Introduction to MATLAB and its data model

Includes non textbook material

but also covers Chapter 1 (1.1-1.5), Chapter 2 (2.1-2.2,2.6,2.8) and 3.3.1

Disclaimer: *These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Fall 2013 ; distribution outside this group of students is NOT allowed.*

(c) Copyright A. Gerbessiotis.

As we mentioned in the previous chapter, programs and data are stored in the main memory of a computer. Both program and data are stored as sequences of a collections of binary digits i.e. as a sequence of bytes. The fundamental aggregate is thus the **byte**, a collection of 8 **bits**.

2.1. Machine code. If one wants to program a computer in the absence of helpful tools, he needs to manipulate bits and bytes at a very low level according to a specific computer-dependent format that is known as the **machine code** or **machine language**. This is tedious, complicated, error-prone, confusing and buggy.

For the sake of an example, think of the pretty tedious task of writing a simple integer, say `1` into memory. An integer nowadays is represented by 32 bits (i.e. a word) in two's-complement representation, and thus following the notation and discussion of the previous chapter its binary representation might look like the bit sequence `00000000 00000000 00000000 00000001` with spaces added every eights bits for readability and to highlight the bytes in this 32-bit word. Suppose we intend to store this integer in main memory and let memory locations 10,11,12,13 be used for this purpose. How do we store those 4 bytes in those 4 memory locations, one byte per location? Depending on the CPU or microprocessor, one or the other byte-order shown below might be used. A microprocessor using the former byte-order is called a big-endian processor, as opposed to the alternative; Intel processors are little-endian.

```
Big-Endian                              Little-endian
10       11       12       13          10       11       12       13
00000000 00000000 00000000 00000001    00000001 00000000 00000000 00000000

Byte order for storing 32-bit integer 1 = 00000000 00000000 00000000 00000001
```

2.2. Assembler (Assembly language). If machine code is too complicated and too bit- or byte-oriented, one can avoid to use such low-level manipulation/programming by programming at a higher level of abstraction by using an **assembly language**. An assembly language is a user-friendlier (but not very user-friendly) version of machine code/language in which the user (the programmer) uses mnemonic codes to execute machine code instructions. In other words, the input to an assembly language is a higher-level language (eg. mnemonics) converted into the output language which is machine code. For example instead of using machine code `10101011`, one could use a mnemonic code `STORE A` that writes back into memory the contents of processor register `A` (a register is a small piece of memory located on the CPU that can be accessed very quickly and efficiently).

(c) Copyright A. Gerbessiotis. All rights reserved.

2.3. Compiler. Even assembly language is too low-level for most programmers and end-users. It is also machine-dependent. A user can use instead a **compiler** that translates code written in a higher-level programming language that is mostly machine independent (for example one can write a piece of code in `C, C++, Java` that is identical on a Windows PC, a Sun Workstation, a Linux PC, an Apple OS-X laptop) into a lower-evel language that can then be executed by the target platform (Sun workstation or Windows PC). The lower-level language can be assembly (and then needs to be converted by an assembler into machine code) or even plain machine code that can just be executed directly.

Thus when using a compiler a user performs two independent tasks:

(a) the user first writes a program in a higher-level programming language that is saved into a file (or more than one files), and then compiles the file(s) using a compiler, with the compiler converting these file(s) into assembly or machine code,

(b) the object code (generic for machine code or assembly that is almost ready for execution) thus generated by the compiler can then be separately executed by the CPU.

Thus in executing a program with the use of a compiler there are at least two distinct steps: compilation (step (a)) and execution (step (b)). In several instances, compilation might require more steps because for example the compiler sometimes does not translate higher-level language code into assembly code directly but to some intermediate code, that is then translated into some machine-independent assembly and so on. And if someone made a mistake in the higher-level language program, both steps need to be repeated.

For example, a C program `testprog.c` can be compiled by a command `gcc testprog.c -o testprog` that compiles the source code `testprog.c` that contains the source program into an executable file named `testprog`, and then the executable file is run by issuing a `./testprog`.

2.4. Interpreter. If the two steps of a compiler-based program execution seem complicated, an interpreter can be used that can execute directly (or indirectly) higher-level programming language statements with out the requirement for separate compilation and execution. **MATLAB** (initials for **MAT**rix **LAB**oratory), Python, Perl, Ruby, csh/tcsh/bash, Lisp, are interpreted programming languages that is, programs written in them can be executed directly by an interpreter of the language. The interpreter for **MATLAB** is slightly more complex and intelligent. It executes several precompiled pieces of code that are available to it, and compiled by a compiler that is part of **MATLAB** itself.

(c) Copyright A. Gerbessiotis. All rights reserved.

In this course we will learn how to write small programs that implement efficiently computer-based solutions of computational problems.

2.5. Algorithm. An **algorithm** is a well-defined (i.e. correct) sequence of computational steps that takes some value or set of values as input and generates some value or set of values as an output.

An algorithm is thus a method/tool for solving well-specified computational problems. A (computational) *problem* defines an input-output relationship. An **algorithm** describes a specific procedure for achieving this relationship, i.e. for each problem we may have more than one algorithms that solve it.

Before one embarks on solving a computational problem with a computer (and MATLAB) one needs to understand the problem itself i.e the input and output relationship of the problem. Computers are inherently precise and error-prone (buggy) and human beings are inherently imprecise. Combining (or coordinating) the two can have very disastrous consequences.

For example some time ago a piece of code contained the code `DO 10 I=1.10` written in programming language FORTRAN. The intent was to write `DO 10 I=1,10`. This piece of code was an orbit computation program that was used in NASA's Mercury flights in the early 60s. However when a user started running this program with known input and output data, the output the program was generating was close to but not identical to the known output data. Only because of this discrepancy was the missing comma (or incorrectly appearing period) discovered, and disaster was averted.

