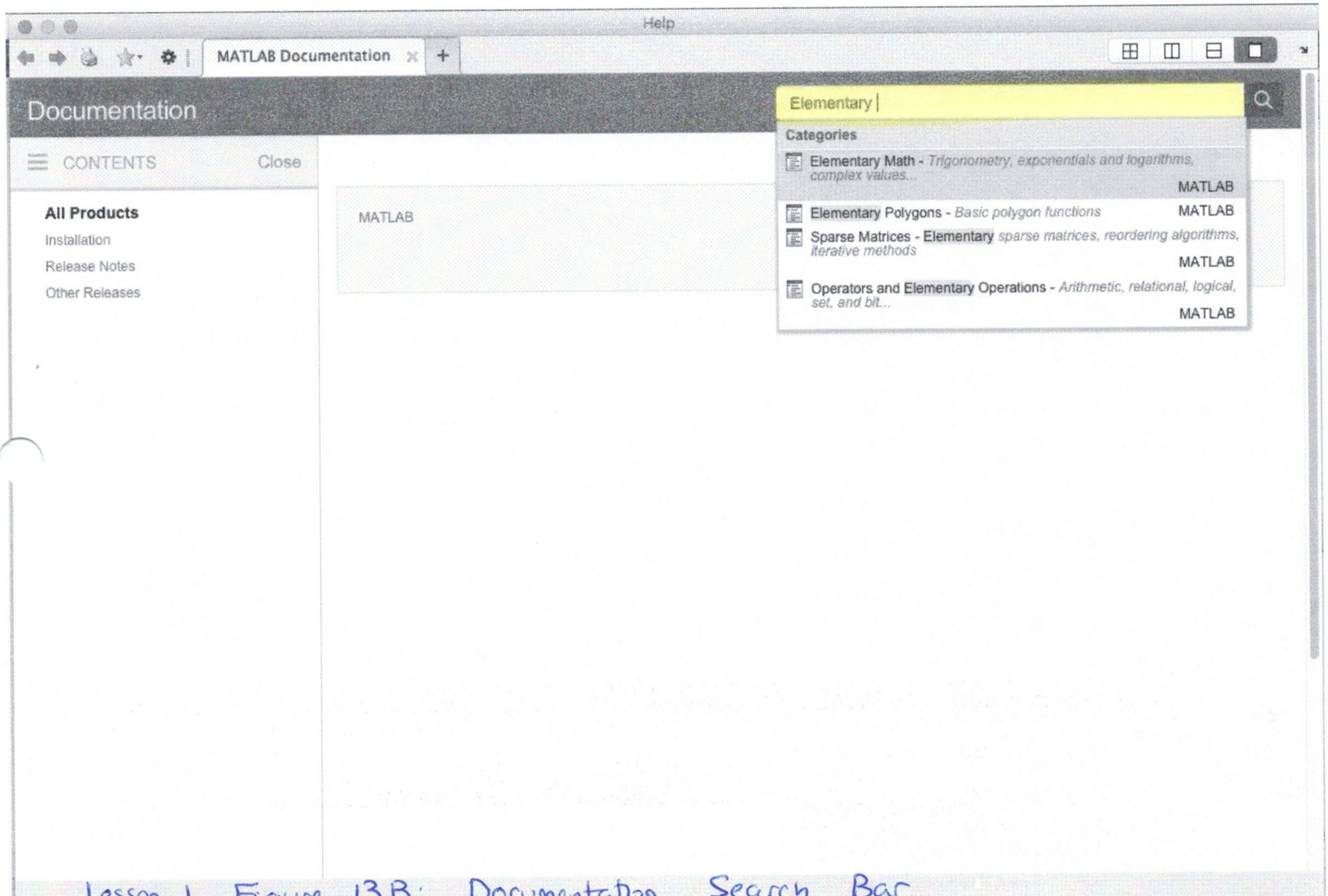
☐ While this list of elementary math functions provides a good start for getting to know MATLAB's environment, it is shortened for the sake of brevity. To access a complete list of functions, we can search MATLAB's Documentation via the Help menu.



