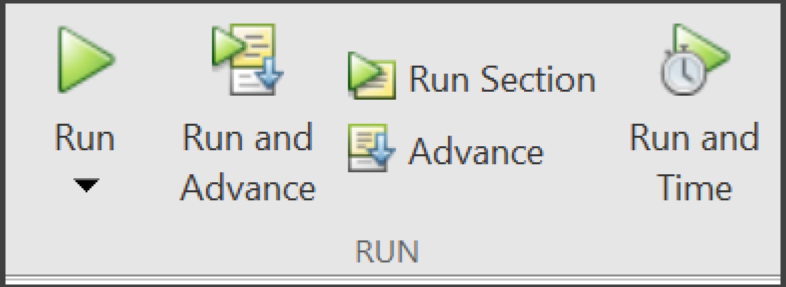
3강 - MATLAB Programming

목차

Running MATLAB code

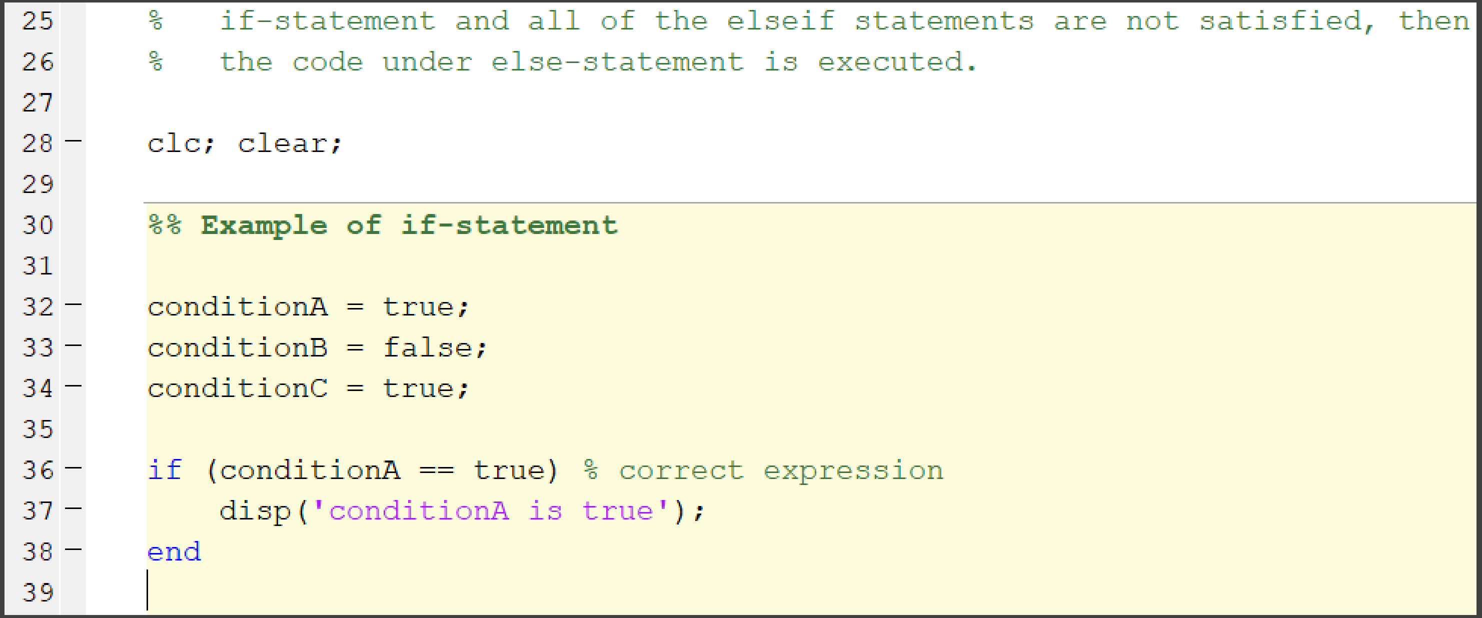
- How to run a program
  - Run – execute all the code!
  - Run and Advance – execute codes in 1 section, and then pause before starting the next one!
  - Run Section – execute the code in 1 section, and exit.
  - Advance – (we don't use it often)
  - Run and Time – Measuring time! (but if you need fast time measurement… use C++)