Consider for example the very simple problem of addition of numbers. One could describe the input output relationship for this problem as follows.

```
Input:  Two numbers a and b
Output:  a+b
```

Every NJIT student can understand what the problem is. Finding an algorithm that realizes a solution to that problem is also easy (it probably requires elementary school background). However the description of the problem is imprecise and writing code that solves it might be a complicated task even for a Computer Science student.

(c) Copyright A. Gerbessiotis. All rights reserved.

Question 2.5.1. Why is it so?

Answer. The user never specified explicitly how long the numbers can be.

- The user never specified whether a, b are natural (positive integer) numbers or
- arbitrarily sized integer numbers (say, `13213239219321` or `-23123210321020210210`),
- in decimal or binary or hexadecimal notation or some other notation,
- or fractionals (say, `1232132/1323223`),
- or real numbers (say, `232321.213213223`),
- or a mix of all of these,
- or even a complex number (i.e. $21323.23 + i2132$),
- or something else.

The same applies to the output.

Question 2.5.2. Why specifying the input form is important.

Answer. Computers are not as flexible as human beings. For example, computers are susceptible to attacks (by criminals) known as **buffer overflow attacks**. Because the input (and output) specification is imprecise, a reckless (or not very careful) programmer might make some generic assumptions on how large the input can be. A careless or intentionally devious user might supply a non-conforming input that would crash the program or force it to behave unpredictably and harmfully.

2.6.a What is a function (mathematics) and what is a function (computer science)? In mathematics a function is a relation between a set of elements (the domain) and another set of elements (the codomain) defined so that each element of the domain is associated with exactly one element of the codomain. In computer science (or programming), a function is a piece of code that performs a specific task. A function in computer science is sometimes called a subroutine, subprogram, method, procedure, routine, or just program (if it performs only a single task).

2.6.b Function names in mathematics and computer science and MATLAB. In mathematics when we write `incr`(x), we so denote the INCREMENT functions that returns $x+1$ for input x. If we decide to implement this function, we use the same name `incr` or something different that is more informative (e.g. `incrementbyone`). Of course using such a long name raises readability issues: we might write instead `increment_by_one` by using underscores to separate words. For other cases, a more drastic action is need; for example most of us will be hard-pressed to type in a keyboard $\sqrt{x}$. Instead, we usually write something like `sqrt`(x) to denote the $\sqrt{x}$ function. Thus for some very familiar mathematical functions (e.g. those appearing in a scientific calculator), MATLAB might be using different names than the ones with which we are familiar.

2.6.c How is a function defined ? In mathematics we could define `incr`(x) very simply by writing $f(x) = x + 1$. Math function names are very short and most of the times f, g, h are being used as names. When we write computer programs we need to be more imaginative: we can't name all of our functions f. In addition when we write x the implicit assumption is that x is a number. In MATLAB for example inputs to a functions can be a number, a vector (a linear collection of numbers), or a generic object called matrix (and do not forget that MATLAB stands for MATrix LABoratory). Given that computers are dummy, they do not know that f denotes a mathematical function and not say the sixth letter of the alphabet in lower-case. Thus when we write programs we need to provide lots of details about what we attempt to do (program) and what we expect as input and as an outcome (output). Is the output a number similarly to the input, or is it just a number despite the input being a vector?

(c) Copyright A. Gerbessiotis. All rights reserved.

2.6.d Parameter vs Argument. For example is $f(1/3)$ ok , i.e. is $x = 1/3$ a legitimate input? Are rational numbers allowed as an input? And if you say yes, how about complex numbers? If you don't know what a complex number is that is fine: our intend is to show you to expect the unexpected! Now that we are still talking about f, let us remind ourselves that x is the **parameter** of function f. In MATH sometimes we call x the **indeterminate**. When we evaluate f at a point and we ask 'What is the value of $f(1/3)$' we substitute $1/3$ for x. This value becomes the **argument** to the function. Thus **parameter** is something generic that is used to help us define a function, and argument is something specific that help us with the evaluation of a function. Several MATLAB functions are associated with filenames of the same name as the function itself. A simpler form is what is known an anonymous function that is not associated with a computer file (and name); an example appears on page 20. MATLAB functions might have different names than their MATH counterparts and might also behave differently. Be prepared!

When we write in math $f(x) = x^2 - 2x + 1$, we define a function on an unknown variable x (sometimes we call x the indeterminate). This specific function is called a polynomial. The degree of the polynomial is two, and sometimes we call $f(x)$ a quadratic function. Human beings are quite intelligent and can easily fill-in details. Computers are dummy unless designed by extremely intelligent humans; the presence of many bugs (i.e. irregular instructions that cause problems in a computer code/program) indicates that we still have to accept that computers are not very intelligent or human beings do not yet know how to program them intelligently.

2.6.e Human intelligence and inherent lack of precision

For a human $f(x) = x^2 - 2x + 1$ means $f(x) = x^2 - 2 \cdot x + 1$. (A $\cdot$ multiplication operator is implied between the 2 and the x.) When a human is asked `How much is` $f(0.5)$?, that human understands that the polynomial must be evaluated at $x = 0.5$, i.e. the human evaluates $f(0.5) = 0.5^2 - 2 \cdot 0.5 + 1$ and responds with the answer of `0.25`.

(c) Copyright A. Gerbessiotis. All rights reserved.

2.6.f Computer precision and lack of intelligence
A computer on the other-hand needs guidance. It needs to be told what is expected to do.

- It needs to be told that it ought to evaluate a function.
- It needs to be told what the name of the function is going to be i.e. `f`.
- It needs to be told how the input is going to be made available i.e. `through a parameter to the function`.
- It needs to be **precisely** told what the range of values for the input parameter will be.
- It needs to be told what to do with the computed value i.e. `return it to the user who invoked the function` and what the range of values for the computed function value is going to be.
- It needs to be given precisely the input function (i.e. $\cdot$) or be told otherwise that $2x$ means $2 \cdot x$.
- It needs to be allocated in advance the scratch space where extra calculations can be performed.

