SEEE1022 INTRODUCTION TO SCIENTIFIC PROGRAMMING

CH2
Variables

Dr. Mohd Saiful Azimi Mahmud (azimi@utm.my)
P19a-04-03-30, School of Electrical Engineering, UTM

OBJECTIVES

1. To understand what is syntax, variables and operation of programming language.
2. To be able to create array of different sizes.
3. To be able to access array in order to get and set the array’s elements.
4. To understand the different data type of variables and how to create and use them.
5. To understand the constant variables available in MATLAB.

MATLAB

$$\begin{array}{c|cccc|} & 1 & 2 & \ldots & n \\ \hline 1 & a_{11} & a_{12} & \ldots & a_{1n} \\ 2 & a_{21} & a_{22} & \ldots & a_{2n} \\ 3 & a_{31} & a_{32} & \ldots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m & a_{m1} & a_{m2} & \ldots & a_{mn} \end{array}$$

INTRODUCTION

PROGRAMMING LANGUAGE BASIC

SYNTAX

- Particular layout of words and symbols describing **variables** and **operation**.

VARIABLES

- Stored value, which can be retrieved by referring to an associated name.

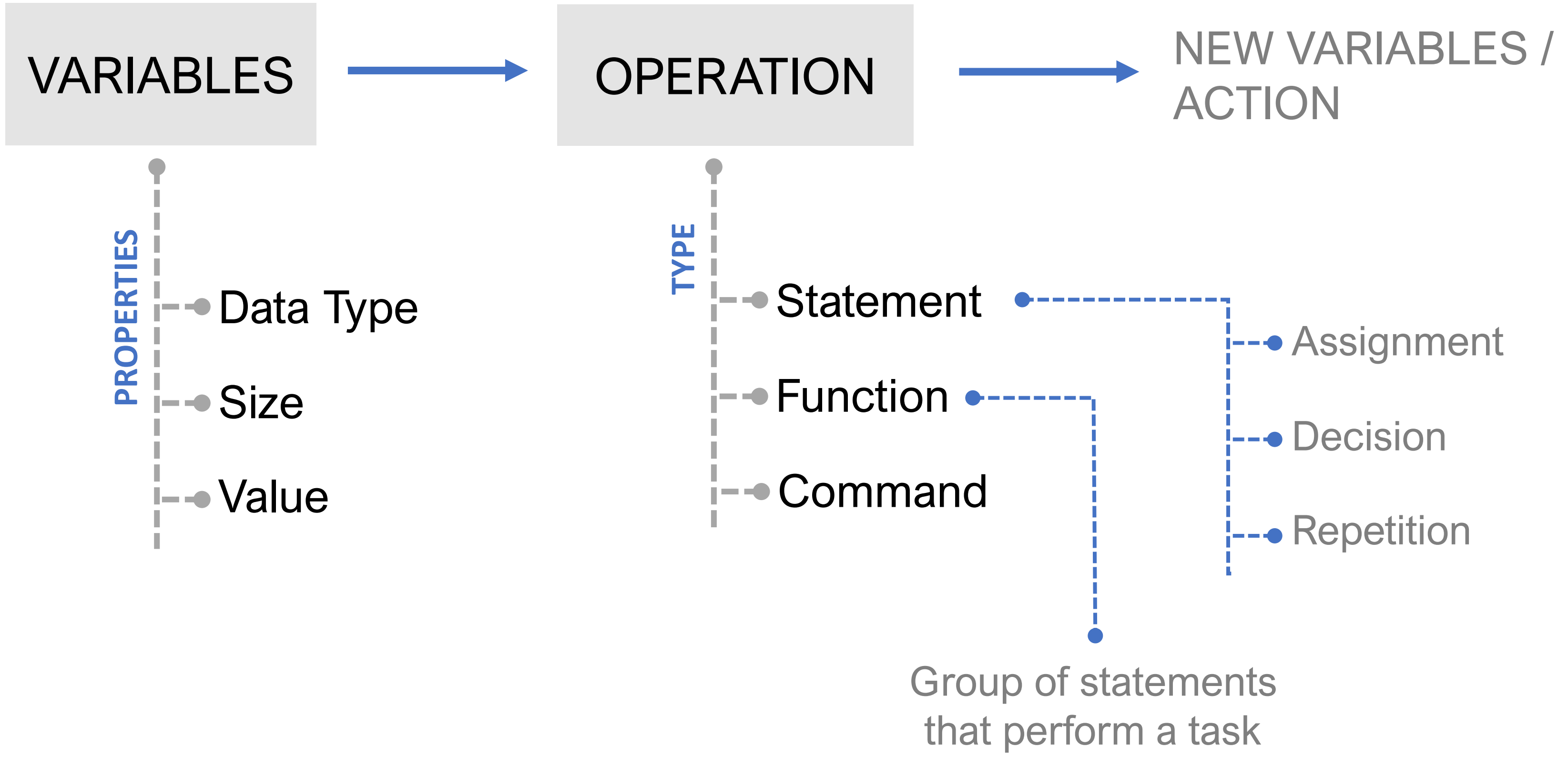
OPERATION

- Action taken to the variables to either create a new variable or perform a new operation.

LEARNING THE LANGUAGE

Syntax	GRAMMAR
Variables	NOUN
Operation	VERB
MATLAB Documentation	DICTIONARY

PROGRAMMING LANGUAGE BASIC



VARIABLES → OPERATION → NEW VARIABLES / ACTION

PROPERTIES
- Data Type
- Size
- Value

TYPE
- Statement
  - Assignment
  - Decision
  - Repetition
- Function
  - Group of statements that perform a task
- Command

VARIABLES

INTRODUCTION

- A variable is created simply by assigning a value to it at the command line or in a program. For example:

```
>> a = 5
```

