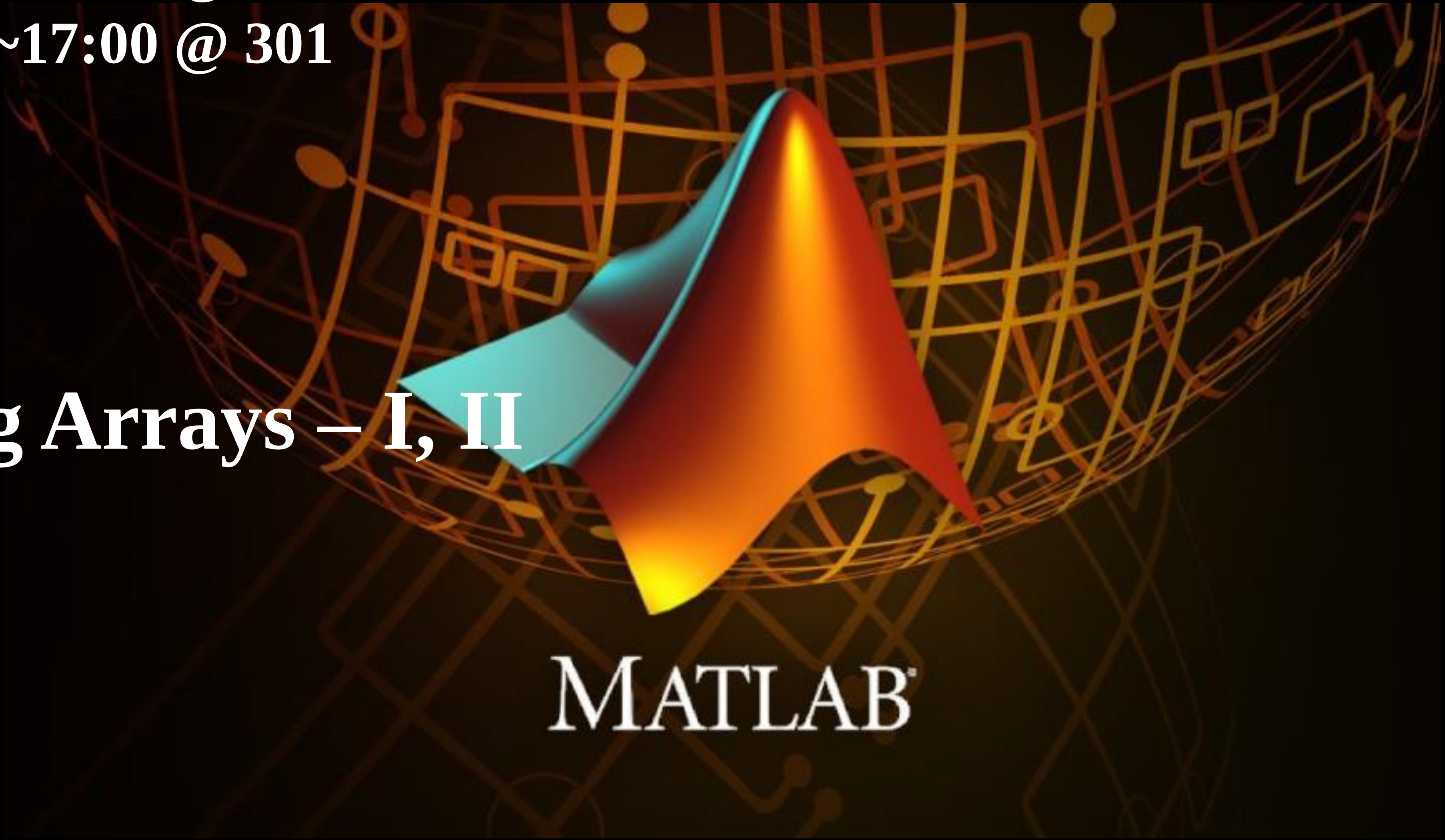
Introduction to MATLAB
Mon. 15:00~17:00 @ 301
Wed. 15:00~17:00 @ 301

03.
Creating Arrays – I, II



Seung-Tae Yoon

Department of Oceanography, KNU

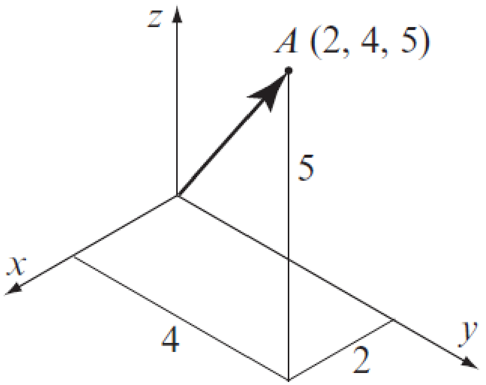
※ In this week,

✓Create a matrix

✓Handle a matrix

✓Use strings as variables

2.1 Creating a one-dimensional array (vector)



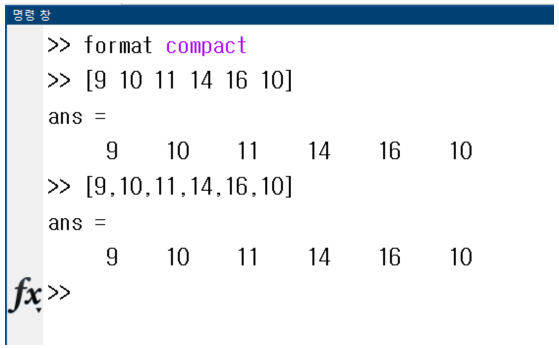
- A one-dimensional array is a list of numbers arranged in a row or a column.

- One example is the representation of the position of a point in space in a three dimensional Cartesian coordinate system.

- The vector is created by typing the elements (numbers) inside square brackets [ ].

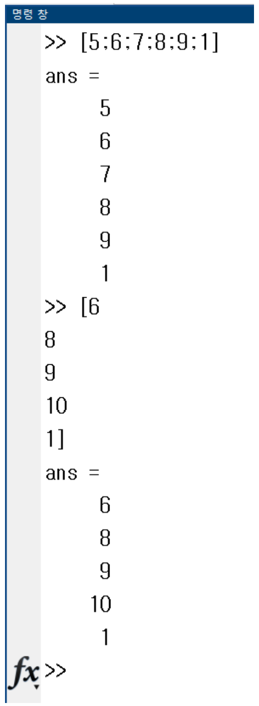
```
variable_name = [ type vector elements ]
```

- Row(행) vector: To create a row vector type the elements with a space or a comm between the elements inside the square brackets.



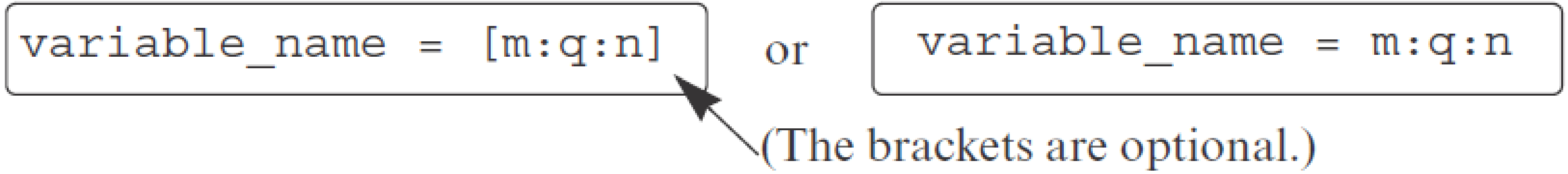
```
>> format compact
>> [9 10 11 14 16 10]
ans =
     9    10    11    14    16    10
>> [9,10,11,14,16,10]
ans =
     9    10    11    14    16    10
>>
```

- Column(열) vector: To create a column vector type the left square bracket [and then enter the elements with a semicolon between them, or press the **Enter** key after each element. Type the right square bracket] after the last element.



```
>> [5:6:7:8:9:1]
ans =
     5
     6
     7
     8
     9
     1
>> [6
8
9
10
1]
ans =
     6
     8
     9
    10
     1
>>
```