(c) Copyright A. Gerbessiotis. All rights reserved.

2.7. Constant. If the value of an object can never get modified, then it's called a constant. 5 is a constant, its value never changes (ie. a 5 will never have a value of 6).

2.8. Variable. In computer programs every parameter or argument whose value changes is a **variable**. Every variable has a **data-type**. What is the data-type of a variable?

2.9. Data-type. In a programming language, every variable has a data-type, which is the set of values the variable takes and the operations defined (or are allowable) on the data-type.

2.9.1 Built-in or primitive data types. Composite data types. In mathematics, an integer or a natural number is implicitly defined to be of arbitrary precision. Computers and computer languages have **built-in** (also called **primitive**) data-types for integers of finite precision. These primitive integer data-types can represent integers with 8-, 16-, 32- or (in some cases) 64-bits. An integer data-type of higher precision is only available not as a primitive data-type but as a **composite** data-type through aggregation and composition and built on top of primitive data-types.

2.10. Strongly-typed and Weakly-typed computer programming languages. In languages that are **strongly-typed** a variable needs to be fully defined (i.e. its name is presented and so is its data type specified) before it is used. In **weakly-typed** languages the first time a variable is used becomes the declaration point of the variable and implicitly serves as its definition. Its data-type is implied by the value assigned to the variable. This is convenient but sometimes confusing.

2.11 Declaration and definition. In computing we use the term **definition** of a variable to signify where space is allocated for it and its data-type explicitly defined for the first time, and **declaration** of a variable to signify our intend to use it. A declaration assumes that there is also a definition somewhere else, does not allocate space and serves as a reminder. For a variable there can be only ONE definition but MULTIPLE declarations.

(c) Copyright A. Gerbessiotis. All rights reserved.

2.12 MATLAB. MATLAB is a weakly-typed language and thus we do not need to bother with a variable's definition or declarations. A variable is defined (and also declared) the first time it is being used.

2.12.1 MATLAB primitive data types (sample). MATLAB supports a variety of integer data-types: 8-,16-,32-, and 64-bit signed (two's complement) and unsigned integers. Also single (32-bit) and double (64-bit) precision real numbers in floating-point format. These data types are called in MATLAB `int8, int16, int32, int64, uint8, uint16, uint32, uint64, single, double` respectively.

2.12.2 Characters in MATLAB. Characters are represented in MATLAB as UNICODE characters (2bytes). A string is a sequence of characters. A MATLAB string of n characters uses $2n$ bytes. The data-type **char** for character comes from the programming language C. MATLAB is implemented in the programming language called `C`. Thus it uses data types available in C. But MATLAB uses UNICODE to present characters.

2.12.3 Default data type for a MATLAB number: double The default data type for a number in MATLAB is `double`. A `double` is a real number data type in C (and MATLAB). Thus when one writes **10** in MATLAB, MATLAB assumes that it is in fact **10.0**. **NOTE THAT a `double` represents only a very small fractions of all real numbers.** Thus very often an approximation of what we have in mind is being used. This for example is the case for π, for which MATLAB uses the variable `pi` that holds an approximation of it to some decimal digits. We can normally view only four of those digits through the default disply (i.e. `format short`) but we can view 15 decimal digits if we choose `format long`. How many decimal digits of π MATLAB stores internally depends on the implementation details of the `double` data type. A `double` is a 64-bit (8-byte) representation of a real number. A `single` is an alternative that uses only 32-bits (4 bytes) for a real number.

2.12.4 logical data type. The results of relational (or logical operator) that evaluate to a true (in the form of 1) or false (in the form of 0) are stored not in a `double` but in a `logical` data type that uses only 1 byte.

2.12.5 Conversion. The results of relational (or logical operator) that evaluate to a One can convert a number to a specific data type by using the name of the data type as a function i.e. `int32(10)` converts 10 into an 32-bit signed (two's complement) integer. Note that an `int8` data type can only represent in two's complement the integers from -128 to 127 (inclusive). Thus `int8(128)` and `int8(-129)` will return 127 and -128 respectively, the largest and smallest number that can be represented in 8-bit two's complement.

2.12.6 Bonus For the `int, uint` data types use `format hex` to view them in hexadecimal format.

(c) Copyright A. Gerbessiotis. All rights reserved.

2.13 MATLAB variables. All variables introduced by a user can be retrieved by a `who` or `whos` function in MATLAB that shows all the variable introduced in the user's **WORKSPACE**. A user can name a variable in a very specific MATLAB-dictated way. The of a MATLAB variable is a sequence of characters (i.e. string) that is unique to 63 characters. That is, a string can be longer than 63 characters but only the first 63 count and identify uniquely the variable. What characters can appear in a variable's name. The first character must be an `alphabetic` one, i.e. one of the 52 English lower-case or upper-case characters `A-Za-z`. The remaining one can be `alphabetic`, `numeric` (i.e. the ten digits `0-9`) or the underscore _. No other characters are allowed; thus among other things space and punctuation marks can not be part of a variable's name. Note that variables are **case sensitive**. Remember that MATLAB is a weakly-typed language and thus variables need to be defined/declared before their use. The first use of a variable serves as its definition. Variable `namelengthmax` gives the number of significant characters in the implementation of MATLAB (currently 63). In MATLAB we can view the internal representation of numbers or strings by using the function `whos` that shows the variables defined their internal data-type plus size information.

Example 2.13.a. Thus `alex1` and `alex_1` are ok, but `alex 1` and `1alex` are not valid MATLAB variables.

Example 2.13.b. The variables `alex`, `Alex` and `ALEX` are all different.

(c) Copyright A. Gerbessiotis. All rights reserved.

2.13.1 MATLAB special ariables pi,i,j. MATLAB variable `pi` has a fixed value that of π i.e. $3.14\ldots$. MATLAB variables i, j are in fact complex numbers and are defined to be the square roots of -1, i.e. $i = j = \sqrt{-1}$. If you want to convince yourselves about that type in MATLAB $i * i$ or $j * j$. It is imperative that you not use these names as variable names.