Lesson 1, Figure 13A: Open MATLAB Documentation Window

☐ To do so, click on the Help menu, hover down over Documentation and left click.

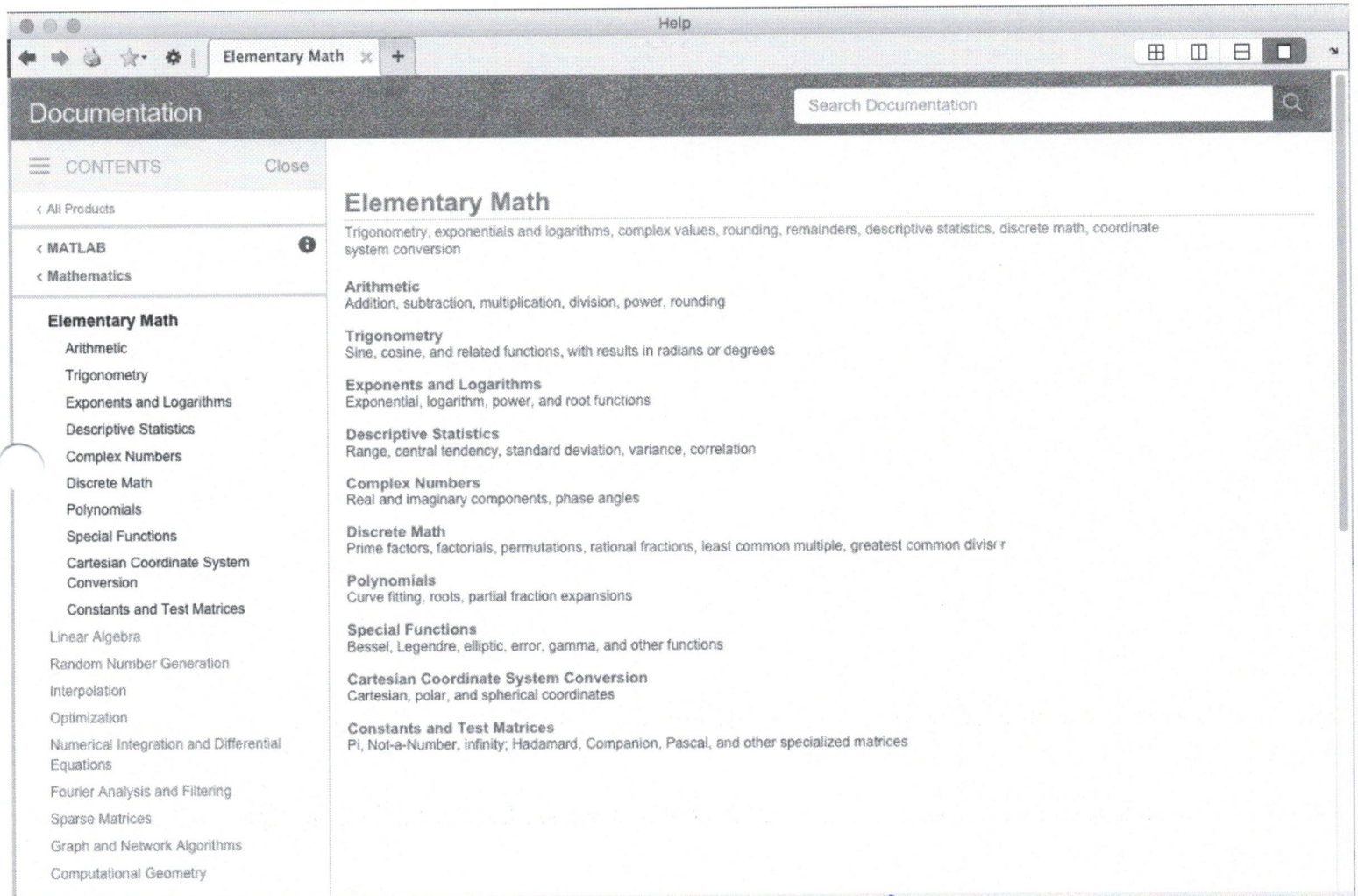
- Upon opening MATLAB's Documentation, a dedicated window will appear on screen. We can then type in our query to the search bar.