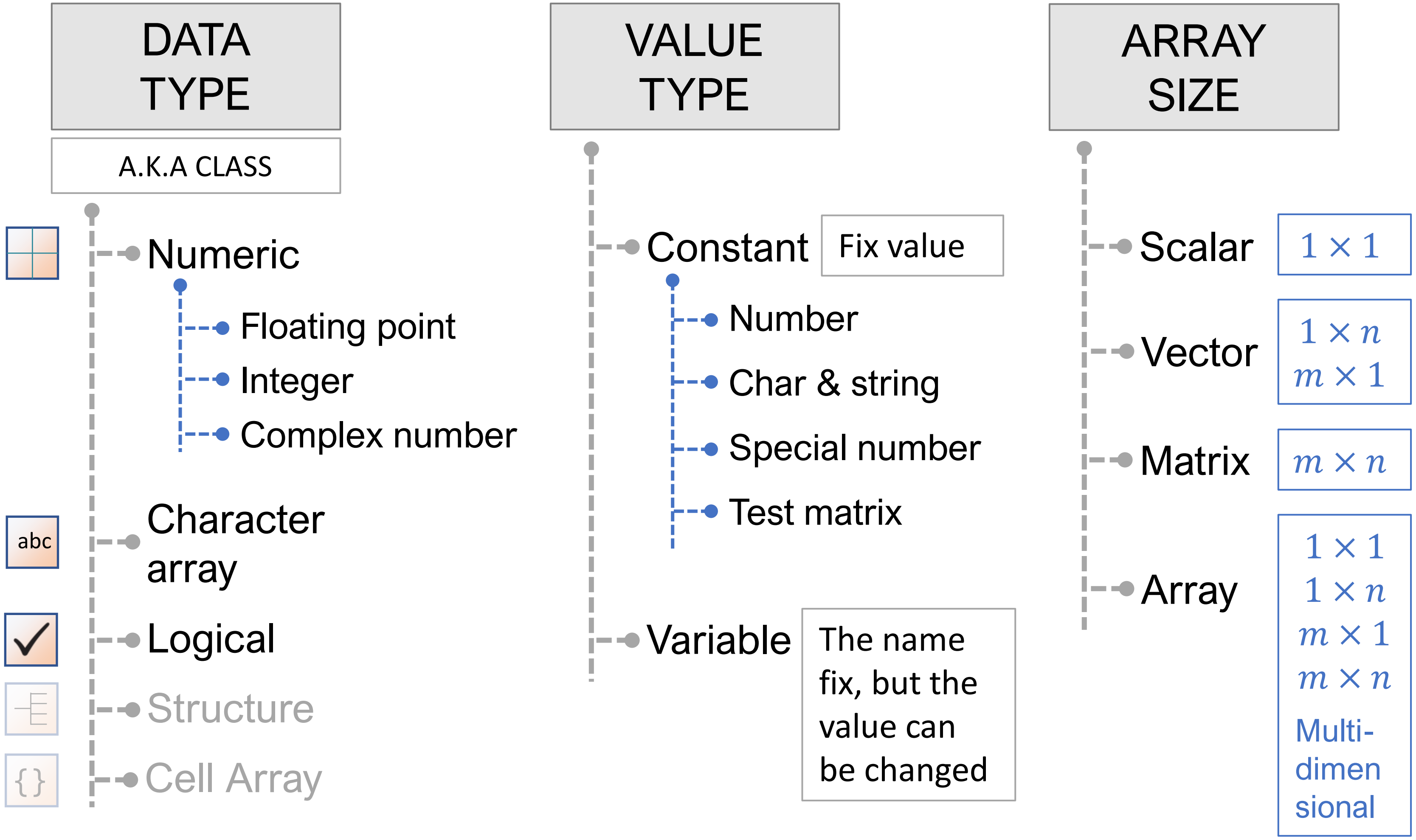
- This is read as: variable ‘a’ is assigned a value of 5
- We are telling the machine to store the value on the right hand side of the equation in a memory location, and to name that location as ‘a’

- Attempt to a non-existent variable, you will get an error message.

```
>> a = 1+b

Undefined function or variable ‘b’
```

- In MATLAB, all variables are arrays. To understand this, lets get to next slide on the properties of the variables.

VARIABLES PROPERTIES



DATA TYPE

A.K.A CLASS

- Numeric
  - Floating point
  - Integer
  - Complex number
- abc Character array
- ✓ Logical
- Structure
- { } Cell Array

VALUE TYPE

- Constant — Fix value
  - Number
  - Char & string
  - Special number
  - Test matrix
- Variable — The name fix, but the value can be changed

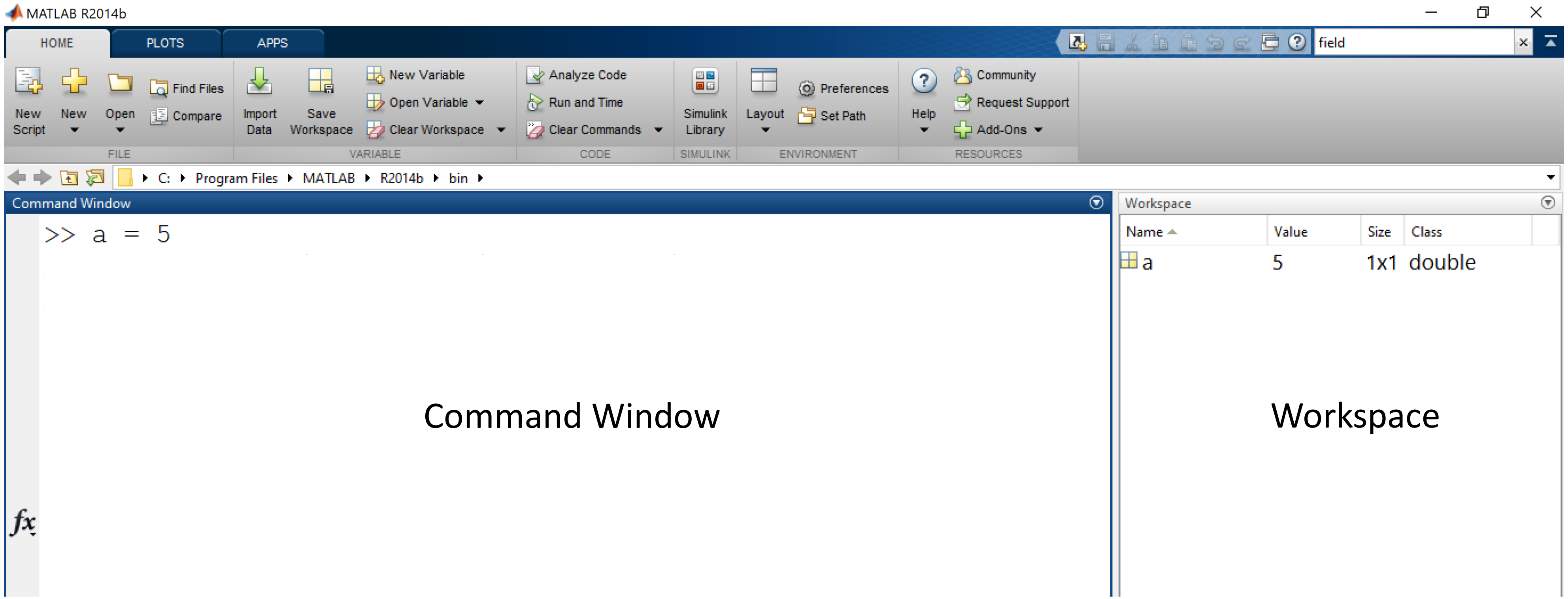
ARRAY SIZE

- Scalar — 1×1
- Vector — $1 \times n$, $m \times 1$
- Matrix — $m \times n$
- Array — 1×1, $1 \times n$, $m \times 1$, $m \times n$, Multi-dimensional