2.13.2 MATLAB caution: playing with fire. MATLAB is a weakly-typed language and thus it allows use to mess with these variables. So One can write $pi = 4$. What one does with this statement is to create its own local variable to the user's WORKSPACE that overrides the default value for variable *pi*. This is one good reason to avoid using such names. Other variable you should not be redefining are `ans, true, false`. All of these are also called `reserved words.` More **keywords** that should also not be used as variables are. For now we use the terms reserved words and keywords interchangeably. The former might be used without MATLAB generating an error or a warning.

```
break case catch continue end else elseif for function global if otherwise persistent
return switch try while
```

2.14 Elementary Operators and Operations. Binary and Unary. An operator such as $+, -, *, /, \backslash$ is a **binary operator** that accepts two arguments (sometimes called operands), and combines them according to the rules implied by the indicated operator by performing the corresponding operation. Thus operator + (i.e. plus) adds its two arguments by realizing the operation known as addition. The operator − indicates subtraction and the operator \ backdivision, that is $x\backslash y$ is y/x. Operator ^ used as $a\hat{}b$ raises a to the b-th power. The unary operator + indicates or assigns a positive sign to a scalar(number) and the − a negative sign, i.e. turns a number into negative. So these operators are context specific and can behave as binary and unary operators.

2.14.1 Commutative and Associative operators. Operators $*$ and $+$ are commutative, that is, $a+b = b+a$ and $a*b = b*a$, for all a, b. Operations such as addition and multiplication are called associative in the sense that the corresponding operators satisfy $a * (b * c) = (a * b) * c$ and $a + (b + c) = (a + b) + c$ for all a, b, c. Subtraction and division are not associative. For example, $(1 - 2) - 3 = -4 \neq 1 - (2 - 3) = 2$.

(c) Copyright A. Gerbessiotis. All rights reserved.

2.15.1 When is a zero a zero? Note also that when we apply a variety of functions the result is not the real number we have in mind but an approximation of it as well. For example sin(pi) is the mathematical function of applying the `sine` function to MATLAB's representation (or approximation of) π. Though normally $\sin(\pi)$ is 0, the result given by MATLAB is not zero although it might be very close to it. In my case sin(pi) returns $1.2246e-16$ which is how MATLAB writes 1.2246×10^{-16}. It is a very small, close-to-zero number, but not zero. So beware of real numbers in MATLAB, and how they compare (or not) to zero.

2.15.2 Exponential notation. Note that MATLAB has problems raising 10 to powers that need to appear as a superscript to 10. Thus writing 10^{-16} in MATLAB poses a variety of output problems. An easy way is to write it as $1.e-16$ instead, or $1.0e-16$, or $1.E-16$, or $1.0E-16$, where a lower-case `e` or an upper-case `E` might be used.

2.15.3 `true` and `false`. The result of a logical (sometimes called boolean operator) is a `true` or `false`. MATLAB stores the result in a `logical` data type that only uses one byte. Thus `1` the number may be stored in a `double` or 8-bytes but `1` for `true` only uses one-byte. Thus `1==1` will give `ans=1` and this would use one byte (do a `whos`).

(c) Copyright A. Gerbessiotis. All rights reserved.

The next few pages show examples of using certain features of MATLAB. Some of the reserved words already introduced or to be introduced are included in those examples. The list below shows what page's examples use what keywords.

```
Page 15: help, clc, clear, who, whos, pi, ans,  i, j,

Page 16: int8, ..., uint8, ..., single, double (def) , char , format hex, format long, format short (def)

Page 17: length

Page 18: ls , dir, pwd, type, cd

Page 19: bin2dec, dec2bin, hex2dec, base2dec, intmax, intmin, realmax, realmin
         cos,sin,log,exp,tan, acos,asin,atan, eps, sqrt

Page 20: Operators +,-,*,/,\, @, ^  and an anonymous function


Page 23: length,size, '
Page 24:  prod,sum, triu, tril, ones, zeros, eye, rand, diag, end
Page 28:  + , - , .*
```

(c) Copyright A. Gerbessiotis. All rights reserved.

```
>>  % the percent sign denotes a comment; everything past the percent on that line would be ignored
>> help clc
>> clc           % clear screen
>> clear         % clear MATLAB workspace for the user
>> who           % find variables defined in user's workspace
>> whos          % same as who but with size information (s of whos denotes size)
>> pi            % pi is a MATLAB constant for 3.1416....
ans = 3.1416     % ans is a variable just defined in the workspace
                 % The previous statement asks MATLAB to evaluate expression    pi
                 % Its value is the value of constant pi i.e. 3.1416
                 % That value is returned to a MATLAB local variable named ans for answer
>> clear         % clear workspace
>> whos
>> pi;           % suppress output but ans is still created and value is still available
>> whos          % ans was define with previous statement
>> ans           % and its value can be retrieved
ans = 3.1416
>> i             % i,j are MATLAB constants
>> j
>> i*i           % i and j are the square roots of -1 i.e
 ans = -1        % i * i = j *j = -1
>> j*j           % they are complex numbers
 ans = -1
>> whos          % i,j,pi are constant
>> pi     % vs
>> pi ;   %
           %semicolon ; suppresses output
>> who    %
>> clear
>> pi =10 % OUCH !!!
>> whos   % MATLAB creates a local variable named pi; the constant pi is not available any more
          % local pi hides constant pi. To unhide the latter delete (aka clear) the former
>> clear
>> pi     % Our 3.1416 is back...
>> 5 + 6
ans =  11
>>  5 +
      |
    Error: Expression or statement is incomplete or incorrect
>>  5 + ...  % ... allows you to continue to next line
    6
ans = 11
```

