الكلية	العلوم
القسم	الرياضيات
المادة باللغة الانجليزية	Programming (MATLAB)
المادة باللغة العربية	البرمجة بلغة (ماتلاب)
المرحلة الدراسية	الثانية
اسم التدريسي	صفوت عبدالقادر حمد
عنوان المحاضرة باللغة الانجليزية	Data Type & Variable
عنوان المحاضرة باللغة العربية	انواع البيانات والمتغيرات
رقم المحاضرة	الثانية
المصادر والمراجع	MATLAB A Practical Introduction to Programming and Problem Solving
	MATLAB The Language of Technical Computing
	MATLAB numerical computing

University Of Anbar

College Of Science

Math Department

MATRIX - LABORATORY (MATLAB)

Lecture 2

Data Type & Variable

By:

Safwat A. Hamad

For the 2nd stage Math Department

1. Hands on Practice

MATLAB environment behaves like a super-complex calculator. You can enter commands at the >> command prompt. MATLAB is an interpreted environment. In other words, you give a command and MATLAB executes it right away.

In this section we learn how to perform the simple arithmetic operations of addition, subtraction, multiplication, division, and exponentiation. The first four operations of addition, subtraction, multiplication, and division are performed in MATLAB using the usual symbols +, -, *, and /.

- **The following are some examples:**
  - Addition `>> 3+6`
  - Subtraction `>> 5-6`
  - Multiplication `>> 5* 6`
  - Division `>> 4/9`
  - exponentiation `>> 3^6`
  - Parentheses `>> (5+5)*8`

- **Arithmetic Operations with the MATLAB Symbolic Math Toolbox**

Symbolic arithmetic operations may also be performed in MATLAB using the MATLAB Symbolic Math Toolbox. For the next examples, this toolbox needs to be installed with MATLAB. In order to perform symbolic operations in MATLAB, we need to define symbolic numbers using the ***sym*** command. For example, (½) is defined as a symbolic number as follows:

```
>> sym (½)  ??

>> (1/2) + (3/5)    ??

>> sym (1/2) + (3/5)    ??
```

Notice in the above example how the final answer was cast in the form of a fraction without decimal digits and without calculating a numerical value. In the

addition of the two fractions above, MATLAB finds their common denominator and adds them by the usual procedure for rational numbers. **Notice** also in the above three example outputs that symbolic answers are not indented but numerical answers are indented.

- **Use of Semicolon (;) in MATLAB**

Semicolon (;), indicates end of statement. However, if you want to suppress and hide the MATLAB output for an expression, add a semicolon after the expression.

```
>> x = 3    enter ??
```

```
>> x = 3 ; y = 6;   enter ??
```

- **Adding Comments**

The percent symbol (%) is used for indicating a comment line. For example,

```
>> x = 6   % assign value 6 in variable x
```

2. Data Type in MATLAB

MATLAB supports various data types for handling different kinds of data. Data types in MATLAB are important because they determine how data is stored, represented, and manipulated in memory. And the MATLAB has a variety of data types, which can be classified into two main categories: numeric and non-numeric.

- **Numeric data types represent numbers, and include:**
  - ***Double***: The default numeric data type in MATLAB, representing double-precision floating-point numbers. Double-precision provides a high level of accuracy and is suitable for most numerical computations.
  - ***Single***: Represents single-precision floating-point numbers, which use less memory than double-precision but with reduced precision.
  - ***Integer***: Integer data types can represent whole numbers, and include signed and unsigned integers of different sizes.

➢ Non - Numeric data types

Non-numeric data types represent other types of data, such as text, logical values, and dates and times. Some common non-numeric data types include:

- ***Char:*** Character arrays store individual characters.

```
>> ('Hello')
```

- ***String:*** String arrays store sequences of characters.

```
>> ("Hello")
```

- ***Logical:*** Logical values can be either true or false.
- ***Date and time:*** Arrays of date and time values.

To determine the data type of a variable in MATLAB, you can use the *class* function. For example, to determine the data type of the variable ***(x)***, you would use the following code:

```
>> class (x)
```

This would return the string '***double***', if *x* is a double-precision floating-point number. In addition, You can also use the *isa* function to determine if a variable has a specific data type. For example, to determine if the variable *(x)* is a double-precision floating-point number, you would use the following code:

```
>> isa(x, 'double')
```

This would return (true) if *x* is a double-precision floating-point number, and (false) otherwise. It is important to choose the correct data type for your variables, to ensure that your code is efficient and accurate. For example, if you are storing a large number of values, you may want to use an integer data type instead of a double-precision floating-point data type, to save memory.