- Creating a vector with constant spacing by specifying the first term, the spacing, and the last term:

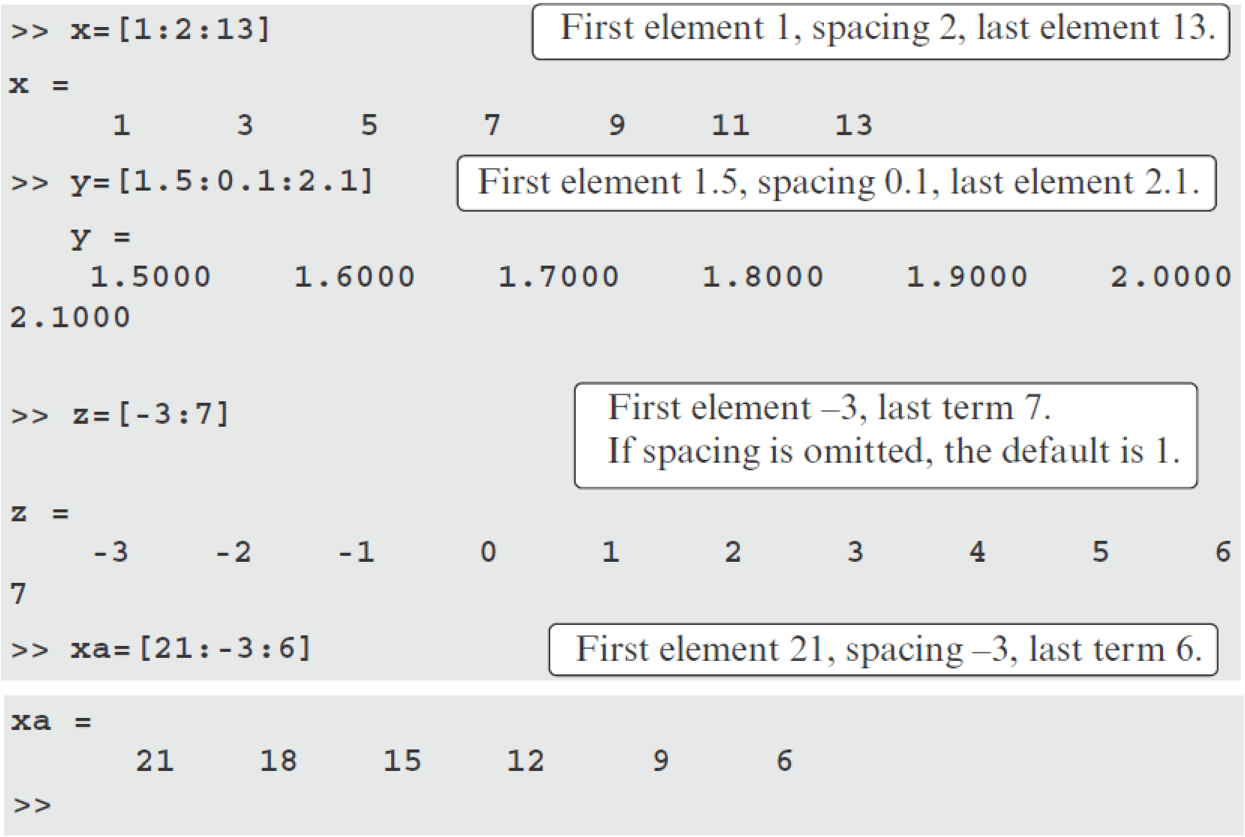


```
variable_name = [m:q:n]
```

or

```
variable_name = m:q:n
```

(The brackets are optional.)



```
>> x=[1:2:13]
```

First element 1, spacing 2, last element 13.

```
x =
      1      3      5      7      9     11     13
>> y=[1.5:0.1:2.1]
```

First element 1.5, spacing 0.1, last element 2.1.

```
   y =
    1.5000    1.6000    1.7000    1.8000    1.9000    2.0000
2.1000


>> z=[-3:7]
```

First element –3, last term 7.
If spacing is omitted, the default is 1.

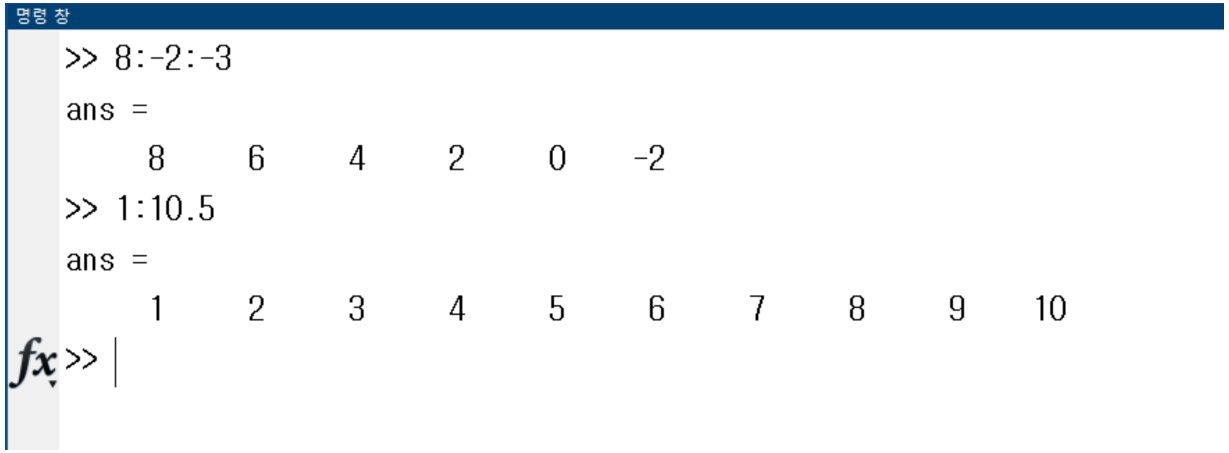
```
z =
    -3     -2     -1      0      1      2      3      4      5      6
7
>> xa=[21:-3:6]
```

First element 21, spacing –3, last term 6.

```
xa =
      21     18     15     12      9      6
>>
```

- If the numbers m, q, and n are such that the value of n cannot be obtained by adding q’s to m, then (for positive n) the last element in the vector will be the last number that does not exceed n.

- If only two numbers (the first and the last terms) are typed (the spacing is omitted), then the default for the spacing is 1.

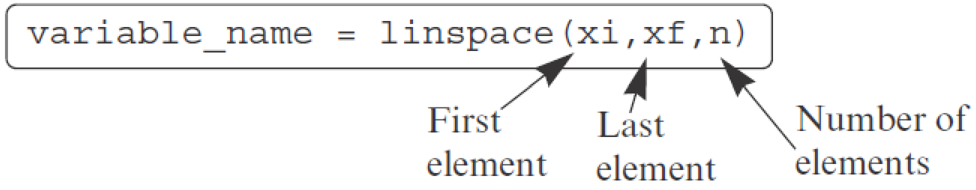


```
>> 8:-2:-3
ans =
     8     6     4     2     0    -2
>> 1:10.5
ans =
     1     2     3     4     5     6     7     8     9    10
>>
```

- Creating a vector with linear (equal) spacing by specifying the first and last terms, and the number of terms:

  - A vector with *n* elements that are linearly (equally) spaced in which the first element is *xi* and the last element is *xf* can be created by typing the `linspace` command (MATLAB determines the correct spacing).

  - When the number of elements is omitted, the default is 100.