WORKSPACE

DESCRIPTION

- Variable properties can be found in the workspace window of the MATLAB desktop.
- **Workspace** — Explore data that you create or import from files.
- **Command Window** — Enter commands at the command line, indicated by the prompt (>>).



```
>> a = 5
```

Name ▲	Value	Size	Class
a	5	1x1	double

Command Window

Workspace

whos COMMAND

DESCRIPTION

- Instead of the workspace, the properties of a variables can also be retrieved by typing whos command on the command window.

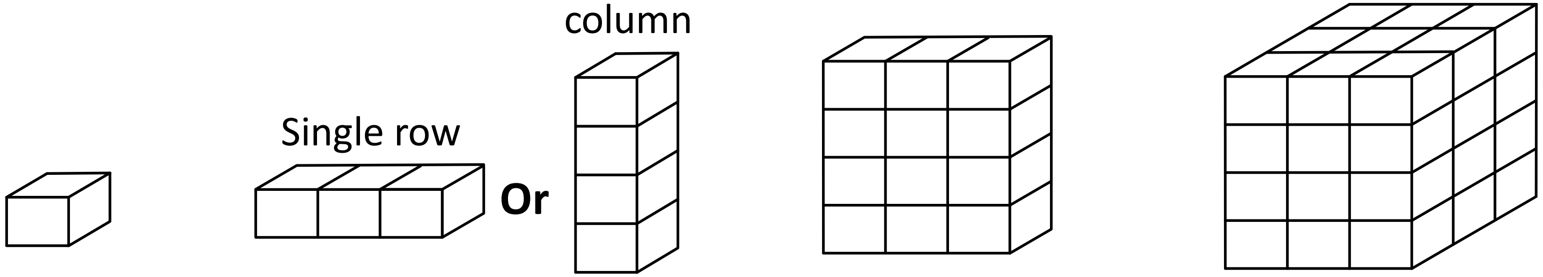
```
>> a = 5
a =
    5

>> whos
  Name           Size         Bytes       Class        Attributes

  a              1x1              8       double
```

ARRAY

DESCRIPTION

- Referring to the slide VARIABLES PROPERTIES, it is known that all variables are array.
- Two important basic on array are on how to **create** the array and how to **access** the elements within the array.
- Accessing array is an action to either **get** or **set** the elements of an array.
- Thus, we are going to discuss the create and access array topics before we go to the details on the data type and value type properties of the variables.
- The create and access array topics are obviously related to the array size.

ARRAY SIZE



Single row

Or

Single column

Scalar

Vector

Matrix

Multi-dimensional array

CREATE & ACCESS ARRAY METHOD

CREATE ARRAY

USING SQUARE BRACKET

```
a = [1,2;3 4]
```

SPACE / COMMA TO SEPARATE COLUMN ELEMENT

SEMICOLON / NEW LINE TO SEPARATE ROW ELEMENT

```
a = [1,2
     3,4]
```

USING COLON OPERATOR

```
a = j:m:k
a = j:k
a = [1:3;3:5]
```

ACCESSING ARRAY

SUBSCRIPT

1,1 2	1,2 4	1,3 1
2,1 10	2,2 9	2,3 5
3,1 7	3,2 8	3,3 15

```
a(1,1) = 2

a(3,2) = 8

a([1 2],1)
   = [2 10]
```

LINEAR INDEX

1 2	4 4	7 1
2 10	5 9	8 5
3 7	6 8	9 15

```
a(1) = 2

a(6) = 8

a([1 2])
   = [2 10]
```

CREATING ARRAY

CREATE ARRAY : COLUMN & ROW VECTOR

EXAMPLE 1

```
>> x = [1 2 3 4]
x =
     1     2     3     4

>> x = [1,2,3,4]
x =
     1     2     3     4
```

- To create a vector, enclose a list of values in bracket.
- Use space or comma as a delimiter in a row vector.

EXAMPLE 2

```
>> y = [1;2;3;4]
y =
     1
     2
     3
     4
```

- Use a semicolon as a delimiter to create a new row.

CREATE ARRAY : MATRIX

EXAMPLE 3

```
>> x = [1 2 3 4;2 3 4 5;3 4 5 6]
x =
     1     2     3     4
     2     3     4     5
     3     4     5     6
```

- Use a semicolon as a delimiter to create a new row.

EXAMPLE 4

```
>> x = [1 2 3 4;
2 3 4 5;
3 4 5 6]
x =
     1     2     3     4
     2     3     4     5
     3     4     5     6
```

- It’s easier to keep track of how many values you’ve entered into a matrix, if you enter each row on a separate line. The semicolons are optional.

CREATE ARRAY : COLON OPERATOR

EXAMPLE 5

```
>> x = 1:4
x =
     1     2     3     4

>> x = [1:4]
x =
     1     2     3     4
```

- Evenly spaced values matrices can be entered much more readily using colon operator.
- The bracket is optional for row vector array.