Lesson 1, Figure 13B: Documentation Search Bar

- Here, we type in the words 'Elementary Math'. MATLAB will automatically provide a list of relevant topics to our search. In our case, let's click on the first suggestion titled 'Elementary Math'.

☐ We gain access to a complete list of MATLAB's built-in Elementary Math Functions organized by category (i.e. Arithmetic, Trigonometry, Exponents and Logarithms, etc.)



Lesson 1, Figure 1BC: Elementary Math Documentation

☐ We can click on any of the category names to view a complete list of all functions that belong to each category. For example, on the next page, lets view the Exponents and Logarithms functions

Exponents and Logarithms

Exponential, logarithm, power, and root functions

Functions

`exp`	Exponential
`expm1`	Compute exp(x)-1 accurately for small values of x
`log`	Natural logarithm
`log10`	Common logarithm (base 10)
`log1p`	Compute log(1+x) accurately for small values of x
`log2`	Base 2 logarithm and dissect floating-point numbers into exponent and mantissa
`nextpow2`	Exponent of next higher power of 2
`nthroot`	Real nth root of real numbers
`pow2`	Base 2 power and scale floating-point numbers
`reallog`	Natural logarithm for nonnegative real arrays
`realpow`	Array power for real-only output
`realsqrt`	Square root for nonnegative real arrays
`sqrt`	Square root

Examples and How To

Powers and Exponentials

Defining Scalar Variables

- As we will see throughout this course, we will often want to store the results of commands executed by MATLAB as variables* in memory. To do so, we will use the assignment operator (=).

* Notes: More about the term variable

- Just like our use of the term function, here again we are overloading (and abusing) the word variable because we are using this one word to represent two different ideas:

  ① a "variable" in mathematics is a symbol that represents a value (or number) and can be used to express algebraic and analytic relationships (as you did in your algebra and calculus classes)

(2) a "variable" in computer science (and programming) has three important attributes including:

i. a specific storage location identified by a memory address in our computer's memory system (this includes a pre-defined fixed-length sequence of binary digits as well as a pointer to the exact location to this sequence of bits)

ii. a user-defined, symbolic name that can be used as a reference. In MATLAB, variable names will be created using a mixture of letters and digits. We will also follow a set of rules and guidelines when assigning variable names.

iii. an assigned value that can be bound to the variable when executing commands and can be changed during the course of executing a program in MATLAB

- When we define a new variable in MATLAB, a few things happen:

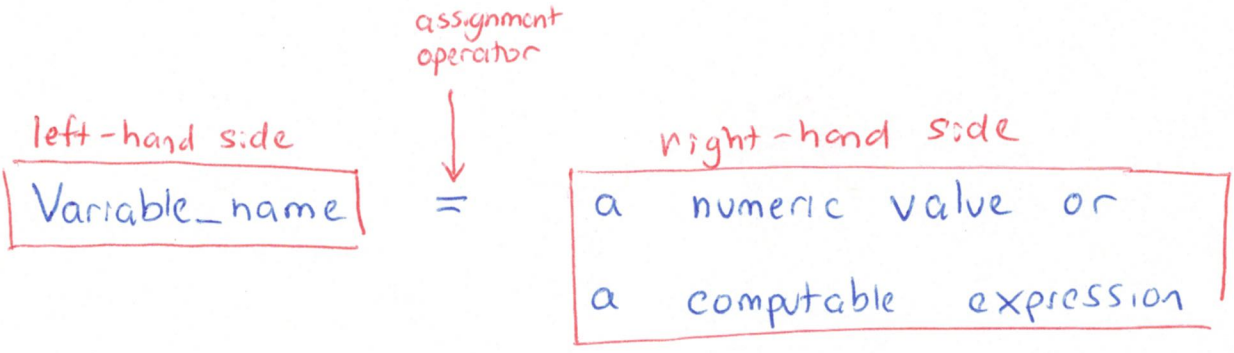
  First: MATLAB allocates appropriate memory space to that variable (a sequence of bits in memory) where the value of that variable is to be stored and associates the address of this memory with the variable name

