

Basic MATLAB Programming

Chul Min Yeum

Assistant Professor

Civil and Environmental Engineering
University of Waterloo, Canada

AE121: Computational Method



UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

Last updated: 2019-04-29

Variables and Assignments

- To store a value, use a *variable*
- One way to put a value in a variable is with an *assignment statement*
- General form:
variable = expression
- The order is important
 - **variable name on the left**
 - the assignment operator “=” (Note: this does NOT mean equality)
 - expression on the right

Variables and Assignments



Modifying Variables

- *Initialize* a variable (put its first value in it)

```
mynum = 5;
```

- Change a variable (e.g. by adding 3 to it)

```
mynum = mynum + 3;
```

- *Increment* by one

```
mynum = mynum + 1;
```

- *Decrement* by two

```
mynum = mynum - 2;
```

NOTE: after this sequence, *mynum* would have the value 7 (5+3+1-2)

Example: Arithmetic Operation in MATLAB

If the car has a mass of 300kg and you push the car with an acceleration of 0.5 in/s^2 , compute the force that is generated from the car in newton

```
% solve this problem using MATLAB as a calculator  
300*5*0.0254
```

```
ans = 38.1000
```

```
% solve this problem using MATLAB as a programming tool  
inch2m = 0.0254; % inch to m  
mass = 300; % kg  
acceleration = 5; % inch/s/s  
  
% force = mass(kg) * acceleration (m/s/s)  
force1 = mass * acceleration * inch2m;  
force1
```



```
force1 = 38.1000
```

Which approach is better?

- Names must begin with a letter of the alphabet
- After that names can contain letters, digits, and **the underscore character(_)**
- MATLAB is **case-sensitive**
- Names should be **mnemonic** (they should make sense!)
- The commands **who** and **whos** will show variables (**Recommend: You would need to use a workspace browser rather than type these command**)
- To delete variables: **clearvars**
- **clear** clears out variables and also functions

Example: Variable Names

```
% 8val = 10; % error
% comment: must begin with a letter of the alphabet

% _col = 0; % error
% comment: must begin with a letter of the alphabet

% val = 10;
% Val % error because you don't define the variable of 'Val'
% comment: case-sensitive
% recommendation: do not use a capital letter for variable names

% row_3 = 1;
% row@3 = 10; % error
% col_-3 = 10; % error
% comment: can only contain the underscore (among non-alphabet characters)

% asdf1 = 10; % no error
% love10 = 10; % no error
% aaaa1 = 100; % no error
% comment: they work but names should be mnemonic
```

- Every expression and variable has an associated *type*, or *class*
 - Real numbers: **single**, **double**
 - Integer types: numbers in the names are the number of bits used to store a value of that type
 - Signed integers: **int8**, **int16**, **int32**, **int64**
 - Unsigned integers: **uint8**, **uint16**, **uint32**, **uint64**
 - Single characters and character vectors: **char**
 - Strings of characters: **string**
 - True/false: **logical**
- The default type for numbers is **double**

Range and Type Size

- Range of integer types found with **intmin/intmax**
 - e.g. **intmin('int8')** is -128, **intmax('int8')** is 127
- Converting from one type to another, using any of the type names as a function, is called *casting* or *type casting*, e.g:

1 byte	8 bits
1 kilobyte	1024 byte
1 megabyte	1024 kilobyte
1 gigabyte	1024 megabyte
1 terabyte	1024 gigabyte

```
clear; clc;
|
intmin('uint8')
```

```
ans = uint8
      0
intmax('uint8')
```

```
ans = uint8
      255
```

```
val1_uint8 = uint8(10);
val_double = 10;
whos % or see the workspace browser
```

Name	Size	Bytes	Class	Attributes
ans	1x1	1	uint8	
val1_uint8	1x1	1	uint8	
val_double	1x1	8	double	

Constants

- In programming, variables are used for values that **could change**, or **are not known** in advance
- **Constants** are used when the value is known and is not updated in the program.
- Examples in MATLAB (these are actually functions that return constant values)
 - pi** 3.14159....
 - i, j**
 - inf** infinity
 - NaN** stands for “not a number”; e.g. the result of 0/0

Example: Constants - Common Mistakes

```
pi; % pi (a circle's circumference to its diameter)  
val = 2 + 3i; % complex number
```