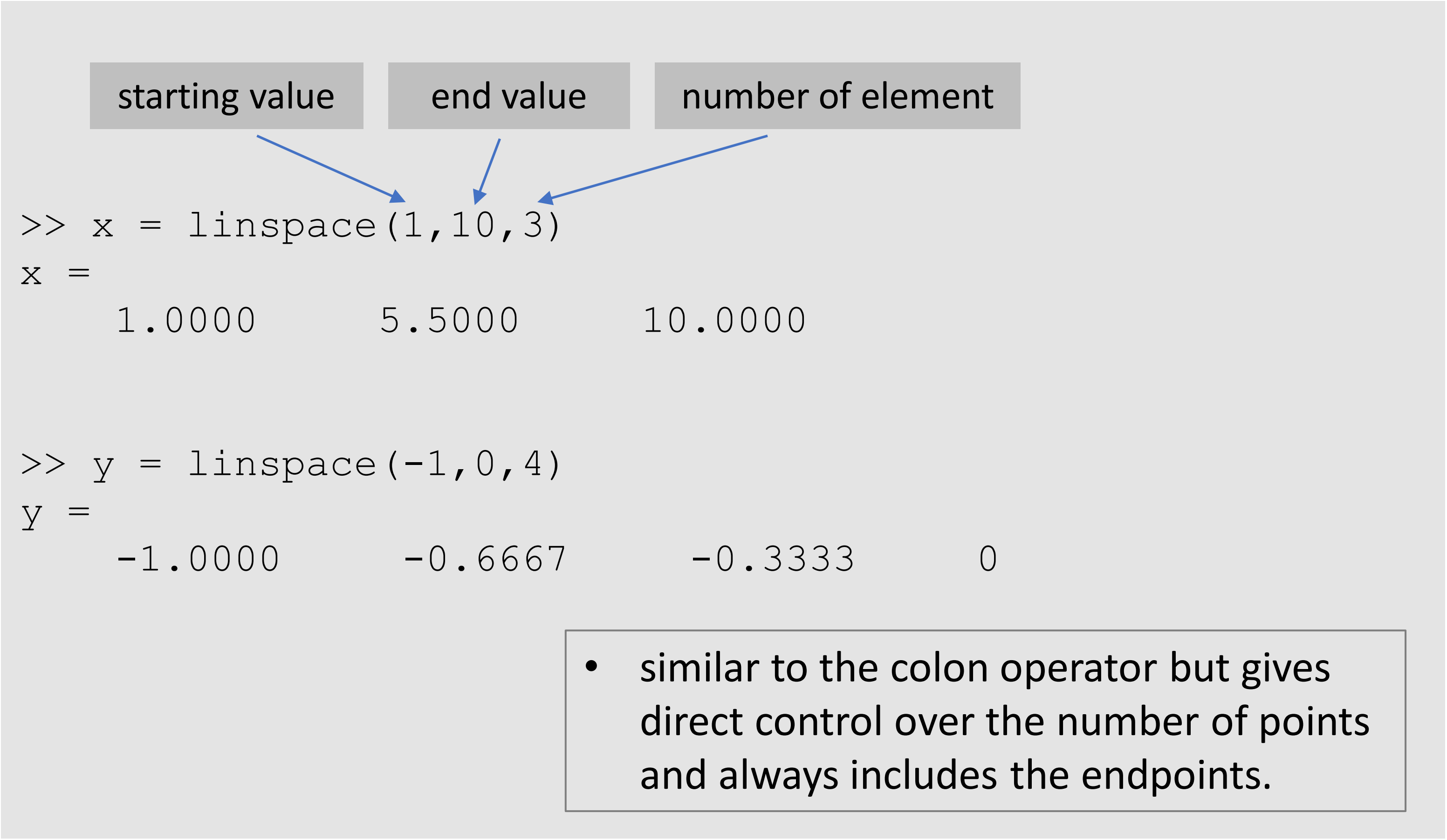
EXAMPLE 6

```
>> y = 1:2:5
y =
     1     3     5

>> y = 1:-1:-1
y =
     1     0    -1
```

- Use two colon operator to have increment other than 1.

CREATE ARRAY : LINSPACE

EXAMPLE 7



starting value

end value

number of element

```
>> x = linspace(1,10,3)
x =
    1.0000     5.5000     10.0000


>> y = linspace(-1,0,4)
y =
    -1.0000     -0.6667     -0.3333     0
```

- similar to the colon operator but gives direct control over the number of points and always includes the endpoints.

CREATE ARRAY : FROM OTHER ARRAY

EXAMPLE 8

```
>> a = [1 2]
a =
     1      2

>> b = [3 4 a]
x =
     3      4      1      2
```

- We can also create array from other array.

EXAMPLE 9

```
>> a = [1 2]
a =
     1      2

>> b = [3 4;a]
b =
     3      4
     1      2
```

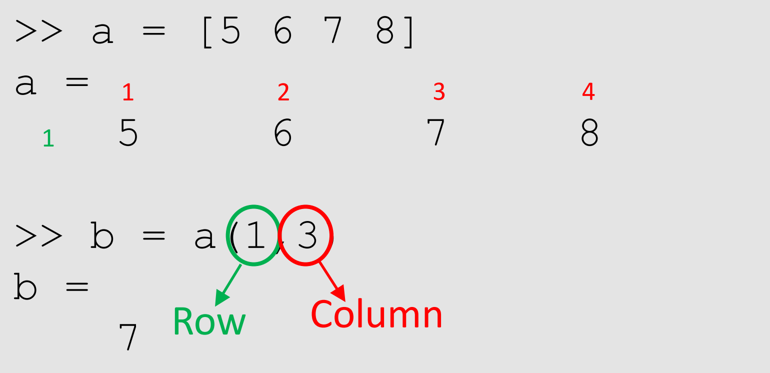
- When creating a new row, make sure the number of columns elements are similar for all rows.

ACCESSING ARRAY

SUBSCRIPTING : GET ELEMENT

Subscripting is based on the row and column position of an element.

EXAMPLE 1



```
>> a = [5 6 7 8]
a =  1     2     3     4
  1  5     6     7     8

>> b = a(1,3)
b =
     7
```

- ‘b’ is taking one element from ‘a’ at row 1 and column 3.

EXAMPLE 2

```
>> a = [1,2;3,4]
a =  1     2
  1  1     2
  2  3     4

>> b = a(2,1)
b =
     3
```

- ‘b’ is taking one element from ‘a’ at row 2 and column 1.

SUBSCRIPTING : SETTING ELEMENTS VALUE

EXAMPLE 3

```
>> a = [5 6 7 8]
a = 1     2     3     4
 1  5     6     7     8

>> a(1,2) = 0
a =
    5     0     7     8
```

- Second element of ‘a’ is replaced with new value.

EXAMPLE 4

```
>> a = [1,2;3,4]
a = 1     2
 1  1     2
 2  3     4

>> a(1,2) = 0
a =
    1     0
    3     4
```

- Element of ‘a’ at row 1 and column 2 is replaced with new value.