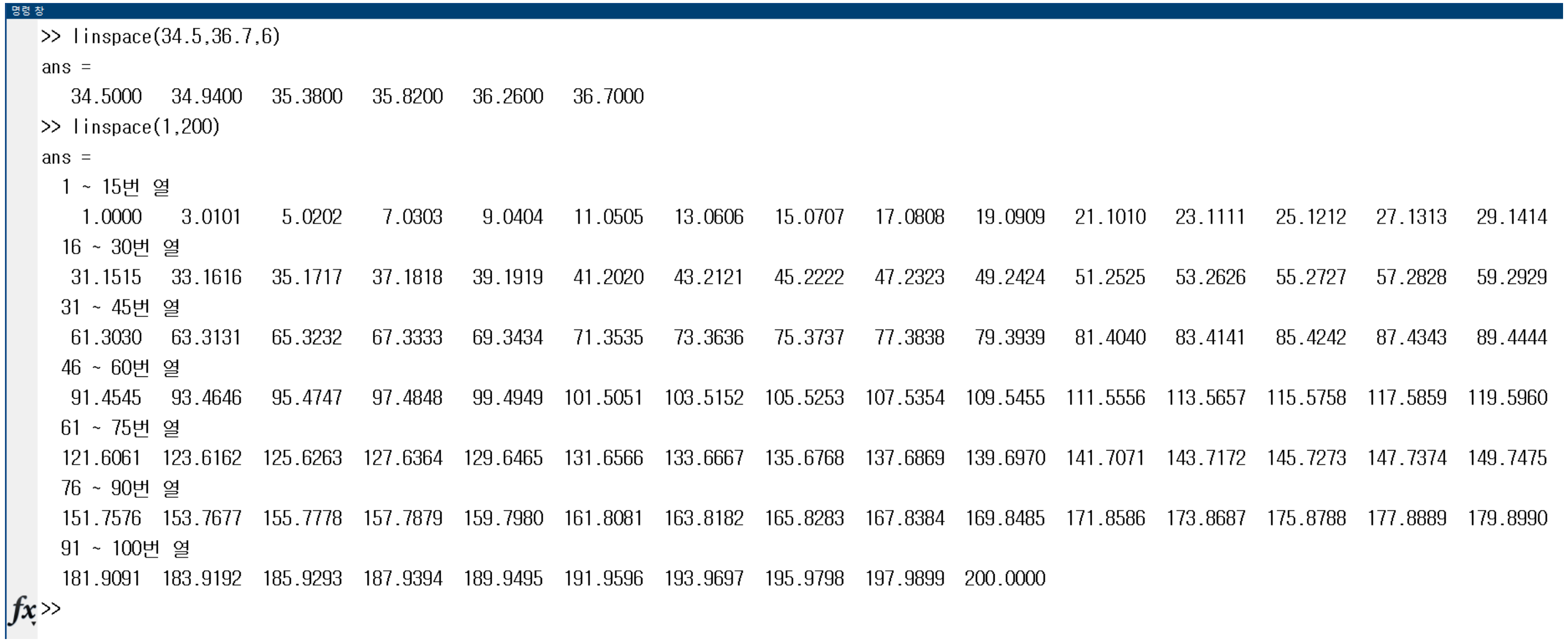


```
variable_name = linspace(xi,xf,n)
```

First element

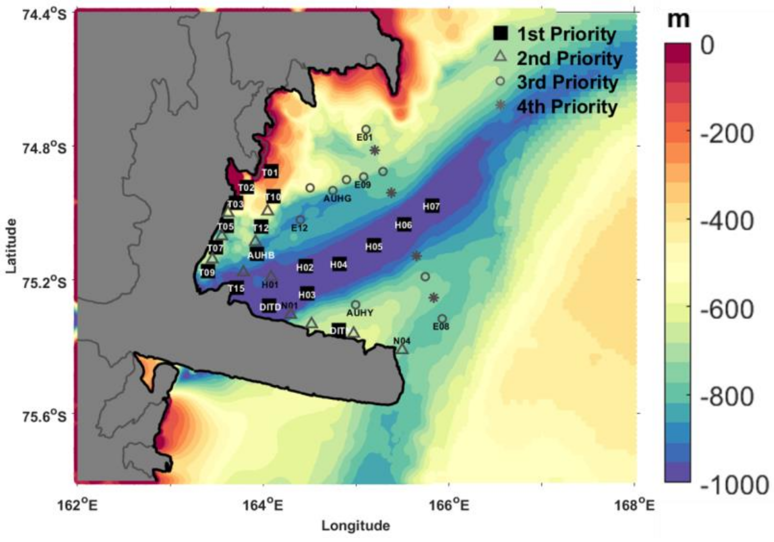
Last element

Number of elements



```
>> linspace(34.5,36.7,6)
ans =
   34.5000   34.9400   35.3800   35.8200   36.2600   36.7000
>> linspace(1,200)
ans =
  1 ~ 15번 열
    1.0000    3.0101    5.0202    7.0303    9.0404   11.0505   13.0606   15.0707   17.0808   19.0909   21.1010   23.1111   25.1212   27.1313   29.1414
  16 ~ 30번 열
   31.1515   33.1616   35.1717   37.1818   39.1919   41.2020   43.2121   45.2222   47.2323   49.2424   51.2525   53.2626   55.2727   57.2828   59.2929
  31 ~ 45번 열
   61.3030   63.3131   65.3232   67.3333   69.3434   71.3535   73.3636   75.3737   77.3838   79.3939   81.4040   83.4141   85.4242   87.4343   89.4444
  46 ~ 60번 열
   91.4545   93.4646   95.4747   97.4848   99.4949  101.5051  103.5152  105.5253  107.5354  109.5455  111.5556  113.5657  115.5758  117.5859  119.5960
  61 ~ 75번 열
  121.6061  123.6162  125.6263  127.6364  129.6465  131.6566  133.6667  135.6768  137.6869  139.6970  141.7071  143.7172  145.7273  147.7374  149.7475
  76 ~ 90번 열
  151.7576  153.7677  155.7778  157.7879  159.7980  161.8081  163.8182  165.8283  167.8384  169.8485  171.8586  173.8687  175.8788  177.8889  179.8990
  91 ~ 100번 열
  181.9091  183.9192  185.9293  187.9394  189.9495  191.9596  193.9697  195.9798  197.9899  200.0000
fx >>
```