% comment: I highly recommend that you should avoid constant names as variable
% names. For example, when you define a for-loop, people often use i or j as a
% iterator. I always use ii or jj not to overwrite constants.

```
pi
```

```
ans = 3.1416
```

```
pi = 3;  
pi % no error but it's a really bad practice. You can't use the pi constant anymore.
```

```
pi = 3
```

Expressions

- Expressions can contain values, variables that have already been created, operators, built-in functions, and parentheses
- Operators include:
 - + addition
 - * multiplication
 - / division (divided by e.g. $10/5$ is 2)
 - \ division (divided into e.g. $5\backslash 10$ is 2)
 - \wedge exponentiation (e.g. 5^2 is 25)
- Scientific or exponential notation: use e for exponent of 10 raised to a power
 - e.g. $3e5$ means $3 * 10^5$ (e becomes a operators).
 - **Not recommend** to use e because it is confused with exponential (exp) in your code.

Operator Precedence

- Some operators have precedence over others
- Precedence list (highest to lowest) so far:
 - () parentheses
 - ^ exponentiation
 - negation
 - *, /, \ all multiplication and division
 - +,- addition and subtraction
- Nested parentheses: expressions in inner parentheses are evaluated first

Using Functions: Terminology

- To use a function, you *call* it
- To call a function, give its name followed by the *argument(s)* that are *passed* to it in parentheses
- Many functions calculate values and *return* the results
- For example, to find the absolute value of -4

```
>> abs(-4)  
ans =  
4
```

- The name of the function is “abs”
- One argument, -4, is passed to the **abs** function
- The **abs** function finds the absolute value of -4 and returns the result, 4

Random Numbers

- Several built-in functions generate random (actually, pseudo-random) numbers
- Random number functions, or random number generators, start with a number called the *seed*; this is either a predetermined value or from the clock
- By default MATLAB uses a predetermined value so it will always be the same
- To set the seed using the built-in clock:

```
rng('shuffle')
```

```
>> rand  
  
ans =  
  
0.0558  
  
>> rand  
  
ans =  
  
0.4968
```

NIST's New Quantum Method Generates *Really* Random Numbers

April 11, 2018

Researchers at the National Institute of Standards and Technology (NIST) have developed a method for generating numbers guaranteed to be random by quantum mechanics.

[Described in the April 12 issue of *Nature*](#), the experimental technique surpasses all previous methods for ensuring the unpredictability of its random numbers and may enhance security and trust in cryptographic systems.



MEDIA CONTACT

Laura Ost
laura.ost@nist.gov
(303) 497-4880



ORGANIZATIONS

[Physical Measurement Laboratory](#)
[Applied Physics Division](#)
[Faint Photonics Group](#)
[Quantum Nanophotonics Group](#)

Random Real Numbers

- The function **rand** generates uniformly distributed random real numbers in the open interval $(0, 1)$
- Calling it with no arguments returns one random real number
- To generate a random real number in the open interval $(0, N)$:
rand * N
- **randn** is used to generate normally distributed random real numbers

val1 = 10;

val2 = 11;

Ex) write a program to add 'val1' and 'val2' and assign a result to 'val3'

val3 = val1 + val2;

val3 = 21; **hard coding**

This random number generator is to avoid your hard-coding in your labs and assignments because we do not know what values are used to check your code

Example: Random Numbers

```
rng('shuffle'); % a different sequence of random number is generated. It's default in Matlab.  
rand  
rand  
rand  
% comment: Every time you run 'rand', different random numbers in the open  
% interval (0,1) are generated. However, it is not "real" random number. They  
% just refer a "pseudo" random number table corresponding the current time.  
  
% doc rng  
rng(10); rand % if you set a seed (table number), a fixed number is generated.
```

Characters and Strings

- A ***character*** is a single character in single quotes
- All characters in the computer's character set are put in an order using a ***character encoding***
 - In the character encoding sequence, the letters of the alphabet are in order, e.g. 'a' comes before 'b'
 - Common encoding ASCII has 128 characters, but MATLAB can use a much larger encoding sequence
- The ***character set*** includes all letters of the alphabet, digits, punctuation marks, space, return, etc.
- Character vectors are sequences of characters in single quotes, e.g. 'hello and how are you?'
- ***Strings*** are sequences of characters in double quotes, e.g. "ciao bella"

Characters and Encoding

- Standard ASCII has 128 characters; integer equivalents are 0-127
- Any number function can convert a character to its integer equivalent

```
>> numequiv = double('a')
```

```
numequiv =
```

```
97
```