Running MATLAB code

- How to make a section
  - Use %%

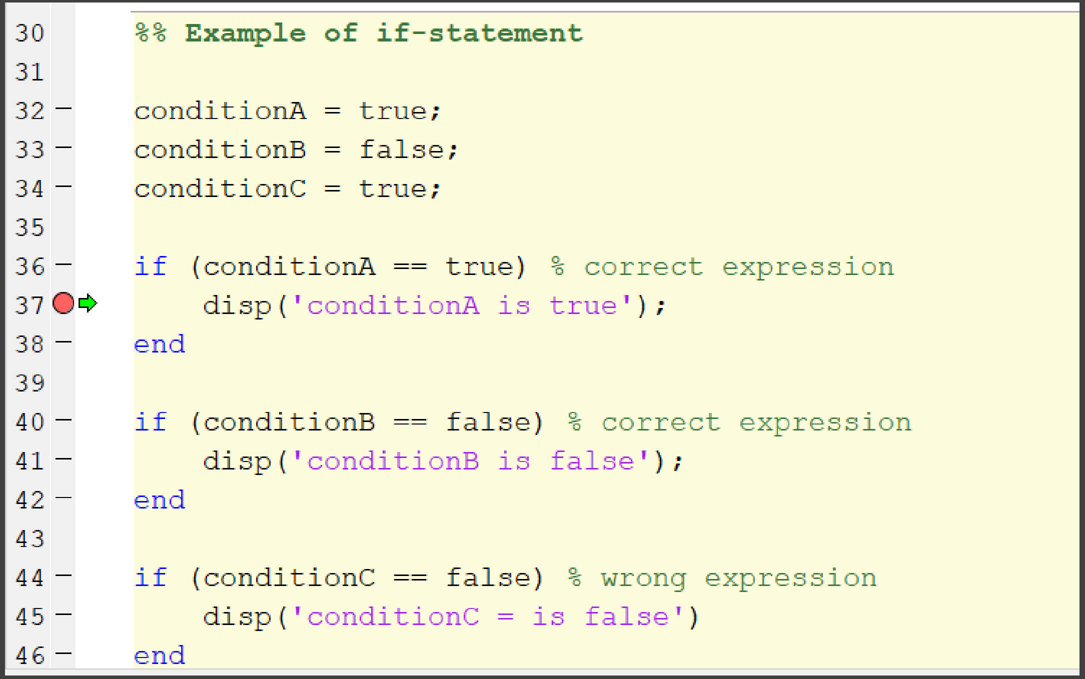
- How to comment
  - Use %



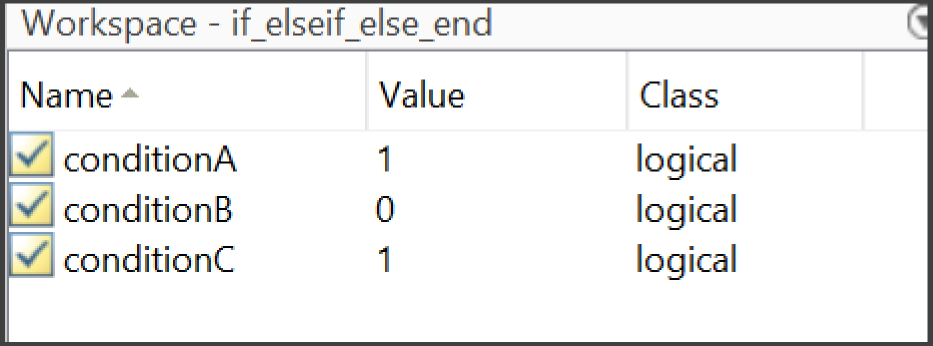
```
%   if-statement and all of the elseif statements are not satisfied, then
%   the code under else-statement is executed.

clc; clear;

%% Example of if-statement

conditionA = true;
conditionB = false;
conditionC = true;

if (conditionA == true) % correct expression
    disp('conditionA is true');
end

```

How to debug

- Red dot - Pauses code execution for debugging
  - You can check the values of all variables in the workspace
  - You can add more code during debugging
    - How? -> Because MATLAB uses interpreter, not compiler. C++ cannot do this.



```
%% Example of if-statement

conditionA = true;
conditionB = false;
conditionC = true;

if (conditionA == true) % correct expression
    disp('conditionA is true');
end

if (conditionB == false) % correct expression
    disp('conditionB is false');
end

if (conditionC == false) % wrong expression
    disp('conditionC = is false')
end
```



Workspace - if_elseif_else_end

Name	Value	Class
✓ conditionA	1	logical
✓ conditionB	0	logical
✓ conditionC	1	logical