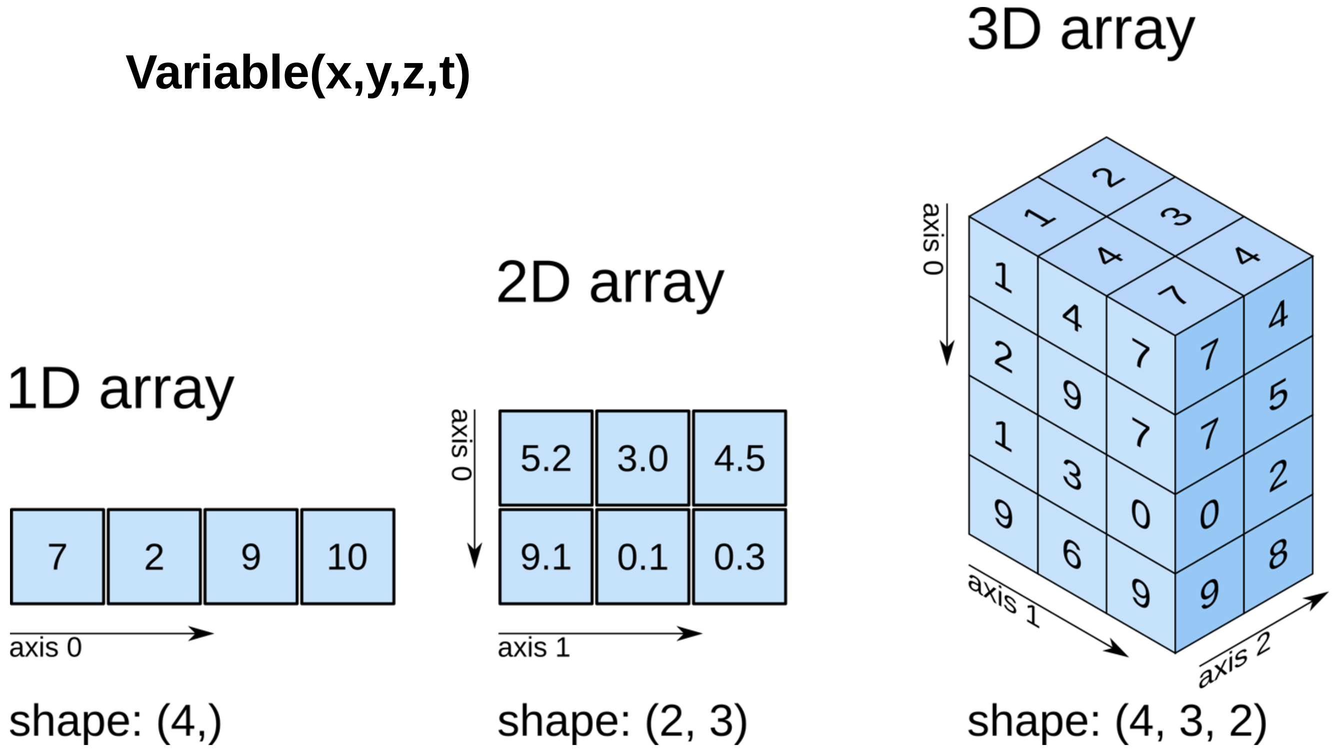
▼ Set-up observation line



2.2 Creating a two-dimensional array (matrix)

- A two-dimensional array, also called a matrix, has numbers in rows and columns.
- Matrices can be used to store information like the arrangement in a table.
- Matrices play an important role in linear algebra and are used in science and engineering to describe many physical quantities.

Variable(x,y,z,t)



Square matrix

$$\begin{array}{ccc} 7 & 4 & 9 \\ 3 & 8 & 1 \\ 6 & 5 & 3 \end{array} \qquad 3 \times 3 \text{ matrix}$$

General matrix

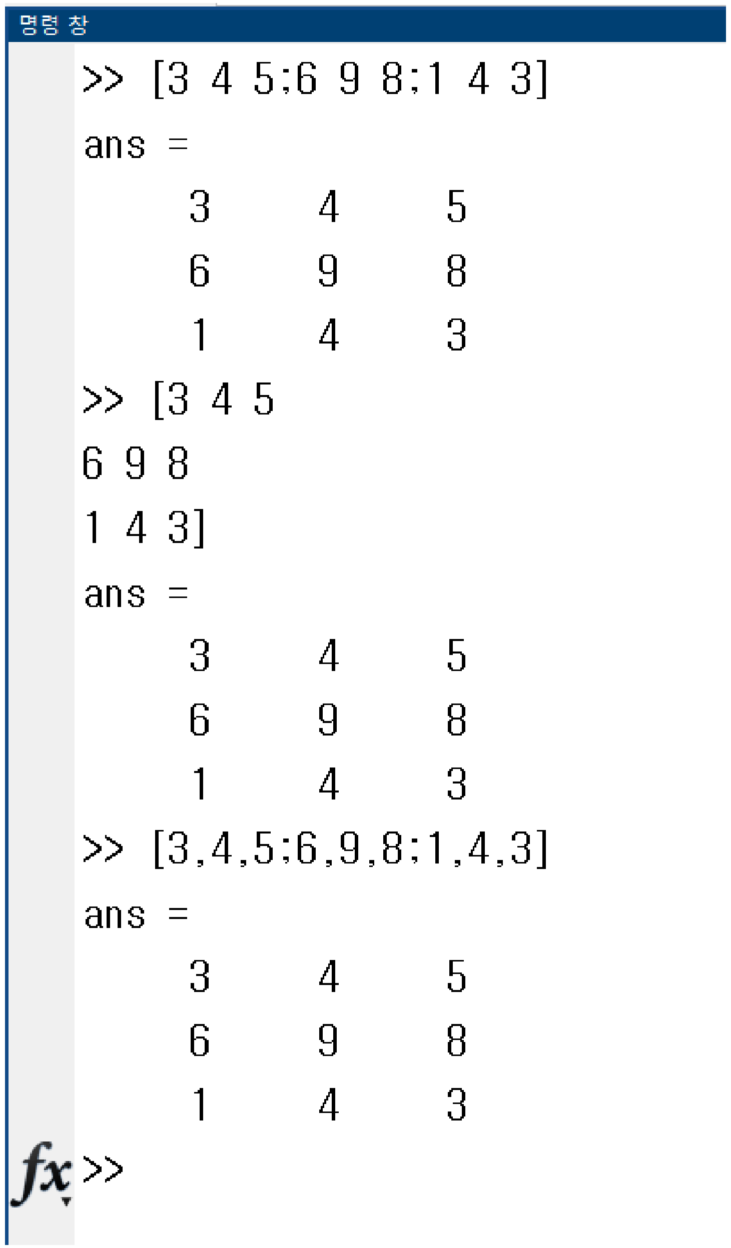
$$\begin{array}{cccccc} 31 & 26 & 14 & 18 & 5 & 30 \\ 3 & 51 & 20 & 11 & 43 & 65 \\ 28 & 6 & 15 & 61 & 34 & 22 \\ 14 & 58 & 6 & 36 & 93 & 7 \end{array} \qquad 4 \times 6 \text{ matrix}$$

- A matrix has *m* rows and *n* columns, and *m* by *n* is called the size of the matrix.

- A matrix is created by assigning the elements of the matrix to a variable. This is done by typing the elements, row by row, inside square brackets [ ].

- First type the left bracket [then type the first row, separating the elements with spaces or commas. To type the next row type a semicolon or press **Enter**. Type the right bracket] at the end of the last row.

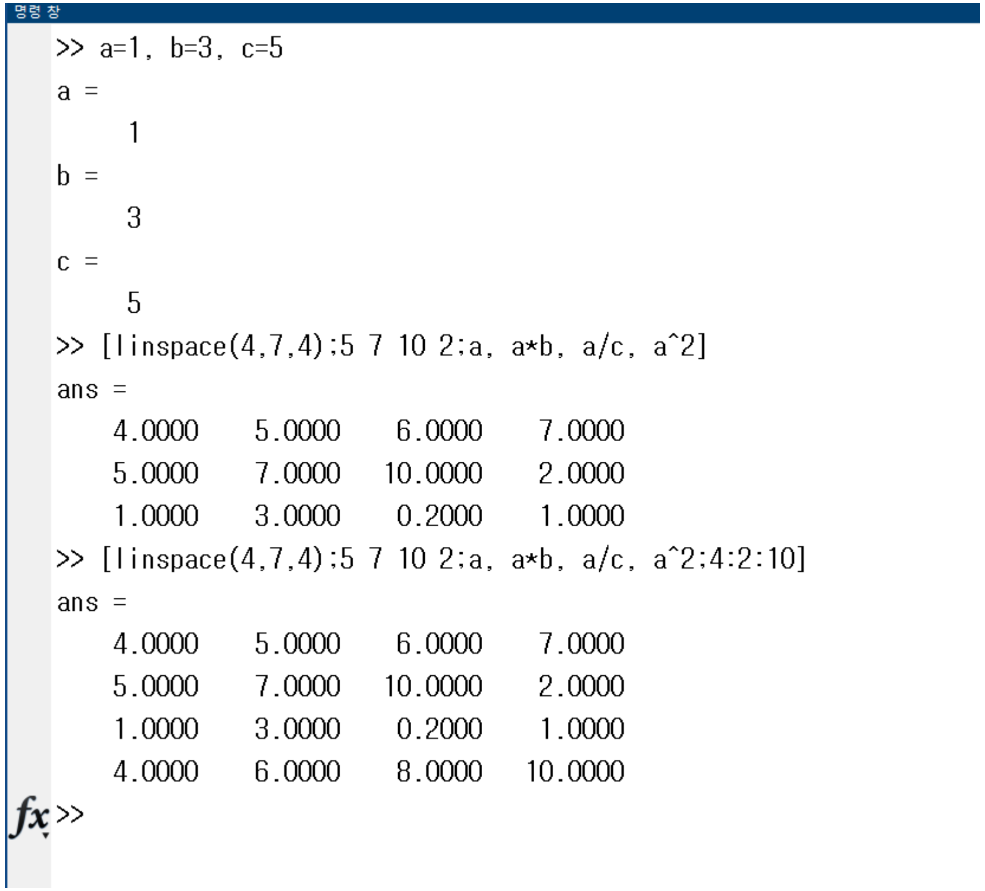
```
variable_name=[1st row elements; 2nd row elements; 3rd
               row elements; ... ; last row elements]
```



```
>> [3 4 5;6 9 8;1 4 3]
ans =
     3     4     5
     6     9     8
     1     4     3
>> [3 4 5
6 9 8
1 4 3]
ans =
     3     4     5
     6     9     8
     1     4     3
>> [3,4,5;6,9,8;1,4,3]
ans =
     3     4     5
     6     9     8
     1     4     3
>>
```

- The elements that are entered can be numbers or mathematical expressions that may include numbers, predefined variables, and functions.