- The function **char** converts an integer to the character equivalent (e.g. **char(97)**)
- MATLAB uses an encoding that has 65535 characters; the first 128 are equivalent to ASCII

```
name_cm_char = 'Chul Min Yeum';  
size(name_cm_char)
```

```
ans = 1x2  
      1     13
```

```
% comment: name_cm_char is a set of characters. It's an array
```

```
name_cm_str = "Chul Min Yeum";  
size(name_cm_str)
```

```
ans = 1x2  
      1     1
```

```
% comment: name_cm_str is a "string" type. It's a value.
```

Output

- There are two basic output functions:
 - **disp**, which is a quick way to display things
 - **fprintf**, which allows formatting
- The **fprintf** function uses **format specifiers** which include **place holders**; these have **conversion characters**:
 - %d integers
 - %f floats (real numbers)
 - %c single characters
 - %s string of characters
- Use %#x where # is an integer and x is the conversion character to specify the **field width** of #
- %#.#x specifies a field width and the number of decimal places
- %.#x specifies just the number of decimal places (or characters in a string); the field width will be expanded as necessary

Example: Field & Numeric Formatting

```
x = 1234.56789;  
fprintf('case1: x is %f \n', x);  
fprintf('case2: x is %9.4f \n', x);  
fprintf('case3: x is %9.1f \n', x);  
fprintf('case4: x is %.1f \n', x);  
fprintf('case5: x is %6.1f \n', x);  
fprintf('case6: x is %10.6f \n', x);  
fprintf('case7: x is %10.10f \n', x);
```

```
case1: x is 1234.567890  
case2: x is 1234.5679  
case3: x is 1234.6  
case4: x is 1234.6  
case5: x is 1234.6  
case6: x is 1234.567890  
case7: x is 1234.5678900000
```

Formatting Output

- Other formatting:
 - \n newline character
 - \t tab character
 - left justify with ‘-’ e.g. %-5d
 - to print one slash: \\
 - to print one single quote: “ (two single quotes)
- Printing vectors and matrices: usually easier with **disp**

Example: Print Formatting

```
fprintf('Hello! Chul Min');
fprintf('Hello! \tChul Min');
fprintf('Hello! \nChul Min');
fprintf('Hello! AE121');
fprintf('Hello! %cE121', 'A');
fprintf('Hello! %s', 'AE121'); % string of characters
fprintf('Today is 04\\23\\2019.');?>
fprintf('What is ''fprintf'''?'); % two single quotes to print one quote
% Please copy above lines and paste them in the command window.
%
% comment: all outputs are placed on the same line (except the third one).
% This is because the live editor print the output based on the line of the
% code but the command window (and editor) run them without adding new line.
% Please add \n at the end of your text to print ensuing output text on
% next line.
```

```
Hello! Chul Min
Hello! Chul Min
Hello!
Chul Min
Hello! AE121
Hello! AE121
Hello! AE121
Today is 04\23\2019.
What is 'fprintf'?
```

```
fprintf('Hello! %cE121', 'A');
fprintf('Hello! %s', 'AE121'); % string of characters
fprintf('Today is 04\\23\\2019.');?>
fprintf('What is ''fprintf'''?'); % two single quotes to print one quote
Hello! Chul MinHello! Chul MinHello!
Chul MinHello! AE121Hello! AE121Hello! AE121Today is 04\23\2019.What is 'fprintf'?>>
```

Relational Expressions

- The relational operators in MATLAB are:
 - > greater than
 - < less than
 - >= greater than or equals
 - <= less than or equals
 - == equality
 - ~= inequality
- The resulting type is **logical** 1 for true or 0 for false
- The logical operators are:
 - || or for scalars
 - && and for scalars
 - ~ not
- Also, **xor** function which returns logical true if only one of the arguments is true

Very important !!!

Truth Table

- A truth table shows how the results from the logical operators for all combinations

x	y	$\sim x$	$x \parallel y$	$x \&& y$	$\text{xor}(x,y)$
true	true	false	true	true	false
true	false	false	true	false	true
false	false	true	false	false	false

- Note that the logical operators are commutative (.e.g., $x \parallel y$ is equivalent to $y \parallel x$)