SUBSCRIPTING : COLON OPERATOR

To access more than one elements at once, use colon subscripting

`A(:,j)`	is the `j`th column of `A`.
`A(i,:)`	is the `i`th row of `A`.
`A(:,:)`	is the equivalent two-dimensional array. For matrices this is the same as `A`.
`A(:,j:k)`	is `A(:,j), A(:,j+1),...,A(:,k).`
`A(:,:,k)`	is the `k`th page of three-dimensional array `A`.
`A(i,j,k,:)`	is a vector in four-dimensional array `A`. The vector includes `A(i,j,k,1), A(i,j,k,2), A(i,j,k,3),` and so on.
`A(j:k)`	is `A(j),A(j+1),...,A(k).`
`A(:)`	is all the elements of `A`, regarded as a single column. On the left side of an assignment statement, `A(:)` fills `A`, preserving its shape from before.

`:`	Colon operator alone means all elements
`j:k`	Elements from `j` to `k`

SUBSCRIPTING : GET ELEMENTS USING COLON

EXAMPLE 5

```
>> a = [1 2 3;5 6 7];
a =  1      2      3
 1   1      2      3
 2   5      6      7

>> b = a(1,:)
b =
     1      2      3
```

- ‘b’ is all elements of the first row of ‘a’.

EXAMPLE 6

```
>> a = [1,2,3;3,4,5]
a =  1      2      3
 1   1      2      3
 2   3      4      5

>> b = a(:,2:3)
b =
     2      3
     4      5
```

- ‘b’ is taking elements of ‘a’ at all rows but limited to elements at column 2 to 3 only.

SUBSCRIPTING : CREATE HIGHER DIMENSIONAL ARRAY

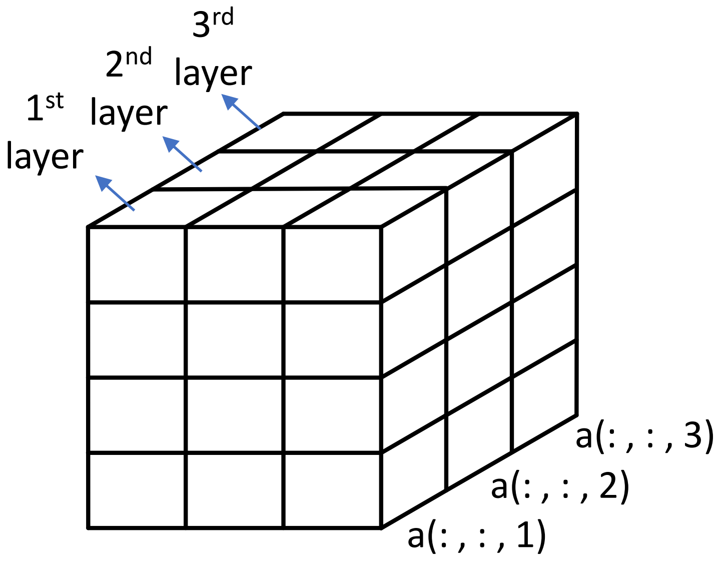
- Previously, we can only create an array up to matrix size, which is 2-D.
- To create array higher than the 2-D, we first create the matrix and extend the dimension using subscripting method.

EXAMPLE 7

```
>> a = [1,2,3;3,4,5]
a =
     1       2       3
     3       4       5

>> a(:,:,2) = 1
a(:,:,1) =
      1       2       3
      3       4       5

a(:,:,2) =
      1       1       1
      1       1       1
```

2nd layer

- ‘a’ is first create as a matrix. Then the third dimension is create by setting the third subscript for ‘a’.
- In this example, the second layer of the matrix is set to all one.
- On command window, the array is shown layer by layer.



Multi-dimensional array

LINEAR INDEXING

- Linear indexing refer to an element of an array based on a single integer number associated to the element.
- The index numbering starts from the top left element, moving to the last row and then continue to the next column.
- Below is an example of the index numbering on 3x3 array.

1	4	7
2	5	8
3	6	9

- Get elements using linear indexing return an array with similar size and arrangement to its indexing array.

LINEAR INDEXING : GET AN ELEMENT

EXAMPLE 1

```
>> a = [5 6 7 8]
a = 1         2         3         4
     5         6         7         8

>> b = a(3)
b =
     7
```

- ‘b’ is taking element no. 3 from ‘a’.

EXAMPLE 2

```
>> a = [1,2;3,4]
a = 1         3
     1         2
   2 3       4 4

>> b = a(2)
b =
     3
```

- ‘b’ is taking element no. 2 from ‘a’.

LINEAR INDEXING : SET AN ELEMENT

EXAMPLE 3

```
>> a = [5 6 7 8]
a =
   1         2         3         4
     5         6         7         8

>> a(2) = 0
a =
     5         0         7         8
```

- Second element of ‘a’ is replaced with new value.

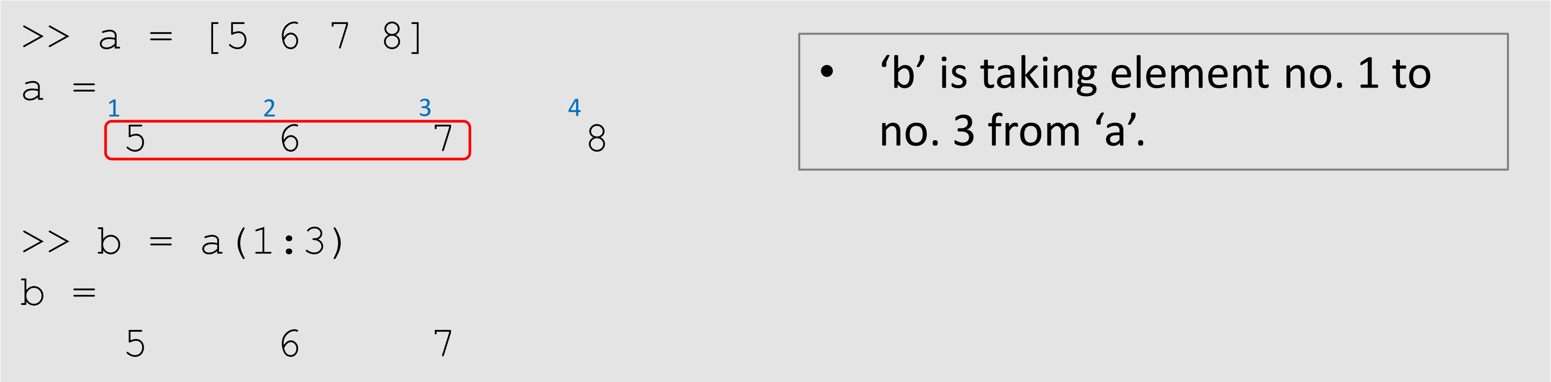
EXAMPLE 4

```
>> a = [1,2;3,4]
a =
   1         3
     1         2
   2         4
     3         4

>> a(3) = 0
a =
     1         0
     3         4
```

- Element no. 3 of ‘a’ is replaced with new value.