- ***All the rows must have the same number of elements.***

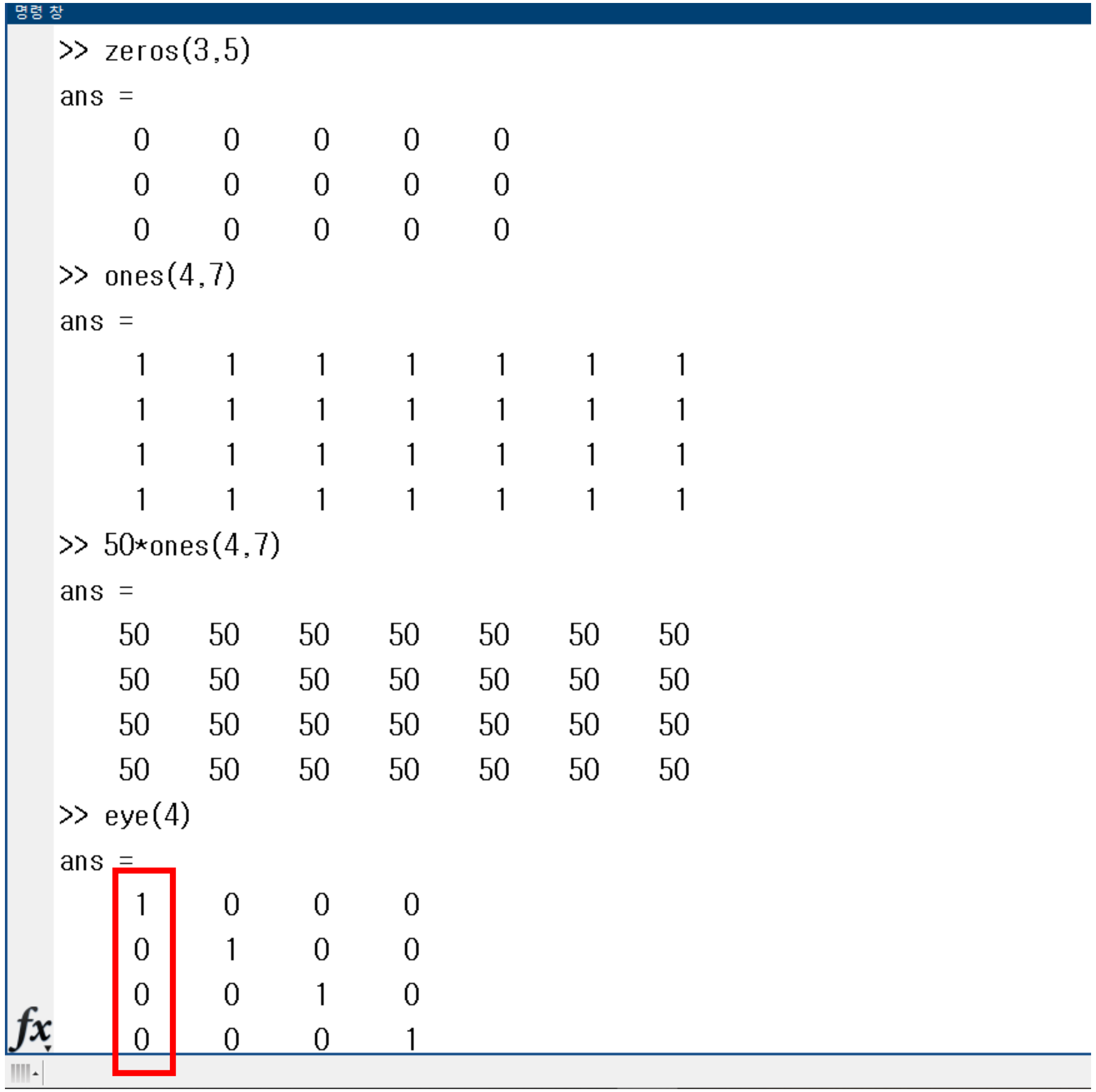


```
>> a=1, b=3, c=5
a =
     1
b =
     3
c =
     5
>> [linspace(4,7,4);5 7 10 2;a, a*b, a/c, a^2]
ans =
    4.0000    5.0000    6.0000    7.0000
    5.0000    7.0000   10.0000    2.0000
    1.0000    3.0000    0.2000    1.0000
>> [linspace(4,7,4);5 7 10 2;a, a*b, a/c, a^2;4:2:10]
ans =
    4.0000    5.0000    6.0000    7.0000
    5.0000    7.0000   10.0000    2.0000
    1.0000    3.0000    0.2000    1.0000
    4.0000    6.0000    8.0000   10.0000
>>
```

2.2.1 The zeros, ones and, eye Commands

- The `zeros(m,n)`, `ones(m,n)`, and `eye(n)` commands can be used to create matrices that have elements with special values.