How to debug

- Make use of clc; clear;
  - clc - clears the Command Window
  - clear - clears the workspace
    - Be careful when using this! Hours of work and data can be wiped ☺
- A good tip
  - Make a section in the beginning using %%
  - Write clc; clear; in the section.
  - Experiment with rest of the code using 'Run Section'

Variable / Function naming conventions

- There are two ways to name a variable / function :
  - numberOfPeople
  - number_of_people

- Always write a unambiguous name for a variable / function
  - vec1, mat1... (X)
  - vNumOfPeople, mInputImage (O)

Variable / Function naming conventions

- No spaces in a variable / function name
  - addNumbers() (O)
  - add numbers() (X)

- A number cannot go as the first letter of a variable
  - num1 (O)
  - firstNum (O)
  - 1num (X)

Variable / Function naming conventions

- For variables, we commonly use these terms
  - k~ suffix for constant values
    - kPI = 3.141592654…
    - kSpeedConstant = 3.0;
  - is~ suffix for Boolean values
    - isBirthday = true;
  - (MATLAB) v~ suffix for vector variables
  - (MATLAB) m~ suffix for matrix variables
  - (C++) m~ for class member variables

Variable / Function naming conventions

- For functions, we commonly use these terms
  - get~ suffix for a getter function
  - set~ suffix for a setter function

  - identity() for getting identity matrix

- We start with a verb for function names
  - filterData()
  - addNumber()
  - multiplyMatrix()

Matrix Data Handling

- In MATLAB, making a vector / Matrix requires filling in values at initialization.
- How to make a vector
  - Vec = [1 2 3 4];
  - Vec = [1, 2, 3, 4];
  - Vec = 1:4 ; (we don't use this often)
  - Vec = 1:2:100 (Only in MATLAB)
  - Vec = linspace(1,100,2) (Only in MATLAB)
  - Vec = zeros(5,1) (Use this!)
  - Vec = NaN(5,1) (Or use this!)

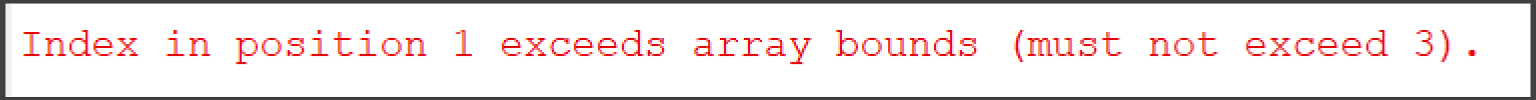
Matrix Data Handling

- How to make a matrix
  - Mat = [1,2,3 ; 4,5,6 ; 7,8,9];
  - Mat = zeros(3,3)
  - Mat = NaN(3,3)

Matrix Data Handling

- Accessing data in a vector / matrix
  - data   Vec(element)
  - data = Mat(row,column)

- What happens when you do Mat(10,10) on a 3x3 matrix?
  - Most common mistake in MATLAB appears… (ㅋㅋ)



```
Index in position 1 exceeds array bounds (must not exceed 3).
```

Matrix Data Handling

- Matrix addition / subtraction
  - mat3 = mat1 + mat2;
- Matrix multiplication
  - mat3 = mat1 * mat2;
- Element-wise operations
  - mat3 = mat1 .* mat2;
- Scalar multiplication on a matrix
  - mat2 = val * mat1;