LINEAR INDEXING : GET ELEMENTS USING COLON

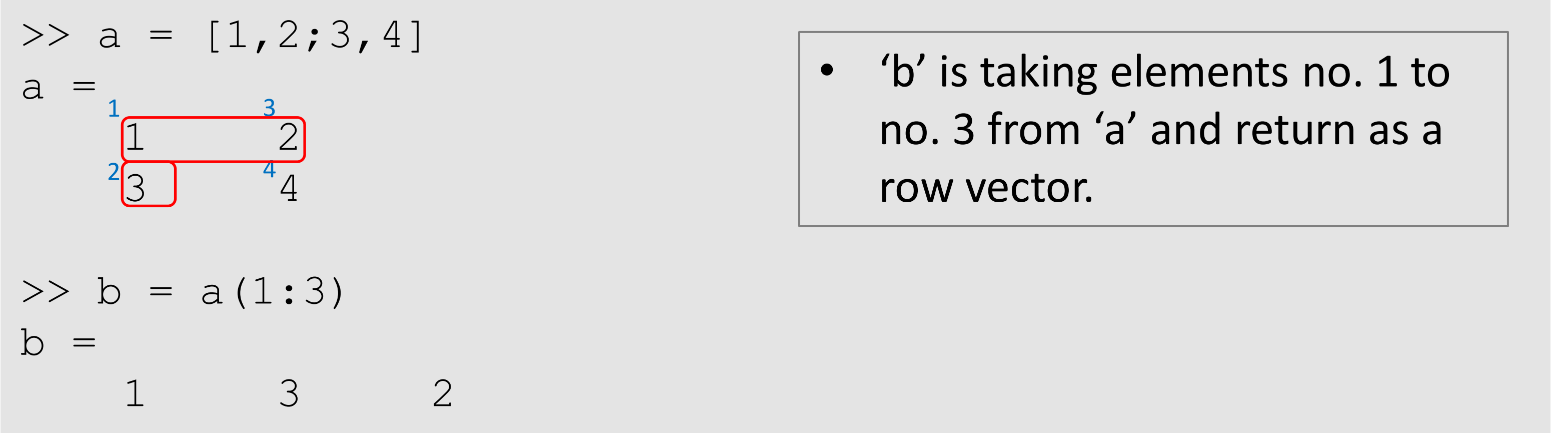
EXAMPLE 5



```
>> a = [5 6 7 8]
a =
     5     6     7     8

>> b = a(1:3)
b =
     5     6     7
```

- ‘b’ is taking element no. 1 to no. 3 from ‘a’.

EXAMPLE 6



```
>> a = [1,2;3,4]
a =
     1     2
     3     4

>> b = a(1:3)
b =
     1     3     2
```

- ‘b’ is taking elements no. 1 to no. 3 from ‘a’ and return as a row vector.

LINEAR INDEXING : CONVERT TO COLUMN VECTOR

EXAMPLE 7

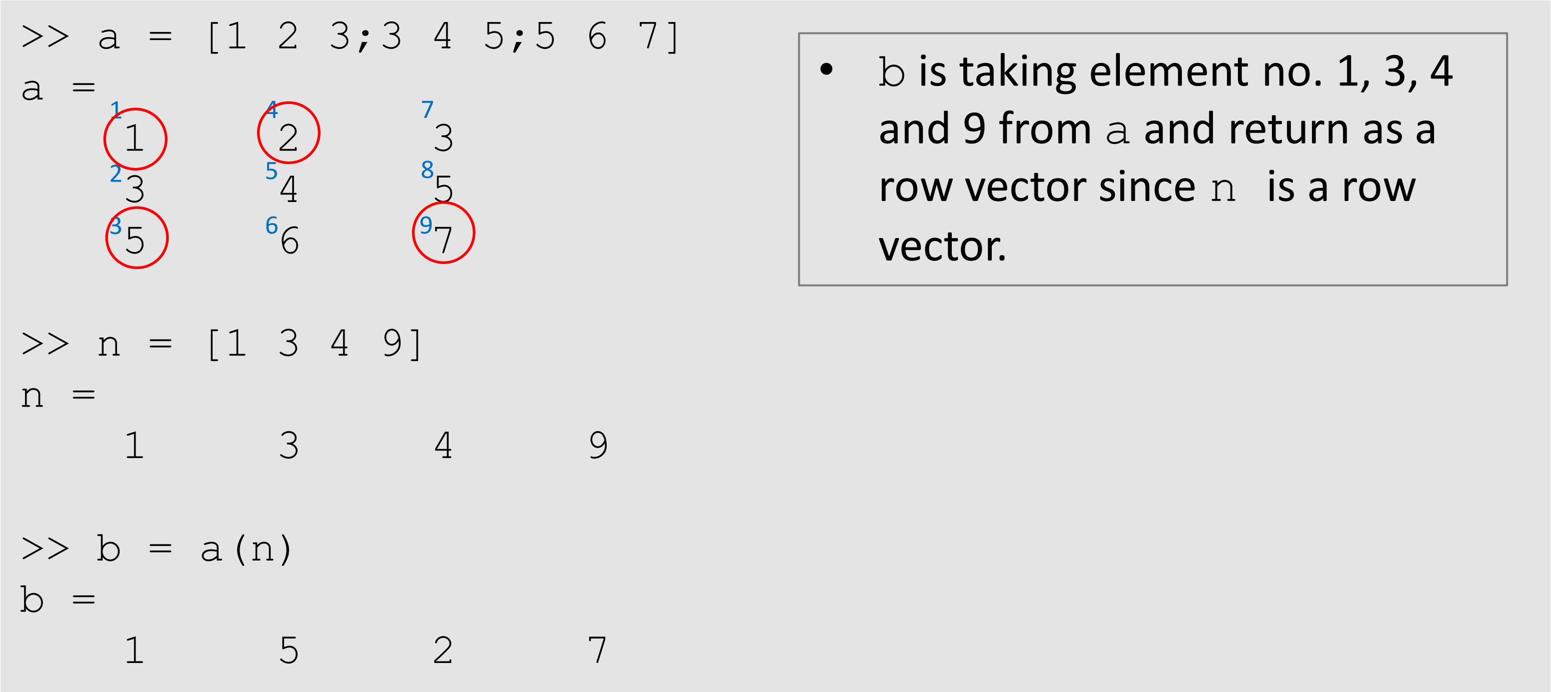
```
>> a = [1 2 3;3 4 5;5 6 7]
a =
   1          4          7
     1          2          3
   2          5          8
     3          4          5
   3          6          9
     5          6          7

>> b = a(:)
b =
     1     3     5     2     4     6     3     5     7
```

- ‘b’ is taking all elements from ‘a’ and return as a row vector.