- The `eye(n)` command creates a square matrix with *n* rows and *n* columns in which the diagonal elements are equal to 1 and the rest of the elements are 0. This matrix is called the identity matrix.



```
>> zeros(3,5)
ans =
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
>> ones(4,7)
ans =
     1     1     1     1     1     1     1
     1     1     1     1     1     1     1
     1     1     1     1     1     1     1
     1     1     1     1     1     1     1
>> 50*ones(4,7)
ans =
    50    50    50    50    50    50    50
    50    50    50    50    50    50    50
    50    50    50    50    50    50    50
    50    50    50    50    50    50    50
>> eye(4)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

eye(4,1)

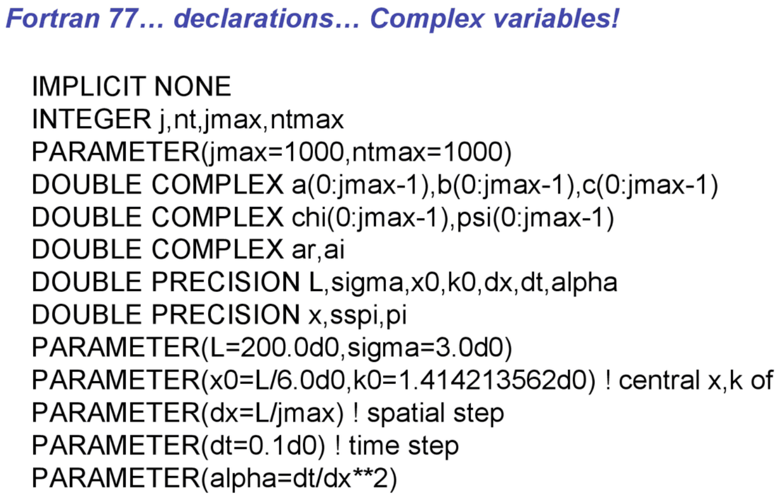
2.3 Notes about variables in MATLAB

- All variables in MATLAB are arrays. A scalar is an array with one element, a vector is an array with one row or one column of elements, and a matrix is an array with elements in rows and columns.

- The variable (scalar, vector, or matrix) is defined by the input when the variable is assigned. There is no need to define the size of the array (single element for a scalar, a row or a column of elements for a vector, or a two-dimensional array of elements for a matrix) before the elements are assigned.

- Once a variable exists—as a scalar, vector, or matrix—it can be changed to any other size, or type, of variable. For example, a scalar can be changed to a vector or a matrix; a vector can be changed to a scalar, a vector of different length, or a matrix; and a matrix can be changed to have a different size, or be reduced to a vector or a scalar. These changes are made by adding or deleting elements.

Example of declarations @ Fortran ▶



Fortran 77… declarations… Complex variables!

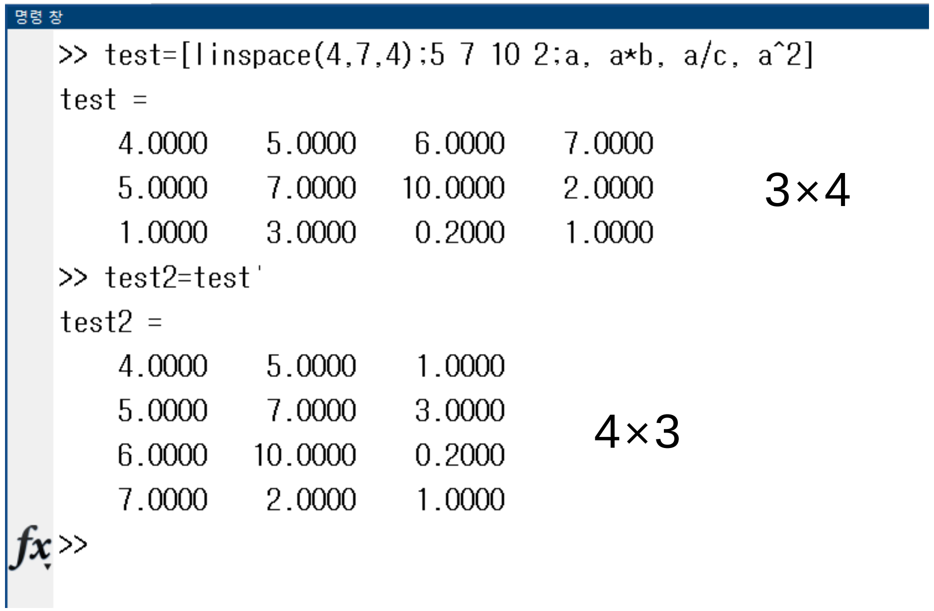
```
IMPLICIT NONE
INTEGER j,nt,jmax,ntmax
PARAMETER(jmax=1000,ntmax=1000)
DOUBLE COMPLEX a(0:jmax-1),b(0:jmax-1),c(0:jmax-1)
DOUBLE COMPLEX chi(0:jmax-1),psi(0:jmax-1)
DOUBLE COMPLEX ar,ai
DOUBLE PRECISION L,sigma,x0,k0,dx,dt,alpha
DOUBLE PRECISION x,sspi,pi
PARAMETER(L=200.0d0,sigma=3.0d0)
PARAMETER(x0=L/6.0d0,k0=1.414213562d0) ! central x,k of
PARAMETER(dx=L/jmax) ! spatial step
PARAMETER(dt=0.1d0) ! time step
PARAMETER(alpha=dt/dx**2)
```

2.4 The transpose operator

- The transpose operator, when applied to a vector, switches a row (column) vector to a column (row) vector.

- When applied to a matrix, it switches the rows (columns) to columns (rows).

- The transpose operator is applied by typing a single quote ’ following the variable to be transposed.



```
>> test=[linspace(4,7,4);5 7 10 2;a, a*b, a/c, a^2]
test =
    4.0000    5.0000    6.0000    7.0000
    5.0000    7.0000   10.0000    2.0000
    1.0000    3.0000    0.2000    1.0000
>> test2=test'
test2 =
    4.0000    5.0000    1.0000
    5.0000    7.0000    3.0000
    6.0000   10.0000    0.2000
    7.0000    2.0000    1.0000
>>
```

3×4

4×3

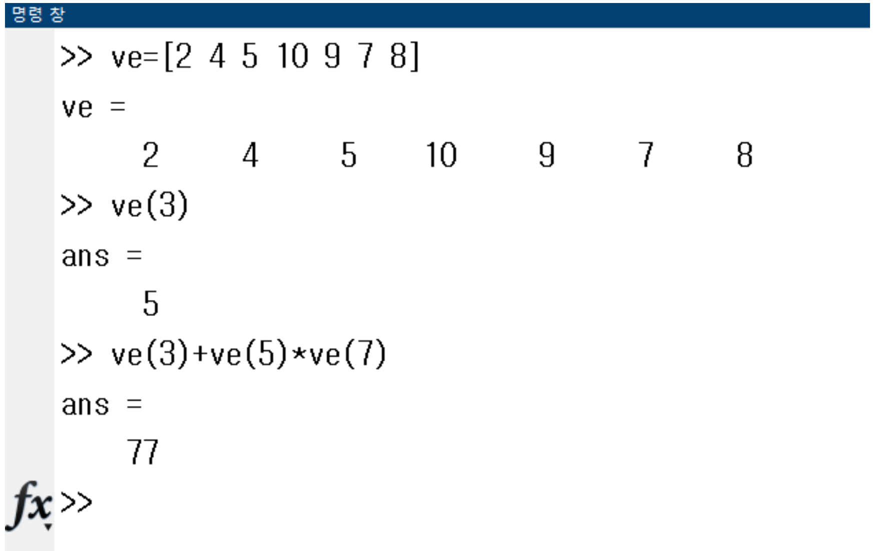
2.5 Array addressing

- Elements in an array (either vector or matrix) can be addressed individually or in subgroups.

- This is useful when there is a need to redefine only some of the elements, when specific elements are to be used in calculations, or when a subgroup of the elements is used to define a new variable.

2.5.1 Vector

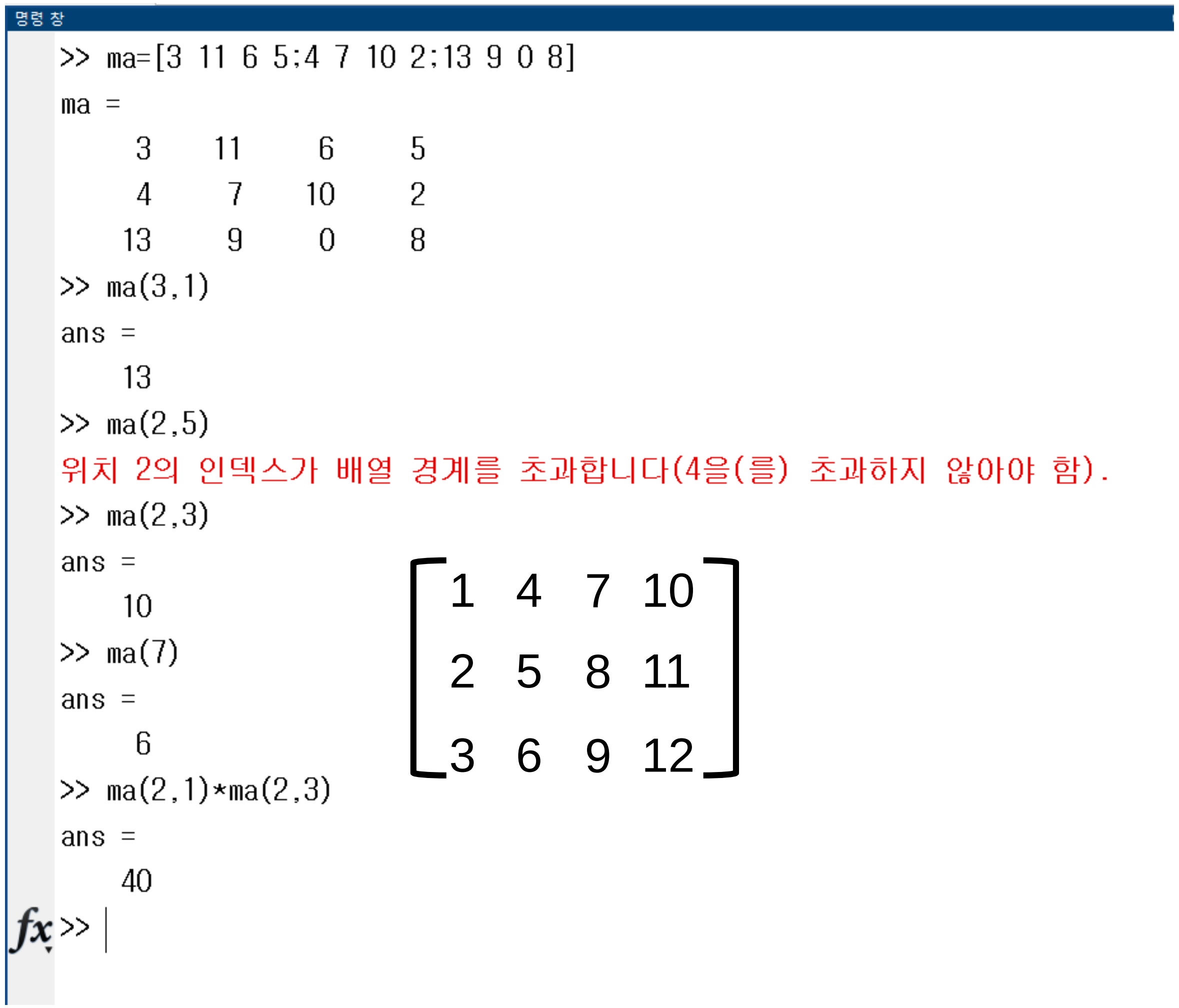
- The address of an element in a vector is its position in the row (or column).