(c) Copyright A. Gerbessiotis. All rights reserved.

```
%  Characters vs Strings: A String is a sequence of 0 or more characters
%  Sequence vs Set  : Order matters (sequence) vs Order does not matter (set)
%  A Set is a collection of elements  : order DOES NOT MATTER
%  Set {1,2,3} is equal to {2,3,1}
%
%  But a sequence is an ordered set of elements: order MATTERS
%  Sequence <1,2,3> is different from <3,1,2}
%  the 1st element of <1,2,3>   is 1
%  but 1st element of <3,1,2>   is 3
%int8, int16, int32, int64     : MATLAB primitive datatypes for 8-,16-,32-, 64-bit integers
%                                (two's complement representation
%                              : int8 represents  -128 ... 0 ... +127
%
%uint8, uint16, uint32, uint64 : MATLAB primitive datatypes for 8-,16-,32-, 64-bit unsigned
%                                 integers
%                              : uint8 represents  0 ... +255
%
% single   : 4B representation of a real number in MATLAB
% double   : 8B representation of a real number in MATLAB  (DEFAULT)
%
% char     : 2B Unicode representation of a character in MATLAB
%
>> a= 1   ;  % 1 is assigned to variable a; the default data type of numerical variable a
             % is a double i.e. the value of a is 1.0 using 8 bytes
>> whos      % here is a confirmation
  a 1x1 8 double
>> format hex
>> a         % a in hexadecimal is 3ff0000000000000
             % a in binary is  00111111 11110000 00000000 00000000 00000000 00000000 00000000 00000000
             % (space between the bytes for readability)
>> format short
>> pi
>> format long
>> pi
>> format
>> pi
>>
>>  a=int8(1)   % a is an 8-bit integer (two's complement) holding integer value 1
                % it is 0x01 in hexadecimal or 00000001 in binary
```

```
>>  a=int32(1)  % a is 32-bit integer
>>  a=uint64(1);   % a is now a 64-bit unsigned integer holding integer value 1
>>  format short   % or just type format
>>  a= '10';       % a is a string of two characters one is character 1 and the other is character 0.
                   % it is 4B long since a character in MATLAB is 2B in UNICODE
                   % The UNICODE value for 0-9 a-z A-z is the same with the ASCII value
>>  a+ 0           % is a trick to retrieve the  UNICODE values of character 1 and 0
    ans  = 49 48   %  Character 1 is 49  and Character 0 is 48 i.e. 0x31 and 0x30 respectively
>>  format hex
>>  a+0            % 49.0 and 48.0 in hexadecimal (real numbers not integers)!
ans = 4048800000000000   4048000000000000
>>  whos  a        % provides information about a : it is a 2-character string/array 1x2
 a  1x2   4 char   % of 4B long

>> a=10;         % semicolon suppresses output
>> a=10          % absence of semicolon forces output;
  a= 10          % confirms 10 is assigned to variable a
>> int8(100)
>> int8(127)
>> int8(128)    % Observe the limits of 8-bit integers
>> intmax('int8');
>> int8(-128)
>> int8(-129)
% Value is the same int8(1) int16(1) uint32(1) single(1) double(1)
% but stored representation is different!
>> format short
>> a='123';
>> length(a) % 3 characters in a; length of string
 ans = 3
>> a=10;
>> length(a)
 ans = 1;
>> a='';    % Empty string of length 0
>> length(a)
>> a= ''''  % String containing ' (1 character)
>> length(a)
```

(c) Copyright A. Gerbessiotis. All rights reserved.

Familiarization with the screen output

```
>> pi
>> pi ;   % semicolon ; suppresses output
>> who
>> whos   % whos is who but includes size information !
>> format short  %default prints only  4 decimal digits for a real
>> pi
>> format long   % prints 15  decimal digits
>> pi
>> format hex    % hexadecimal
>> pi            % pi (a double ) in hexadecimal notation
>> format short  ;
>> pi, single(pi), double(pi)   % multiple commands separated by ,
>> format long   ;
>> pi, single(pi), double(pi)   % Observe differences!
>> pi
   ans = 3.1416
>> ans
  ans= 3.1416
% ans for answer, stores all evaluated expressions!
>> pi   % evaluates pi to 3.141...etc and displays it in the standard format (default short)
>> whos
>> pi = 10;
>> whos
% You redefined pi to be a local variable ... It's not out pi (3.1415...)
>> help who   % offers assistance to commands
>> ls
>> dir
>> pwd
>> type filename  % print contents of file with  name filename (must exist)
>> cd direct   % go into directory direct (must exist or error)
```