Example: Truth Table

```
x = true; % x = logical(1);
y = true; % y = logical(1)
fprintf('x is %d and y is %d. \n', x, y);
fprintf(~x: %d, x||y: %d, x&&y: %d, and(x,y): %d, xor(x,y): %d.', ~x, x||y, x&&y, and(x,y), xor(x,y));

x = true; % x = logical(1);
y = false; % y = logical(0)
fprintf('x is %d and y is %d. \n', x, y);
fprintf(~x: %d, x||y: %d, x&&y: %d, and(x,y): %d, xor(x,y): %d.', ~x, x||y, x&&y, and(x,y), xor(x,y));

x = false; % x = logical(0);
y = false; % y = logical(0)
fprintf('x is %d and y is %d. \n', x, y);
fprintf(~x: %d, x||y: %d, x&&y: %d, and(x,y): %d, xor(x,y): %d.', ~x, x||y, x&&y, and(x,y), xor(x,y));

% how about this?
x = 1;
y = -1;
fprintf('x is %d and y is %d. \n', x, y);
fprintf(~x: %d, x||y: %d, x&&y: %d, and(x,y): %d, xor(x,y): %d.', ~x, x||y, x&&y, and(x,y), xor(x,y));
% comment: it is the same with the result when x and y are true. Basically,
% logical values corresponding all values except 0 are true.
```

x is 1 and y is 1.

~x: 0, x||y: 1, x&&y: 1, and(x,y): 1, xor(x,y): 0.

x is 1 and y is 0.

~x: 0, x||y: 1, x&&y: 0, and(x,y): 0, xor(x,y): 1.

x is 0 and y is 0.

~x: 1, x||y: 0, x&&y: 0, and(x,y): 0, xor(x,y): 0.

x is 1 and y is -1.

~x: 0, x||y: 1, x&&y: 1, and(x,y): 1, xor(x,y): 0.

Operator Precedence

Parentheses ()

Transpose (.'), power (.^), complex conjugate transpose ('), matrix power (^)

Multiplication (. *), right division (./), left division (\.), matrix multiplication (*), matrix right division (/), matrix left division (*)

Addition (+), subtraction (-)

Colon operator (:)

Less than (<), less than or equal to (<=), greater than (>), greater than or equal to (>=), equal to (==), not equal to (~=)

Short-circuit AND (&&)

Short-circuit OR (||)

https://www.mathworks.com/help/matlab/matlab_prog/operator-precedence.html

Example: Good Practice for Arithmetic Operations

I recommend that you always use paranethese to avoid potential mistakes regardless of their operator precedence. For example

$$y = \frac{x^2(100x + 10) + x^3(20x^2 + 3)}{-x^{-3} + 1}$$

```
x = 10;
y1 = (x^2*(100*x + 10) + x^3*(20*x^2 + 3))/(-x^(-3)+1);
y1
```

Tip: However, the more the number of terms and parantheses in your equation, the larger is the probability that you would be making a mistake. In this case, I usually do like this:

```
y_nom = (x^2)*(100*x + 10) + (x^3)*(20*(x^2) + 3);
y_den = -x^(-3)+1;
y2 = y_nom/y_den; clearvars y_nom y_den % clear temporary variables
y2
```

Example: Operator Precedence

How to make an expression of ? in other words, how to write a code to check if x lies in between 5 and 10. If yes, 1 and otherwise 0.

```
x = 1;  
disp('x is 1.')  
5 < x < 10  
(5 < x) < 10  
5 < (x < 10)  
(5 < x) & (x < 10)
```

```
x is 1.  
ans = Logical  
1  
ans = Logical  
1  
ans = Logical  
0  
ans = Logical  
0
```

- Confusing the format of an assignment statement (make sure that the variable name is always on the left)
- Forgetting to use parentheses to pass an argument to a function (e.g., typing “fix 2.3” instead of “fix(2.3)”)
- Confusing || and xor
- Using = instead of == for equality
- Using an expression such as “ $5 < x < 10$ ” – which will always be true, regardless of the value of the variable x (because the expression is evaluated from left to right; $5 < x$ is either true (1) or false (0); both 1 and 0 are less than 10)

Programming Style Guidelines

- Use mnemonic variable names (names that make sense; for example, *radius* instead of *xyz*)
- Although variables named *result* and *RESULT* are different, avoid this as it would be confusing. (recommendation: do not use capital letters in your variable names)
- Do not use names of built-in functions as variable names
- Store results in named variables (rather than using *ans*) if they are to be used later

Slide Credits and References

- Stormy Attaway, 2018, Matlab: A Practical Introduction to Programming and Problem Solving, 5th edition
- Lecture slides for “Matlab: A Practical Introduction to Programming and Problem Solving”
- Holly Moore, 2018, MATLAB for Engineers, 5th edition