```
>> ve=[2 4 5 10 9 7 8]
ve =
     2     4     5    10     9     7     8
>> ve(3)
ans =
     5
>> ve(3)+ve(5)*ve(7)
ans =
    77
>>
```

2.5.2 Matrix

- The address of an element in a matrix is its position, defined by the row number and the column number where it is located. For a matrix assigned to a variable *ma*, *ma*(*k*,*p*) refers to the element in row *k* and column *p*.



```
>> ma=[3 11 6 5;4 7 10 2;13 9 0 8]
ma =
     3    11     6     5
     4     7    10     2
    13     9     0     8
>> ma(3,1)
ans =
    13
>> ma(2,5)
위치 2의 인덱스가 배열 경계를 초과합니다(4을(를) 초과하지 않아야 함).
>> ma(2,3)
ans =
    10
>> ma(7)
ans =
     6
>> ma(2,1)*ma(2,3)
ans =
    40
>>
```

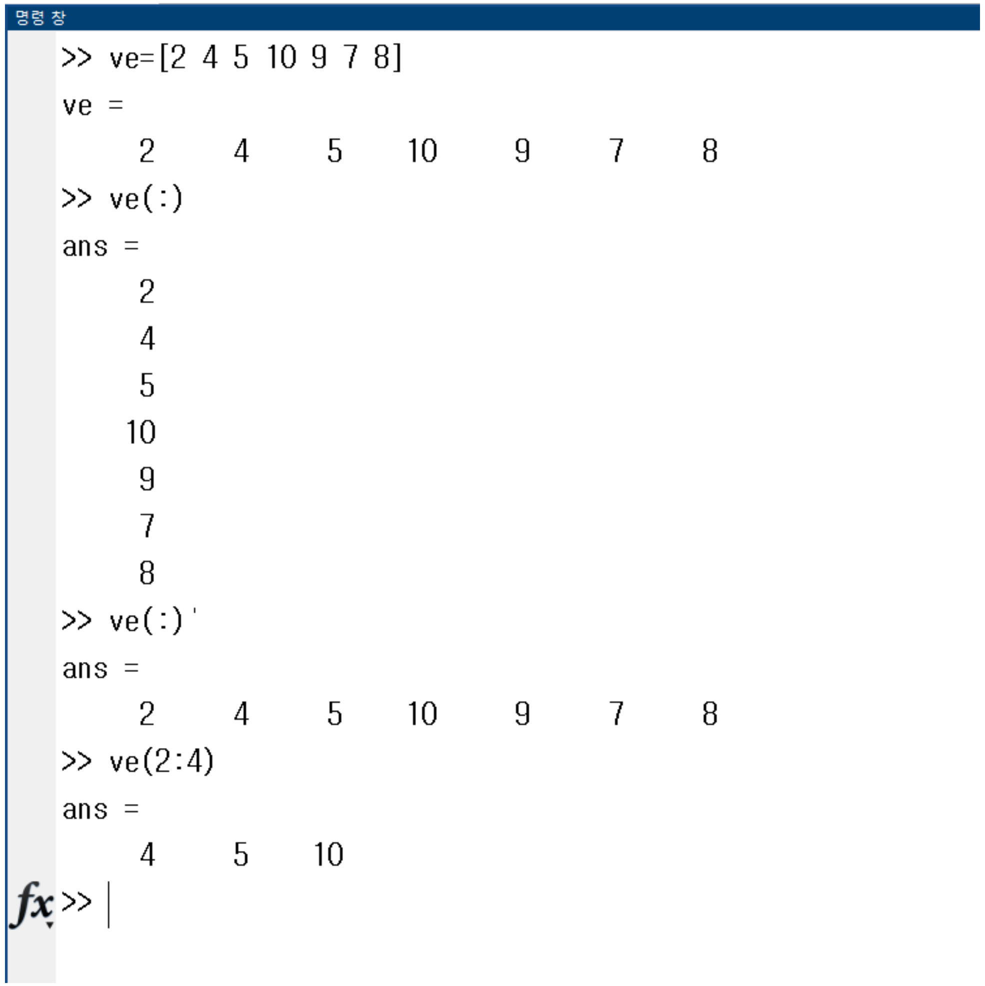
$$\begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

2.6 Using a Colon: In addressing arrays

- A colon can be used to address a range of elements in a vector or a matrix.

- **For a vector**

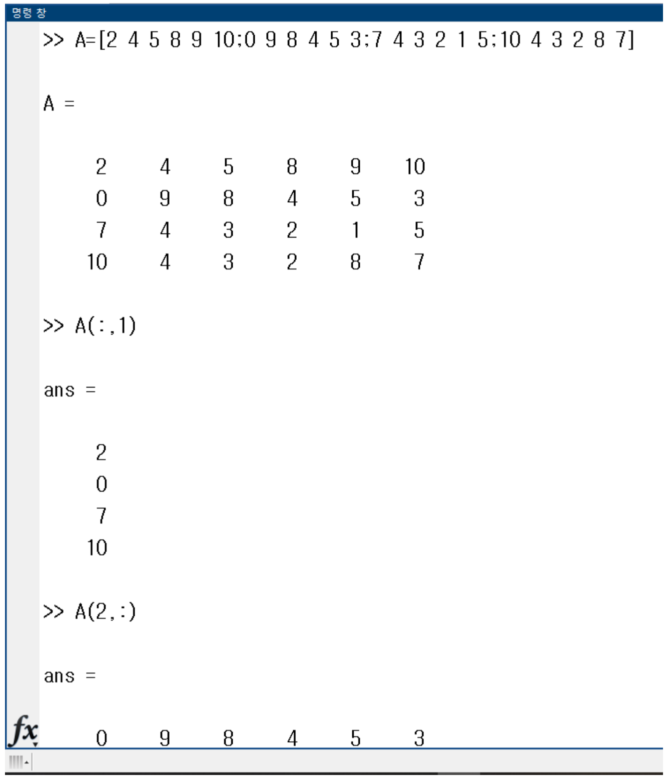
- : refers to all the elements of the vector (either a row or a column vector).
- **m:n** refers to elements m through n of the vector.



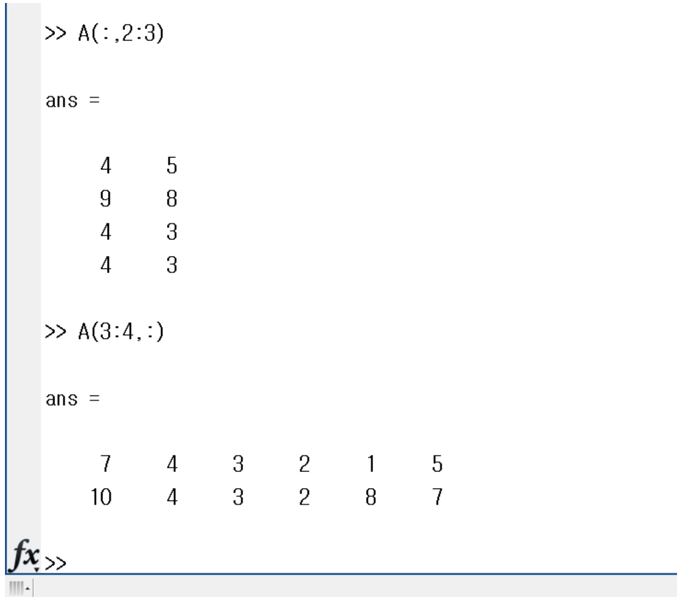
```
>> ve=[2 4 5 10 9 7 8]
ve =
     2     4     5    10     9     7     8
>> ve(:)
ans =
     2
     4
     5
    10
     9
     7
     8
>> ve(:)'
ans =
     2     4     5    10     9     7     8
>> ve(2:4)
ans =
     4     5    10
>> 
```

- **For a matrix**

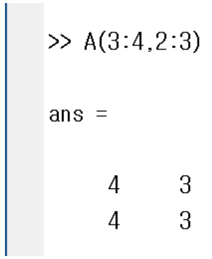
- *A*(:,*n*) refers to the elements in all the rows of column *n* of the matrix *A*.
- *A*(*n*,:) refers to the elements in all the columns of row *n* of the matrix *A*.
- *A*(:,*m*:*n*) refers to the elements in all the rows between columns *m* and *n* of the matrix *A*.
- *A*(*m*:*n*,:) refers to the elements in all the columns between rows *m* and *n* of the matrix *A*.
- A(*m*:*n*,*p*:*q*) refers to the elements in rows *m* through *n* and columns *p* through *q* of the matrix *A*.



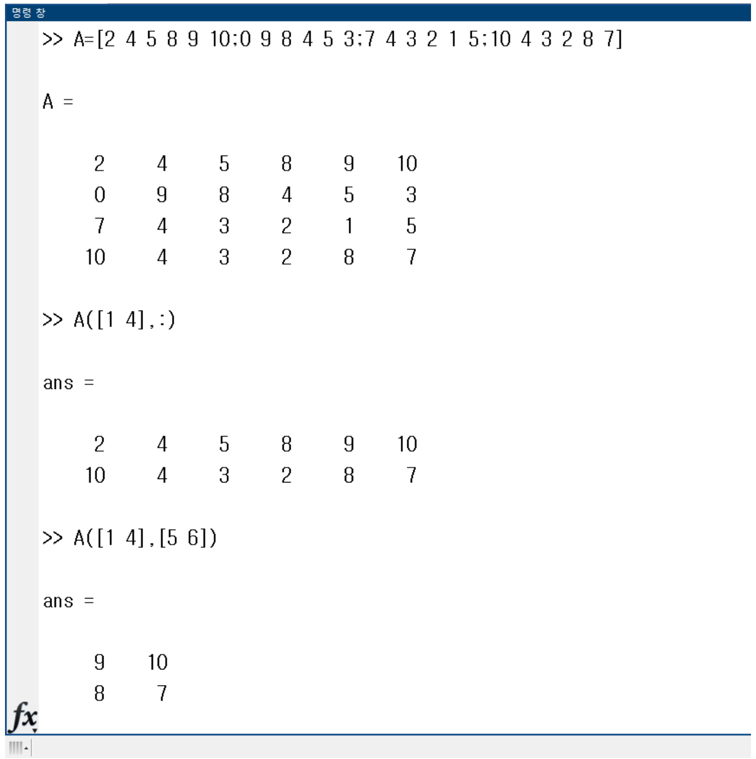
```
>> A=[2 4 5 8 9 10;0 9 8 4 5 3;7 4 3 2 1 5;10 4 3 2 8 7]