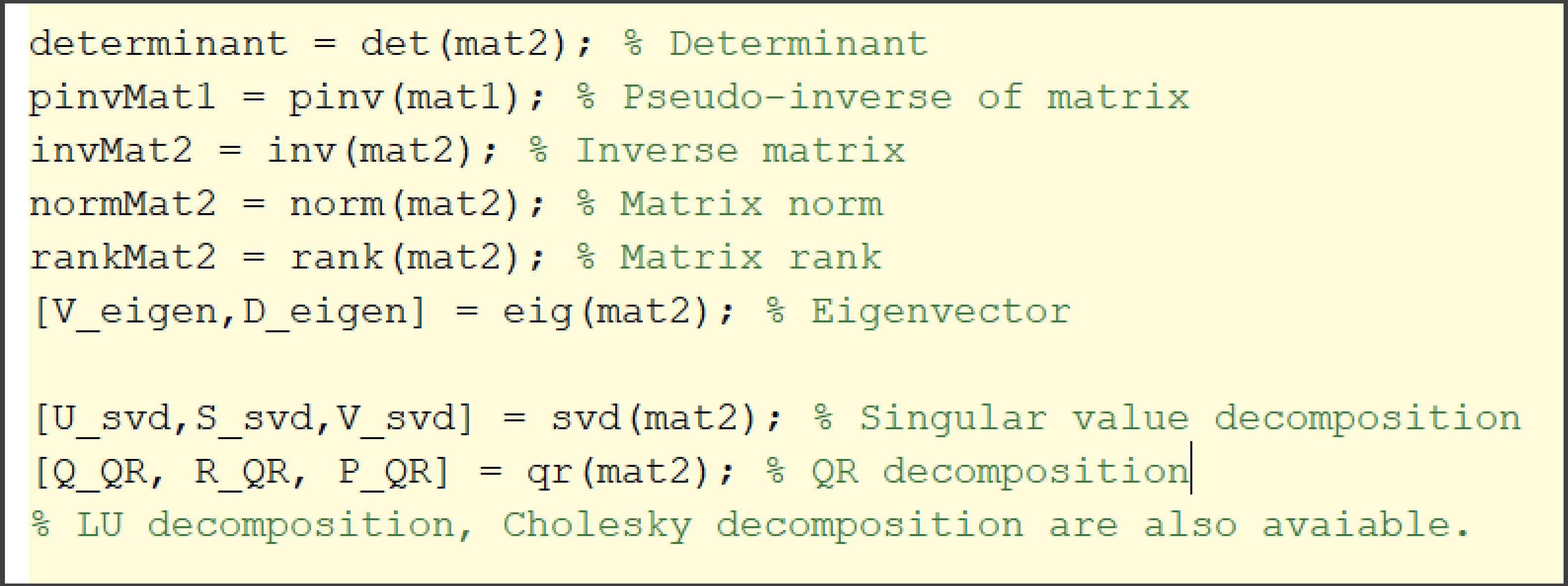
Matrix Data Handling

- Reshape(mat, [row col])
  - Changes shape of matrix

- mat’
  - transpose

Matrix Data Handling

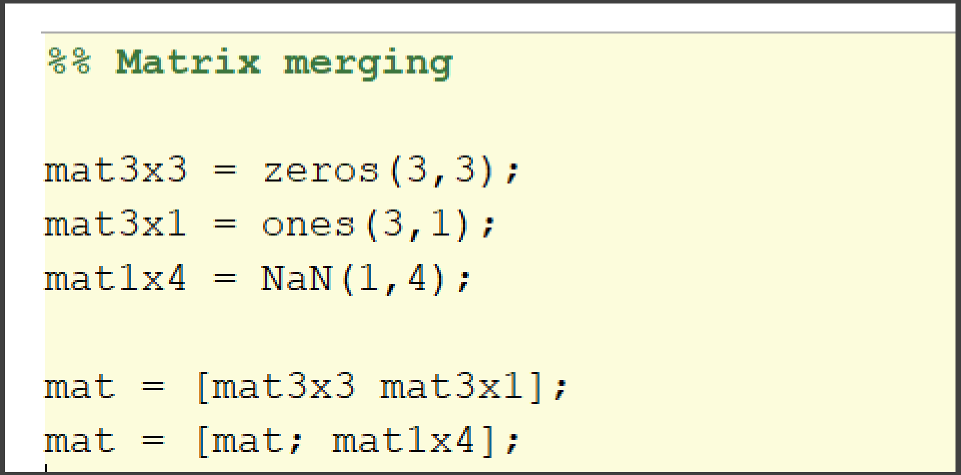
- Other maths functions



```
determinant = det(mat2); % Determinant
pinvMat1 = pinv(mat1); % Pseudo-inverse of matrix
invMat2 = inv(mat2); % Inverse matrix
normMat2 = norm(mat2); % Matrix norm
rankMat2 = rank(mat2); % Matrix rank
[V_eigen,D_eigen] = eig(mat2); % Eigenvector

[U_svd,S_svd,V_svd] = svd(mat2); % Singular value decomposition
[Q_QR, R_QR, P_QR] = qr(mat2); % QR decomposition
% LU decomposition, Cholesky decomposition are also avaiable.
```

Matrix Data Handling

- Matrix merging
  - Concat
  - vertConcat
  - horzConcat…
  - Or…



```
%% Matrix merging

mat3x3 = zeros(3,3);
mat3x1 = ones(3,1);
mat1x4 = NaN(1,4);

mat = [mat3x3 mat3x1];
mat = [mat; mat1x4];
```