- **Data Type Conversion:**
  - MATLAB allows you to convert between data types using functions like double(), single(), int8(), char(), and others.
  - Be mindful of potential data loss when converting between data types with different precision. Exp:

```
>> x_single = single(3.44577635);
>> x_single
x_single =
    single
    3.4458
>> class(x_single)
ans =
    'single'
>> x_double = double(x_single)
x_double =
    3.4458
```

* convert variable from double into single (in MATLAB).

3. Variables

Variables are fundamental elements in programming that serve as named storage locations for data. They are used to store and manipulate information within a computer program. Variables have a name, a data type, and a value, and they play a crucial role in data handling and computation.

- **Significance of Variables in Programming:**
  - **Data Storage:** Variables allow programmers to store data, such as numbers, text, and more complex data structures like arrays and objects. This stored data can be used throughout the program's execution.
  - **Data Manipulation:** Variables are used to perform operations and calculations on data. Programmers can add, subtract, multiply, divide, or apply more complex algorithms to the data stored in variables.
  - **State Management:** Variables help in maintaining the state or condition of a program. For example, they can track user progress in a game, store configuration settings, or remember user inputs.
  - **Interactions:** Variables are a way for programs to interact with users. They can store user inputs and display results back to users.
  - **Code Readability:** Variables make code more human-readable and easier to understand. Instead of working with raw values, you use meaningful variable names, which improve the clarity of your code.
  - **Reusability:** Variables promote code reusability. Once data is stored in a variable, you can use it in different parts of the program without duplicating the same information.

- **Variable Naming Rules**

There are the variable naming rules and best practices in MATLAB:

  - **Case Sensitivity**: MATLAB variable names are case-sensitive. This means that "myVariable" and "myvariable" are treated as two different variables. It's essential to be consistent in your capitalization.
  - **Start with a Letter:** Variable names must begin with a letter (A-Z, a-z). They cannot start with a digit (0-9) or special character.
  - **Valid Characters:** Variable names can include letters, digits, and underscores. Special characters like spaces, hyphens, and punctuation marks **are not allowed.**

- **Reserved Keywords:** Avoid using reserved keywords (e.g., if, for, end) as variable names, as they have predefined meanings in MATLAB.
- **Length Limitation:** Variable names in MATLAB can be relatively long (up to 63 characters), but it's advisable to keep them concise while maintaining clarity.

➢ **Creating and Assigning Variables:**

In start, If you do not use a variable, then MATLAB automatically stores the calculated result in the variable ans. The following is an example:

```
>> 4+5

ans =

     9
```

If you wish to store the number 5 in another variable, then you need to specify it as follow:

```
>> x = 3+2

x =

5
```

The variable is always on the left, followed by the (=) symbol, which is the assignment operator (unlike in mathematics, the single equal sign does not mean equality), followed by an expression. The expression is evaluated and then that value is stored in the variable. Here is an example and how it would appear in the Command Window:

```
>> mynum = 6

mynum =

6
```

Here, the user (the person working in MATLAB) typed “mynum = 6” at the prompt, and MATLAB stored the integer 6 in the variable called mynum, and then

displayed the result followed by the prompt again. As the equal sign is the assignment operator, and does not mean equality, the statement should be read as “mynum gets the value of 6” (not “mynum equals 6”).

Note that the variable name must always be on the left, and the expression on the right. An error will occur if these are reversed.

```
>> 6 = mynum

6 = mynum

"

Error: Incorrect use of '=' operator. To assign a value
to a variable,

use '='. To compare values for equality, use '=='.
```

* The spaces in a statement or expression do not affect the result, but make it easier to read.

To change a variable, another assignment statement can be used, which assigns the value of a different expression to it. Consider, for example, the following sequence of statements:

```
>> mynum = 3

mynum =

3

>> mynum = 4 + 2

mynum =

6

>> mynum = mynum + 1

mynum =

7
```

At this point, if the expression mynum+3 is entered, the default variable *ans* is used since the result of this expression is not assigned to a variable. Thus, the value of *ans* becomes 10, but *mynum* is unchanged (it is still 7). Note that just typing the name of a variable will display its value (the value can also be seen in the Workspace Window).

```
>> mynum + 3

ans =

10

>> mynum

mynum =

7
```

- **Initializing, Incrementing, and Decrementing**

Frequently, values of variables change, as shown previously. Putting the first or initial value in a variable is called initializing the variable. Adding to a variable is called incrementing. For example, the statement

```
>> mynum = mynum + 1
```

increments the variable *mynum* by 1.