  Second: MATLAB executes any commands necessary to obtain the value to be stored in memory by translating this value into the appropriate binary representation and stores this value in memory

- In this lesson, we focus only on assigning scalar values to variables. In Lesson 2, we will learn to assign and address arrays as variables.

- As we discussed before, when we use the = sign in MATLAB, we call this the assignment operator.

- The assignment operator "assigns" a value to a variable with a predictable pattern

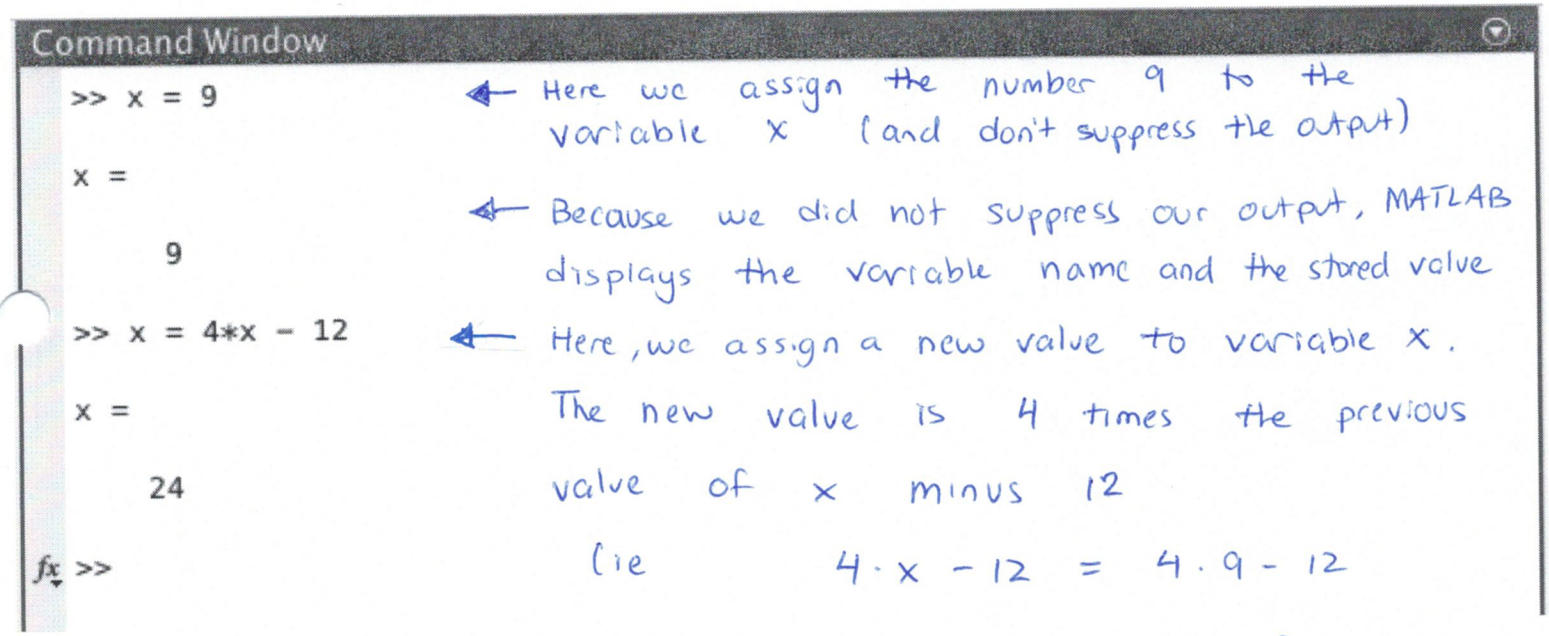


To assign scalar-valued numeric data to variables using =

- The Left-hand side of the assignment operator can only include a single variable name (that follows a specific set of rules).