Boolean and conditional statements

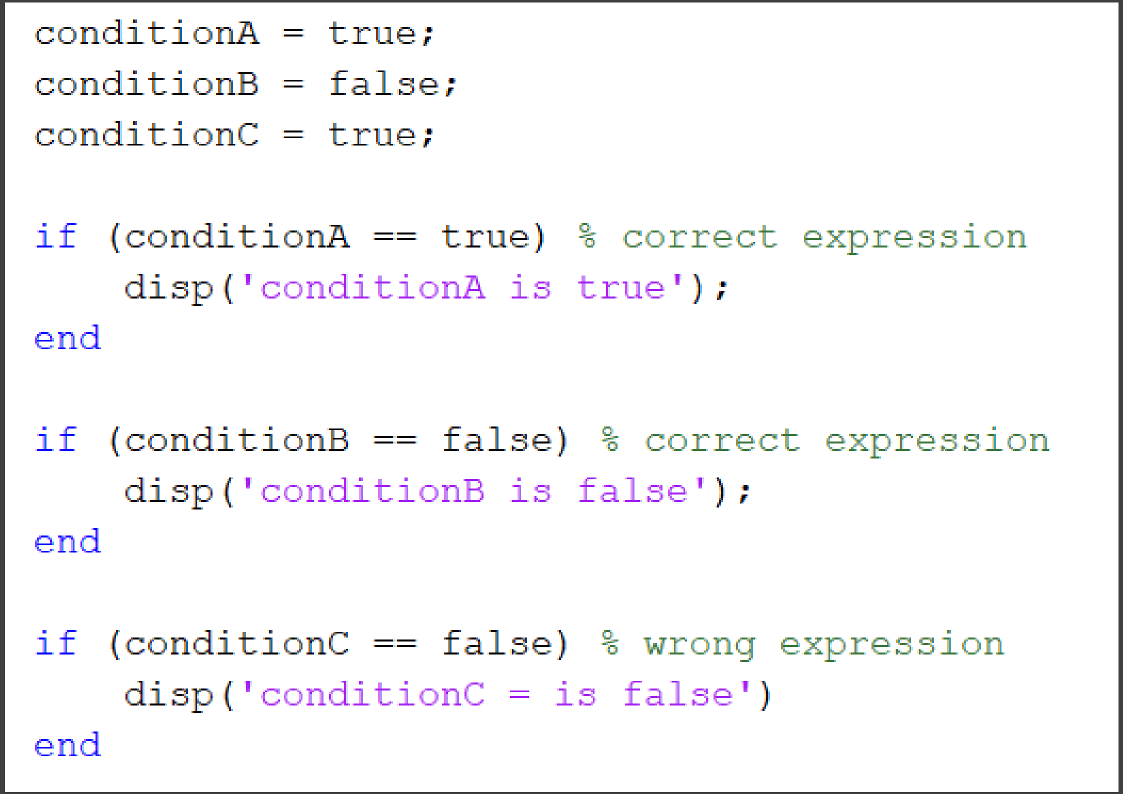
- Boolean = true / false

- Conditional operations

==	is equal to
< / >	is less / greater than
<= / >=	is less or equal to / is greater or equal to

Boolean and conditional statements

- Conditional statements (if-statement)
  - Executes code only when the condition is satisfied.



```
conditionA = true;
conditionB = false;
conditionC = true;

if (conditionA == true) % correct expression
    disp('conditionA is true');
end

if (conditionB == false) % correct expression
    disp('conditionB is false');
end

if (conditionC == false) % wrong expression
    disp('conditionC = is false')
end
```

Boolean and conditional statements

- if-statement
- If-else statement
- If-elseif-else statement

... Refer to the matlab code.

Iterations and for-loops

- For-loops use an iterator value

- (MATLAB) for i = 1:10
  - ...means the iterator value starts at 1, and increases by 1 every iteration, until it reaches 10.

- (C++) for (int i = 0; i < 10 ; i++)
  - ... means the iterator value starts at 0, and performs i++ at the end of every iteration, and escapes the loop when i < 10 is no longer satisfied.

Iterations and for-loops

- With for-loop we can…
  - Easily automate repetitive operations
    - e.g. vector indexing, matrix indexing
  - Easily load / save data
  - Easily log data

While loops

- While loops execute codes repetitively, as long as the condition is satisfied.

- In most cases, we use while loops for
  - Optimization
  - Manual user-in-loop control

Functions

- We can make user-defined functions

- Why do we want functions?
  - To avoid repetitions of commands.
  - Without functions, we need to write all the codes in a single script.

- We should separate the code into
  - 1. main script
  - 2. collection of functions

Functions

-
  ```
  function myFunc(parameter1, parameter2)
          operations...
  end
  ```

- Need to be located at the bottom of the script.