(c) Copyright A. Gerbessiotis. All rights reserved.

```
 >> clc           % clear screen
 >> clear         % clear MATLAB workspace
 >> whos          % no variables listed whos provides size information, who does not
 >> bin2dec('111');  % convert binary number given as string into decimal
    ans = 7         % variable ans (for answer) holds values when MATLAB is being used as
                    % a calculator
 >> whos            %  ans was defined in the workspace. Default numeric data type is double: 8B
    ans  1x1  8  double

 >> dec2bin(7)      % decimal to bin results is a string i.e. 111 (look how it gets printed)
ans=
111
 >> whos
    ans  1x3  6  char
>> hex2dec('7')    % hexadecimal to decimal conversion
  ans =
      7

>> base2dec('100',8)  % arbitrary base (octal in this example) to decimal conversion
ans =
      64
>> % ISSUES WITH REALS NUMBERS
>> intmax , intmin     % separate multiple statements with comma
>> intmax ; intmin     % separate multiple statements with semicolon   , Compare behavior
>> realmax, realmin
>> cos(pi/2)     % it is supposed to be zero but it is not!
ans = 6.123e-17  % MATLAB
>>  1 == sqrt(1e-17+1)  % is 1 and sqrt(1.00000000000000001) the same ?
>>  1 == sqrt(1e-16+1)  % is 1 and sqrt(1.0000000000000001)  the same ?
>>  1 == sqrt(1e-15+1)  % is 1 and sqrt(1.000000000000001)   the same ?
>>  eps  % for epsilon aka accuracy
  ans = 2.20 ....  e-16
>> eps(0), eps(double(0)), eps(single(0))
>>  (5 + eps(5)) == 5
>>  (5 + eps(sqrt(5))) == 5
```

```
operator :  *  is a symbol denoting an operator for operation multiplication
operation:  multiplication is the operation denoted by *
            multiplication is a binary operation
operator :  *  is a binary operator ; it requires two operands

    a * b
 a: left operand
 *: binary operator for multiplication
 b: right operand

    +: binary operator for addition       Example: 5+6  gives 11
    -: binary operator for subtraction    Example: 5-6  gives -1
    *: binary operator for multiplication Example: 5*6  gives 30
    /: binary operator for division       Example: 5*4  gives 1.2500 (format short, default one)
    new operator
    \: binary operator for back division  Example: 5*4  gives 0.8000  (i.e. 4/5)
    ^: binary operator for exponentiation Example: 2^4  gives 16

    but
    +: unary  operator for positive sign  Example: +5
    -: unary  operator for negative sign  Example: -5
                                          Example: -(-(-5))
                              But also in  MATLAB : ---5
  Functions abs, floor, ceil, round,  sin, cos, exp, log, tan, asin, acos, atan etc

>> f = @(x) (x*x +5.0)   % Define an 'anonymous' function  f(x)= x^2 + 5
>> f(5)                  % f is the function and x is the parameter (aka variable, indeterminate)
  ans = 30               % 5 is the argument that substitutes for x in f(x)
>> z=5                   % z is a variable whose value is 5
>> f(z)                  % z is the argument you first evaluate z to 5 and this value then
                         % substitutes for x in f(x)
  ans = 30
```

(c) Copyright A. Gerbessiotis. All rights reserved.

2.16. What is a data model? A data model is an abstraction that describes how data are represented and used.

2.16.1 MATLAB data model. The data model of MATLAB is the concept of a **matrix** i.e. elements arranges into a give number of rows and a given number of columns. The number of rows and columns are the dimensions of the matrix. A matrix with n rows and m columns is denoted by its geometry $n \times m$. Matrix elements are of the same data type. Thus the MATLAB data model supports characters and strings (collections of characters), integers of different sizes (eg. 32-bit or 64-bit etc), real numbers in the form of floating-point numbers of different sizes, complex numbers, and includes not only these (primitive) data-types but also methods that allow aggregation and composition of these data-types (e.g. in the form of matrices and vectors).

2.16.2 MATLAB internal representation. MATLAB implements the abstraction of a matrix by using a data structure called **array**. Array elements are random-access indexed. Thus for array a its elements $a(1)$ and $a(1000000)$ can be accessed in the same amount of time no matter what the indexes are (first and millionth). An array stores a collection of elements of the same data type. Every time we type `whos` in MATLAB we are reminded of that: something like a `double array`, or `char array` or `logical array` is written to the far right of a variable's name!

2.17. What is an Abstract Data Type (ADT)? An abstract data-type (ADT) is a mathematical model and a collection of operations defined on this model.
For example, we might define an ADT called **dictionary** D for which we define two operations: (a) Insert(D,x), for inserting x into the dictionary, and (b) LookUp(D,x), for looking-up object x into dictionary D. Note that at this abstraction level it is not necessary to define whether x is a string of characters (i.e. a word) or some other data-type. The important part of the definition is the collection of the operations and their semantics.

(c) Copyright A. Gerbessiotis. All rights reserved.

2.18. MATLAB Data Model. MATLAB's data model is very simple. Every object in MATLAB is a **matrix** or a special case of a matrix that is called a **vector**. A matrix or a vector are implemented internally by using a computer-based structure known as the **array**. Therefore before we begin talking about MATLAB's capabilities we need to provide definitions for a **matrix** and a **vector**. In the next few pages we review some definitions from matrix theory.

2.19. Vector and Matrix. You will be formally introduced to the concepts of a matrix and a vector in a linear algebra class. Informally, a **matrix** is a rectangular array of numbers. For example, matrix A below is how we write a 3×4 matrix, where 3 indicates the row dimension of the matrix, i.e. the number of its rows, and 4 the column dimension i.e. the number of its columns. We thus say that A has three rows and four columns and we call it a 3×4 matrix A. The number of elements in A is thus the product of its two dimensions, in this case 12. We can individually reference elements of the matrix.

2.20 Indexing. For example we can refer to the element of the second row and third column, i.e. 4. For matrix A we sometimes say $A = (a_{ij})$ to indicate how we reference its elements individually (i.e. using a in lower case). If such a definition is not explicitly given, you may assume it is present or just reference A by using the capital case letter instead. Thus we refer to an element of row i and column j of A by writing A_{ij}, or a_{ij}. Sometimes one might write $A_{i,j}, a_{i,j}$ for clarity i.e. insert a comma between the indices (indexes). If a subscript becomes messy or unreadable one can write $A(i,j)$ or $a(i,j)$. Thus $A(2,3), a(2,3), a_{23}, A_{23}$ all refer to an element of the second row and third column of A that is the element whose value is 4. Note that row and column indices start from 1; in some programming languages that implement matrices indexing starts from 0. MATLAB adheres to the start from 1 indexing.

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{pmatrix}$$

2.21. Notation. A generic (i.e. general) matrix say B may be denoted as an $n \times m$ matrix, i.e. it has n rows and m columns.

2.21.1 Square matrix. A special case of a matrix is a **square matrix** in which the number of rows and columns is the same. Thus if $n = m$ we say that then B is a square $n \times n$ (although one could also write that B is an $m \times m$) matrix. Several computational problems can be described through square matrices. Matrix B on the following page is a 4×4 matrix.