- The right-hand side of the assignment operator can be a number or a computable* expression that includes numbers, arithmetic operations, elementary math functions, or variables that were previously assigned numerical values

- After we've created a command to assign a numerical value (or computable expression) to a variable and press the Enter key, MATLAB allocates memory, stores the desired value in memory, and assigns the location in memory to our variable name



- Notice that the command x = 4 * x - 12 uses the assignment operator in a very different context than the equals sign in mathematics

☐ If we interpret the command

$$x = 4 * x - 12$$

as a mathematical equation from algebra, then solving for the value of x that makes this equation true yields

$$x = 4x - 12$$

$$\Rightarrow \quad 3x = 12$$

$$\Rightarrow \quad \boxed{x = 4}$$ ← This is not the value assigned to variable x, as we saw

☐ Thus, we should remember that the = sign is used to assign values to specific variables.

Rules about Variable Names

When we decide to assign names to variables, there are a specific set of rules that we must follow. These rules are pre-defined by MATLAB. More specifically, a variable name:

1. must begin with a letter
2. can be up to 63 characters long (try typing namelengthmax in command window)
3. Can contain uppercase and lowercase letters, decimal digits (0, 1, 2, ..., 9), and the underscore character
4. Cannot contain punctuation characters. like period, comma, semicolon, colon, exclamation mark, question mark, etc.
5. Cannot contain spaces between characters (but we can use underscore where a space is desired)
6. Cannot be defined as MATLAB's keywords that are reserved for other reasons (see next page for more)

☐ More about Rule # 6: Keywords and Pre defined Variables

6A. MATLAB Keywords :

try typing the command
iskeyword
in the command window

- There are a list of 20 words, known as MATLAB's Keywords, that MATLAB reserves for various purposes.

- Thus, MATLAB does not allow us to use these words as variable names

- The 20 key words (in alphabetical order) are:

1. break
2. case
3. catch
4. classdef
5. continue
6. else
7. elseif
8. end
9. for
10. function
11. global
12. if
13. otherwise
14. parfor
15. persistent
16. return
17. spmd
18. switch
19. try
20. while