LINEAR INDEXING : VECTOR INDICES

The indices can also be in the form of an array

EXAMPLE 8



```
>> a = [1 2 3;3 4 5;5 6 7]
a =
     1     2     3
     3     4     5
     5     6     7

>> n = [1 3 4 9]
n =
     1     3     4     9

>> b = a(n)
b =
     1     5     2     7
```

- b is taking element no. 1, 3, 4 and 9 from a and return as a row vector since n is a row vector.

LINEAR INDEXING : VECTOR INDICES

EXAMPLE 9

```
>> a = [1 2 3;3 4 5;5 6 7]
a =
    1 1       4 2       7 3
    2 3       5 4       8 5
    3 5       6 6       9 7

>> n = [3:7]
n =
     3        4        5        6        7

>> a(n) = 0
a =
     1        0        0
     3        0        5
     0        0        7
```

- Elements no. 3 to 7 of a are replaced with zero.

LINEAR INDEXING : MATRIX INDICES

EXAMPLE 10

```
>> a = [1 2 3;3 4 5;5 6 7]
a =
     1         4         7
      1         2         3
     2 3       5 4       8 5
     3 5       6 6       9 7

>> n = [1 2;5 7]
n =
      1         2
      5         8

>> b = a(n)
b =
      1         3
      4         5
```

- b is taking element no. 1, 2, 5 and 8 of array a.
- b is return as an array with similar size and arrangement to the indexing array n.

DATA TYPE

NUMERIC : FLOATING POINT & INTEGER

- Numeric class (data type) include signed and unsigned integer, single and double precision floating point number and complex number.
- By default, MATLAB stores all numeric values as double-precision floating point.

```
>> a = 5
```

- By default, this create variable ‘a’ with double-precision floating point.

- To create other numeric type, you need to type the function type,

```
>> a = int8(5)
```

- This create variable ‘a’ with signed 8-bit integer.
- Refer to MATLAB documentation for the full list of the numeric type.

NUMERIC : COMPLEX NUMBER

- In MATLAB, The special values i and j stand for √(−1) to represent imaginary number. For example:

```
>> x = 1+3*i
x =
    1.0000 + 3.0000i

>> x = 1+3i
x =
    1.0000 + 3.0000i
```

- The imaginary part of the complex number can be entered with or without the asterisk ‘*’.

- If `z` is a complex number, `real(z)`, `imag(z)`, `conj(z)`, and `abs(z)` all have the obvious meanings.
- A complex number may be represented in polar coordinates of $z = re^{j\theta}$ where `angle(z)` and `abs(z)` return the θ and r values respectively

CHARACTER ARRAY : STRING

- Character array data type is normally use to create string (a sequence of characters).
- We can create a string by enclosing a sequence of characters in single quotation marks.

```
>> myString = ‘Hello, World’
myString =
     Hello, World
```

```
>> otherString = ‘You’’re right’
otherString =
     You’re right
```

- If the text contains a single quotation mark, include two quotation marks within the string definition.

CHARACTER ARRAY : STRING SIZE

- By using whos command, we can observe that the array size of the string is the total number of characters of the string.

```
                1 2 3 4 5 6 7 8 9 10 11 12
>> myString = ‘Hello, World’
myString =
    Hello, World

>> whos myString
  Name          Size          Bytes     Class     Attributes

  myString      1x12             24     char
```

CHARACTER ARRAY : 2-D CHARACTER ARRAY

- To create a two-dimensional character array, make sure the number of elements on each row are the same.

```
         12345   123456   12345
>> Q = [‘Holly’;’Stevan’;’Megan’]
Error using vertcat
Dimensions of matrices being concatenated are not
consistent.
```

- To avoid the error, pad the lesser string with space.

```
>> Q = [‘Holly ’;
         ’Stevan’;
         ’Megan ’]
Q =
    Holly
    Stevan
    Megan
```

CHARACTER ARRAY : 2-D CHARACTER ARRAY

- Or we can use function char to automatically pad the string with spaces.

```
             12345   123456   12345
>> Q = char(‘Holly’,’Stevan’,’Megan’)
Q =
    Holly
    Stevan
    Megan
```

LOGICAL

- Logical value is a value indicating the truth condition.
- It has only two values, represented by either true or false.
- True is given value 1, while false is given value 0.
- Logical values are very useful in indexing and implementing decision statement (will be discussed next week).
- Numeric data type can be converted into logical data type using function logical. All values not equals to zero will be converted to true and zero value will be converted to false.

EXAMPLE 1

```
>> a = [1 3 0 -2 8 0];
>> b = logical(a)
b =
     1      1      0      1      1      0
```

LOGICAL : ARRAY INDEXING

- A logical array can be used as the index to an array.