(c) Copyright A. Gerbessiotis. All rights reserved.

```
 MATRIX and MATLAB
  % Data model : Abstraction that describes how data are represented and used
  MATRIX:                                 ARRAY: internal representation of a
                                                 matrix
  [is the abstraction]                    [is storage inside ]
  MATLAB abstraction that allows          ARRAY is being used to store a matrix
  users to manipulate data

 >> a=10  % A variable is a 1x1 matrix stored internally as an array of doubles (one double
a = 10    % in our case
>> b= 'alex';
>> whos
  Name      Size                    Bytes  Class
  a         1x1                         8  double array <<<<<< array internal representation
  b         1x4                         8  char   array
Grand total is 5 elements using 16bytes
% Array is a  sequence of elements of the same data type
% The data type of a is double : 1 elements  with value 10
% The data type of b is char   : 4 elements  with values 'a' 'l' 'e' 'x' respectively

% data type : set of values a variable takes
% Data model : Abstraction that describes how data are represented and used
% Types of data type :  primitive (aka built-in) and composite
% Primitive Data types in MATLAB:  logical, char, single, double, int8, ..., uint8, ...
% Composite Data types in MATLAB: logical array, double_array, function_handle , etc
>> a = [ 1 2 3 4 ; 5 6 7 8; 9 10 11 12]   % A 3x4 matrix a

>> length(a) % the largest of the dimensions
>> size(a)   % the dimensions
>> size(a,1) % number of rows
>> size(a,2) % number of columns
% MATRIX
%
%               m x n : generic
% m=n           n x n : square  #rows== # columns
% m=1           1 x n : row vector
% n=1           m x 1 : vector (also column vector)
% m=n=1         1 x 1 : scalar (i.e. number, value)
%
% a' is the transpose of a. If a is m x n, then a' is n x m
```

(c) Copyright A. Gerbessiotis. All rights reserved.

```
>> a = [1 2  3 4; 5 6 7 8; 9 10 11 12]
 >> a'  % transpose of a, rows become columns and columns rows
ans =
     1     2     3     4
     5     6     7     8
     9    10    11    12
>> a(2,3)  % retrieve element of 2nd row and 3 column i.e. 7
 ans = 7
>> a(2,3)=70  %set element to 70
 a  =
     1     2     3     4
     5     6    70     8
     9    10    11    12
>> a(end, end)  % end is a variable; if used as row index is number of rows
                %                               col index is number of colums
>> a(10,10) = 100 % MATLAB is a WEAKLY TYPED LANGUAGE. THIS WORKS
                  % THIS WOULDNOT HAVE WORKED IN A STRONGLY TYPED LANGUAGE

>> prod(a);
>> prod(a');
>> sum(a);
>> sum(a');
>> prod(size(a));
>> diag(a);    % vs diag(a,1) or diag(a,-1)
>> diag(diag(a))
% triu(a) : retrieves the elements at diagonal and above; everything below set to 0
% tril(a) :  retrieves the elements at diagonal and below; everything above set to 0
% ones(n)  nxn matrix of ones  [default: double]
% ones(m,n) mxn matrix of ones  [default: double]
% ones(m,n,'int8') mxn matrix of ones  in int8 data type. Replace last argument appropriately
% with other numeric data types (single, double, uint8, uint16, uint32, uin64, etc
%
% zeros , eye are similar to ones in syntax
% zeros(m,n) mxn matrix of zeros  [default: double]
% eye  (m,n) mxn matrix of zeros  except with ones in the diagonal  [default: double]
% rand(m,n)  mxn matrix of random values in [0,1]
```

(c) Copyright A. Gerbessiotis. All rights reserved.

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \quad C = \begin{pmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{pmatrix} \quad D = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 7 & 8 \\ 0 & 0 & 1 & 12 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad E = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 5 & 6 & 0 & 0 \\ 9 & 10 & 0 & 0 \\ 13 & 14 & 15 & 16 \end{pmatrix}$$

2.21.2 Row-major and column-major fill-in of a matrix. Matrices B and C are both square of row/column dimension 4. Numbers $1 - 16$ are used to fill both matrices. You may have observed that the fill-in pattern is **row-major** for matrix B in the sense that the numbers fill-in the first row first, then the second and so one left-to-right top-to-bottom. On the other hand the fill-in pattern is **column-major** for matrix C since the elements first fill the first column, then the second and so on top-to-bottom left-to-right.

2.21.3 (Main) Diagonal of square matrix. The diagonal of a square matrix is the elements whose row and column indices are the same, i.e. all elements a_{ii}. We can describe B's diagonal by writing $1, 6, 11, 16$ top-left to bottom-right. The elements above the main diagonal form the upper triangle and the elements below form the lower triangle.

2.21.4 Upper and Lower triangular matrices. An upper triangular matrix is a matrix whose lower triangle is all zeroes i.e. this is roughly equivalent to saying its upper triangle and the diagonal are NON zeroes. A lower triangular matrix is a matrix whose upper triangle is all zeroes i.e. roughly speaking, the lower triangle and the diagonal are NON zeroes. Matrix D is an upper triangular and E is a lower triangular matrix. If the diagonal elements are zero then a matrix can be further called **strictly upper** or **strictly lower** triangular.

2.21.5 Diagonal Matrix. A diagonal matrix is a matrix whose non-diagonal elements are all zero.

2.21.6 Zero matrix. The zero matrix **O**, is a matrix whose elements are all 0. Thus a zero matrix can be viewed as an extreme example of a diagonal matrix with all elements including the diagonals being zero.

2.21.7 All-one matrix. The all-one matrix **1**, is a matrix whose elements are all 1.

2.21.8 Unit matrix. A unit matrix **I** is different from an all-one matrix. All its elements are zero except for the main diagonal elements that are all 1. That is a unit matrix is a diagonal matrix whose main diagonal elements are all 1.

(c) Copyright A. Gerbessiotis. All rights reserved.