- Below we see the result of Entering the

  iskeyword

  command into the command prompt

Command Window

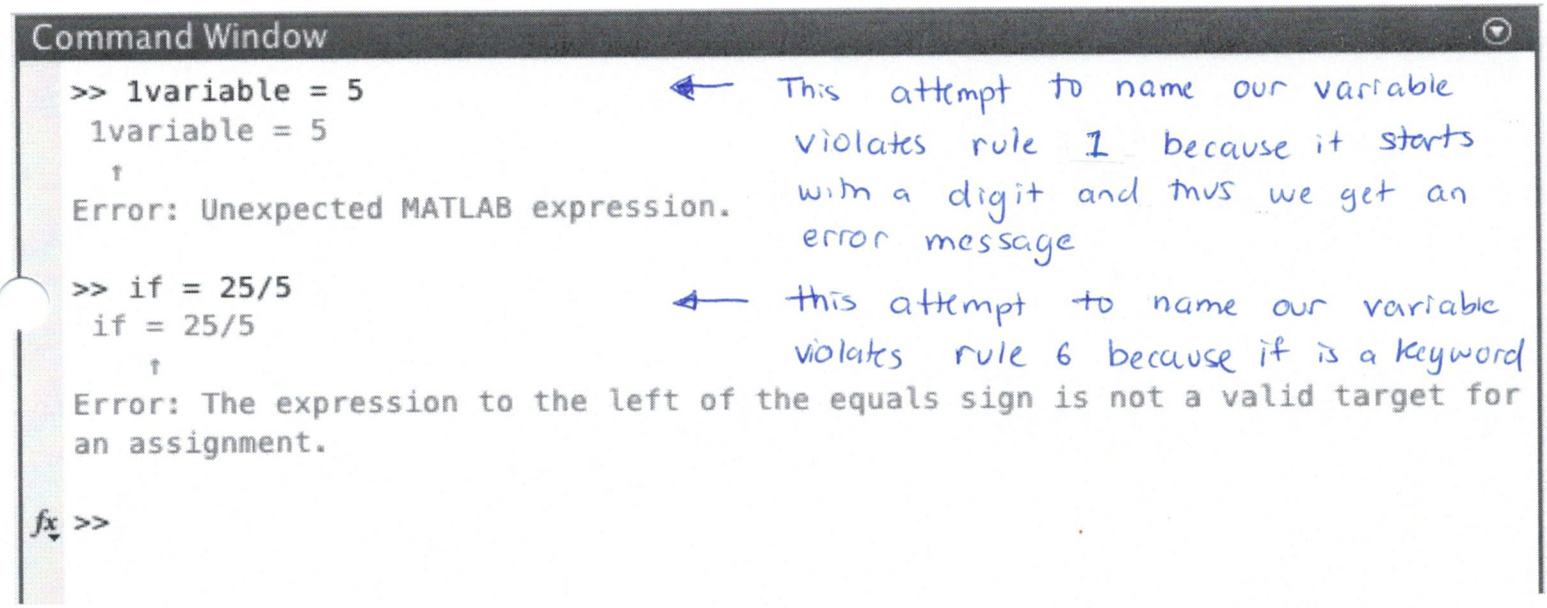
```
>> iskeyword

ans =

    'break'
    'case'
    'catch'
    'classdef'
    'continue'
    'else'
    'elseif'
    'end'
    'for'
    'function'
    'global'
    'if'
    'otherwise'
    'parfor'
    'persistent'
    'return'
    'spmd'
    'switch'
    'try'
    'while'

fx >>
```

- These 6 rules for naming variables have dire consequences when we violate these specifications. For example, In the code below, we violate rules 1 and 6 resulting in Error messages



- Not all violations of these rules will cause error messages. For example, if we try to use a period in our variable name, we will be defining fields in a data structure which is an useful tool but not designed for simple variable names...

Guidelines for Variable Names

In addition to the hard rules for naming variables, there are some general guidelines to keep in mind when creating variable names; including:

A. Avoid using the name of built-in functions for a variable (i.e. avoid using sin, cos, sqrt, exp, log, size, char, path, etc.). In general, variable names take precedence over function names meaning that once a function name is used to name a variable, the function cannot be used (which can lead to unexpected results).

- One way to check whether a proposed name for a variable is already in use is to use the

  exist

  function.

Command Window

```
>> exist sin

ans =

     5

>> exist exp

ans =

     5

>> exist checkname

ans =

     0

>> a = 4;
>> exist a

ans =

     1

fx >>
```

- exist returns 0 if there are no existing variables, functions, or other affiliates with the proposed name (meaning we can safely assign the variable to this name).

Guidelines for variable names, continued...

B. We should be aware that MATLAB maintains a list of predefined variables that are not keywords yet are already defined when MATLAB starts up. Some (but not all) predefined variables are listed in the table below.

Expression	Description
`pi`	The number π up to 15 significant digits.
`i, j`	The complex number $\sqrt{-1}$.
`inf`	Represents the mathematical Infinity concept, for example, a result of division by zero.
`aN`	Stands for Not-A-Number. Represents the result of a meaningless mathematical function, like 0/0.
`clock`	Contains the current date and time in the form of a 6-element row vector: year,month,day,hour,minute,second.
`date`	Contains a string representing today's date.
`eps`	Stands for **epsilon**. It represents the smallest number that can be represented by your MATLAB software.
`ans`	A special variable that MATLAB uses to store the result of MATLAB's command line.