EXAMPLE 2

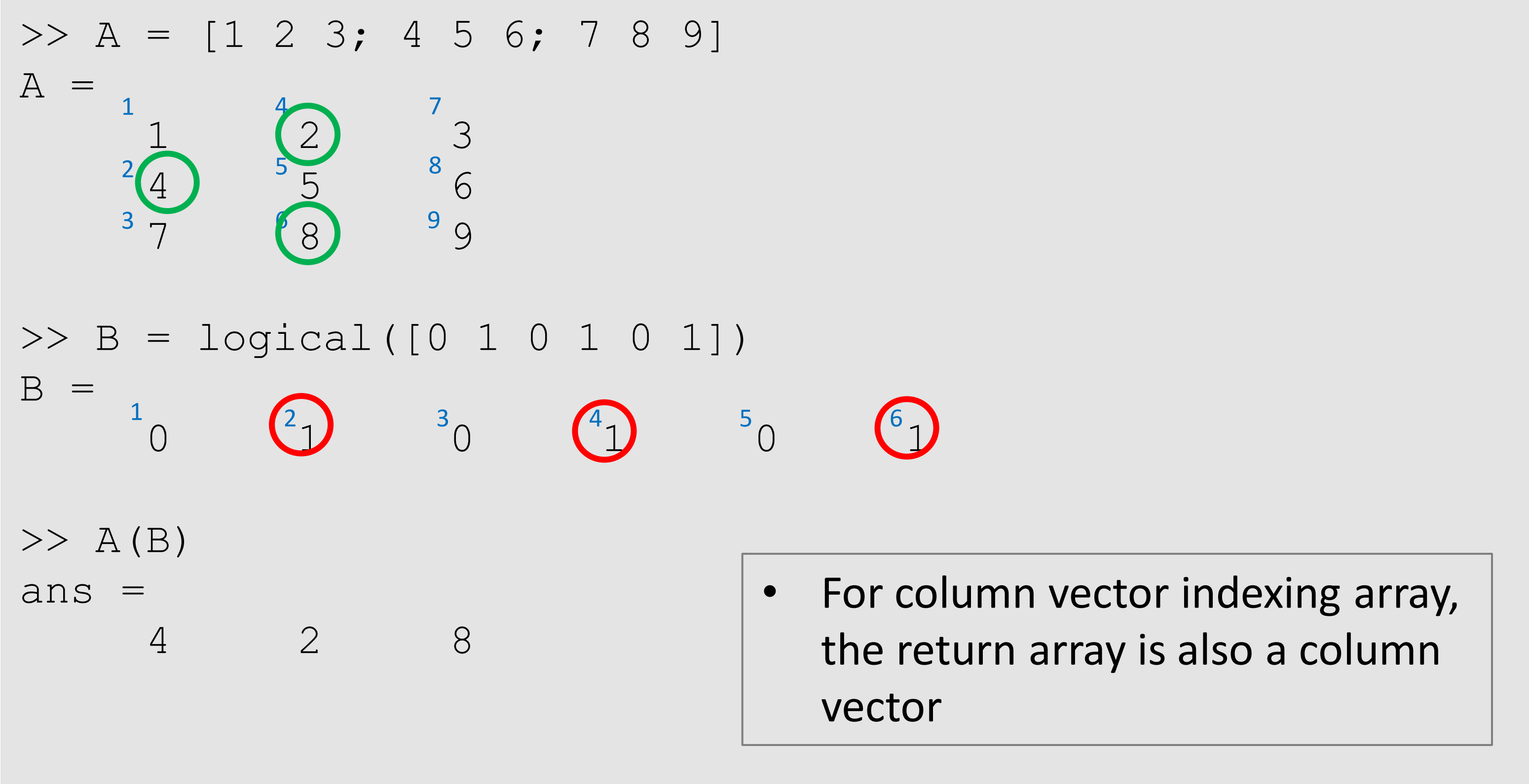
```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
       1     2     3
       4     5     6
       7     8     9

>> B = logical([0 1 0; 1 0 1; 0 0 1])
B =
       0     1     0
       1     0     1
       0     0     1

>> A(B)
ans =
       4
       2
       6
       9
```

- Every `true` element in the indexing array `B` is treated as a positional index into the array `A`.
- The logical indexing return array is always a column vector array except for row vector indexing array.

LOGICAL : ARRAY INDEXING

- A logical array can be used as the index to an array.

EXAMPLE 3



```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
      1      2      3
      4      5      6
      7      8      9

>> B = logical([0 1 0 1 0 1])
B =
      0      1      0      1      0      1

>> A(B)
ans =
      4      2      8
```

- For column vector indexing array, the return array is also a column vector

DATA TYPE CONVERSION

- Below are some of the function to convert between data types:

Function	Description
`Double, single`	Convert to double and single precision respectively
`int8,int16, int32,int64`	Convert to signed integer
`Uint8,uint16, uint32,uint64`	Convert to unsigned integer
`int2str`	Convert integer to string
`num2str`	Convert number to string
`str2double`	Convert string to double-precision value
`str2num`	Convert string to number
`mat2str`	Convert matrix to string
`logical`	Convert numeric values to logicals

DATA TYPE CONVERSION

EXAMPLE 1

```
>> a = 10.329
a =
     10.329

>> b = int8(a)
b =
     10
```

- Converting double to 8-bit integer is a way to round the value `10.329` to the nearest integer.

EXAMPLE 2

```
>> a = 256.57;

>> b = int8(a)
b =
     127

>> b = int16(a)
b =
     257
```

- Make sure to use the right integer conversion since every n-bit integer conversion has its limited range.
- In this example, `int8` has the maximum value of `127`. Value greater than the limit will be capped.

Maximum value determination:

- int8 has total stored value of $2^8 =$ 256. As it is signed integer, the value can goes between -127 to 127.
- uint8 has total stored value of $2^8 =$ 256. As it is unsigned integer, the value can goes between 0 to 256.

DATA TYPE CONVERSION

EXAMPLE 3

```
>> str = '123'
str =
     123

>> a = str2num(str)
a =
     123

>> b = double(str)
b =
     49    50    51
```

- If a string is a number, use `str2num` to convert it into numeric.
- If `double` is use, it will convert every char of '123' into its numeric value.

CONSTANT

CONSTANT

- Constant is a value, predetermined by MATLAB.

Constant	Description
`i, j`	Imaginary unit
`pi`	Ratio of circle's circumference to its diameter
`Inf`	Infinity
`NaN`	Not-a-Number

- Other than the above scalars, there are also matrix type constants.

Constant	Description
`magic`	Magic square
`hadamard`	Hadamart matrix
`hilb`	Hilbert matrix

CONSTANT

EXAMPLE 1

```
>> a = pi
a =
     3.1416

>> b = sin(pi)
b =
     1.2246e-16
```

- The expression `sin(pi)` is not exactly zero because pi is not exactly π.

EXAMPLE 2

```
>> a = 1/0
a =
     Inf

>> b = 0/0
b =
     NaN
```

- NaN is a representation of a numeric value other than infinity that can not be defined.

NaN

These operations produce NaN:

1. Any arithmetic operation on a NaN, such as sqrt(NaN)
2. Addition or subtraction, such as magnitude subtraction of infinities as (+Inf)+(-Inf)
3. Multiplication, such as 0*Inf
4. Division, such as 0/0 and Inf/Inf
5. Remainder, such as rem(x,y) where y is zero or x is infinity

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

univteknologimalaysia

utm_my

utmofficial

Thank You

www.utm.my

innovative ● entrepreneurial ● global