2.21.9 Matrix transpose. The transpose of matrix A is denoted by A^T or A^t or sometimes simply A'. A^t is A with its rows becoming columns and its columns becoming rows of A^t. Thus if A is an $n \times m$ matrix, then A^t becomes a $m \times n$ matrix.

$$A^t = A^T = A' = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 9 \\ 1 & 8 & 27 \end{pmatrix} \qquad x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \qquad y = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \qquad x^t = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$$

2.21.10 Symmetric Matrix. A square matrix A is symmetric if and only if $A = A^t$. (The all-one, the zero, and the unit matrix are all symmetric.)

2.21.11 (Column) vector x. A (column) vector x is an $n \times 1$ matrix, i.e. a matrix with just one column. When we talk about vector x we implicitly mean column vector x.

2.21.12 Row vector y. A row vector y is an $1 \times n$ matrix, i.e. a matrix with just one row.

2.21.13 Writing a vector. On a piece of paper, it's easier to write a (column) vector as a row vector horizontally rather than using the vertical layout. This way we we write down x^t and read `vector` x, or we can write down y and read `vector` y^t.

2.21.14 Scalar. A scalar (i.e. value) is a 1×1 matrix, or (column) vector, or vector, or row vector. Given the multiple possibilities of calling such an entity (as a matrix or vector), we just call it a scalar.

2.21.15 Other matrices: Tridiagonal. A tridiagonal matrix is a square matrix $A = (a_{ij})$, where $a_{ij} = 0$ for all i, j such that $\mid i - j \mid > 1$.

2.21.16 Other matrices: Permutation. A permutation matrix is a square matrix whose entries are zero except for a single one in every row and column.

(c) Copyright A. Gerbessiotis. All rights reserved.

2.21.17. Matrix Addition. For two matrices A and B of geometry $n \times m$, matrix C of geometry also $n \times m$ is called the **sum** of A and B and we write $C = A + B$ if and only if $c_{ij} = a_{ij} + b_{ij}$.

2.21.18. Matrix Subtraction. For two matrices A and B of geometry $n \times m$, matrix C of geometry also $n \times m$ is called the **difference** of A and B and we write $C = A - B$ if and only if $c_{ij} = a_{ij} - b_{ij}$.

Addition and Subtraction requires $n \times m = n \cdot m$ operations. For multiplication, Linear Algebra and MATLAB support the traditional matrix multiplication, to be given in 2.21.20. In addition MATLAB supports an array multiplication which is defined similar ro matrix addition and matrix subtraction.

2.21.19. Array multiplication. For two matrices A and B of geometry $n \times m$, matrix C of geometry also $n \times m$ is called the **array product** of A and B and we write $C = A.*B$ if and only if $c_{ij} = a_{ij}.*b_{ij}$. (Note that the operator $.*$ is a period immediately followed by an asterisk.) Array division or back-division can also be defined.

2.21.20. Matrix multiplication. For two matrices A and B of geometry $n \times m$, and $m \times p$ respectively, matrix C of geometry $n \times p$ is called the **matrix product** of A and B and we write $C = A \times B$ if and only if $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$. Similarly to a scalar product definition, the $\times$ may be dropped or even replaced by a $\cdot$ dot; thus $C = AB$ or $C = A \cdot B$ or $C = A \times B$ are all matrix multiplication products. However the $.*$ needs to be explicitly specified for an array product.

Matrix multiplication thus requires mnp scalar multiplications and $np(m-1)$ scalar additions. Thus for matrix multiplication to be defined the right dimension of A must be equal to the left dimension of B. Or we can say the innermost dimensions of the product as they are implied by $A \times B$ are the same.

2.21.21. Matrix inverse. For a **square** matrix A of geometry $n \times n$, under certain conditions, the inverse of matrix A is defined and denoted by A^{-1} rather than $1/A$. The inverse A^{-1} of A is such that $A \times A^{-1} = A^{-1} \times A = I$.

2.21.22. Matrix division. For **square** matrices A, B of geometry $n \times n$, A/B is to mean $A * B^{-1}$ and $A\backslash B$ to mean B/A.

(c) Copyright A. Gerbessiotis. All rights reserved.

Example Operations

```
a =  1     2     3     4
     5     6     7     8
     9    10    11    12
b =  1     1     1     1
     1     1     1     1
     1     1     1     1

a+b                          % Array operation + arguments are matrices of same geometry
ans =2     3     4     5
     6     7     8     9
    10    11    12    13

a-b                          % Array operation - similarly for arguments!
ans =0     1     2     3
     4     5     6     7
     8     9    10    11

a .* b                          % Array operation mult is denoted .*  not *
                                % Array division/backdivision are ./ and .\
ans =1     2     3     4
     5     6     7     8
     9    10    11    12
```

(c) Copyright A. Gerbessiotis. All rights reserved.

Example Operations

```
x = [ 1 2 3] ;                      % a row vector
y= [ 10 20 30 ]';                   % a vector y  the transpose of row vector [10 20 30];
x = 1     2     3
y = 10
    20
    30
x * y                     % inned/dot/vector product !!
                          % x is 1x3 y is 3x1 result is 1x1
ans = 140
y * x                     % y is 3x1 x is 1x3 result is 3x3
ans=10    20    30
    20    40    60
    30    60    90

% Note that for both operations the result has geometry related to the outer dimensions
%1x3 and 3x1 gives 1x1
%3x1 and 1x3 give  3x3
%but inner dimenstion are the same ..x3 3x.. or  ..x1 1x...
%
% This happens in MATRIX MULTIPLICATION IN GENERAL
%   A    = B   x   C
% (m,n)   (m,p)   (p,n)
%  ^ ^     ^ ^_____^ ^
%  | |     |         |
%  | |_____|_________|
%  |       |
%  ________^
%
%
%The (i,j)-the element of A is
%   A(i,j) = sum from k=1 to p    B(i,k) * C(k,j)
%
%i takes m values
%j takes n values  for an mxn A
%
%k takes p values
%
%
%A(i,j)   requires   p       Mults and  p-1     Adds
%All of A requires   pmn     Mults and (p-1)mn Adds
%
```

(c) Copyright A. Gerbessiotis. All rights reserved.