- These variables can be redefined to have any value and we can change these to use them for our own purposes while coding

- However, we should be very careful when working with predefined variables. In general, its best not to use these to avoid confusion.

☐ In general, we should not redefine variables pi, eps, inf because these arise frequently in many applications

Command Window

```
>> pi

ans =

    3.1416

>> i

ans =

   0.0000 + 1.0000i

>> inf

ans =

   Inf

>> NaN

ans =

   NaN

>> date

ans =

16-Oct-2018

>> eps

ans =

   2.2204e-16

fx >>
```

☐ The predefined variables i and j can be redefined in common association with for loops when complex numbers are not involved

Guidelines for variable names, continued...

C. MATLAB is case-sensitive and distinguishes between upper case and lower case letters. For example, AA, Aa, aA, and aa are four different variable names. We see evidence of this feature in the commands below.

Command Window

```
>> a = 3;
>> A = 15;
>> A - a

ans =

    12
```

← assign lowercase a value of 3

← assign uppercase A value of 15

← MATLAB has stored different values in variables a and A. Thus a - A ≠ 0

Useful Commands for Managing Variables

- When we assign variables in MATLAB, we have already seen how to get a quick look at these variables in the workspace Window.

- However, we can also use commands to manage variables in the command window. Below is a list of such commands. We can type any of these into the command window and press Enter

Command	Outcome
`clear`	Removes all variables from the memory.
`clear x y`	Removes only variables x, y, and z from the memory.
`who`	Displays a list of the variables currently in the memory.
`whos`	Displays a list of the variables currently in the memory and their sizes together with information about their bytes and class (see Section 4.1).

- These commands are designed to enable us to obtain information about variables we've created or eliminate variables from the workspace.

☐ Let's say we want to use MATLAB to solve the quadratic equation: $2x^2 - 5x - 3 = 0$

Below we see how to produce our two solutions

Command Window

```
>> clear          ← removes all variables from memory
>> a = 2

a =

     2

>> b = -5

b =

    -5

>> c = -3

c =

    -3

>> x1 = (-b - sqrt(b^2 - 4*a*c))/(2*a)

x1 =

   -0.5000

>> x2 = (-b + sqrt(b^2 - 4*a*c))/(2*a)

x2 =

     3

>> who            ← displays a list of (the names of) the variables
                     currently stored in memory
Your variables are:

a   b   c   x1  x2

fx >>
```

☐ Notice how many lines in our command window that this solution required.

☐ We can accomplish the same task in fewer lines. Specifically, we can assign many variables in a single line. Each assignment must be separated by a comma or semicolon (and spaces can be added after the comma or semicolon). When we press Enter, assignments are executed left to right.

Command Window

```
>> clear
>> a = 2, b = -5; c = -3, x1 = (-b - sqrt(b^2 - 4*a*c))/(2*a);...
x2 = (-b + sqrt(b^2 - 4*a*c))/(2*a)

a =

     2


c =

    -3


x2 =

     3

>> whos
  Name        Size                    Bytes  Class     Attributes

  a           1x1                         8  double
  b           1x1                         8  double
  c           1x1                         8  double
  x1          1x1                         8  double
  x2          1x1                         8  double

>>
```

clear ← removes all variables from memory

output suppressed for b and x1 since these end in a semicolon

output displayed for a, c, and x2 since these use a comma or do not have a semicolon

whos ← displays a list of variables in memory plus information about their size, class and attributes

☐ When executing multiple assignments in a single line, all assignments that end in a comma will be displayed in the command window. On the otherhand, if we use a semicolon (rather than a comma) the output will be suppressed.

☐ Finally, a variable that already exists in memory can be reassigned (or overwritten) with a new value using the assignment operator.