A =

     2     4     5     8     9    10
     0     9     8     4     5     3
     7     4     3     2     1     5
    10     4     3     2     8     7

>> A(:,1)

ans =

     2
     0
     7
    10

>> A(2,:)

ans =

     0     9     8     4     5     3
```



```
>> A(:,2:3)

ans =

     4     5
     9     8
     4     3
     4     3

>> A(3:4,:)

ans =

     7     4     3     2     1     5
    10     4     3     2     8     7

>>
```



```
>> A(3:4,2:3)

ans =

     4     3
     4     3
```



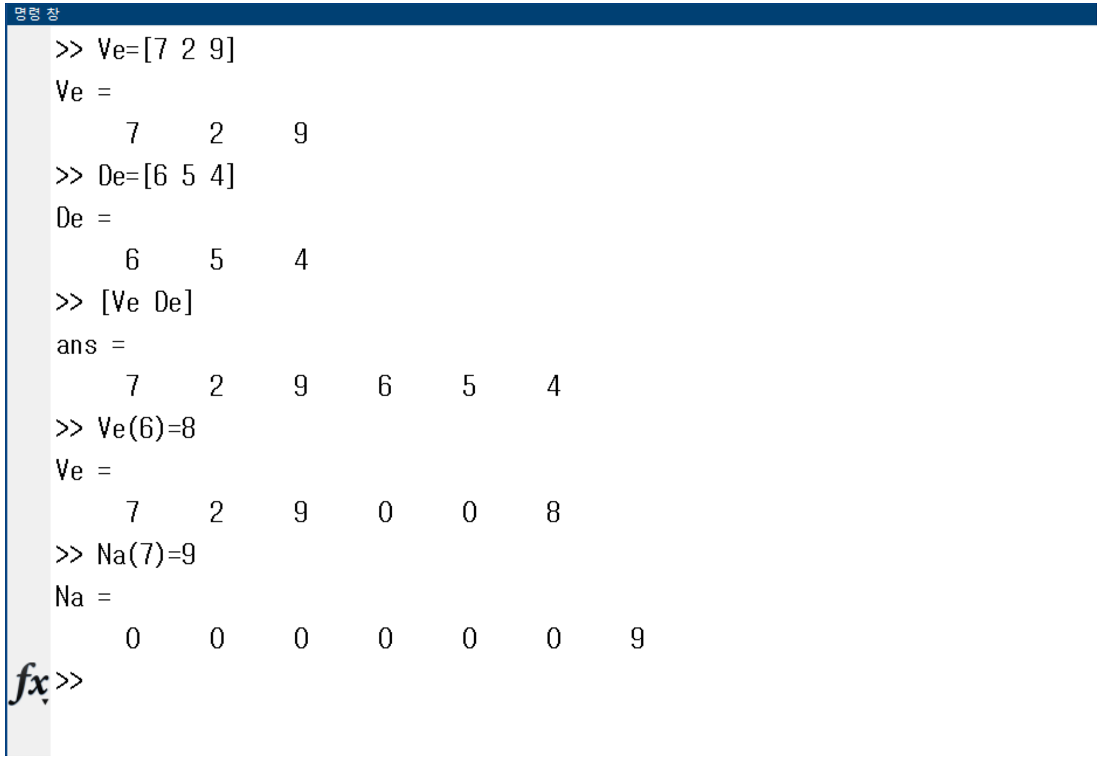
```
>> A=[2 4 5 8 9 10;0 9 8 4 5 3;7 4 3 2 1 5;10 4 3 2 8 7]

A =

     2     4     5     8     9    10
     0     9     8     4     5     3
     7     4     3     2     1     5
    10     4     3     2     8     7

>> A([1 4],:)

ans =

     2     4     5     8     9    10
    10     4     3     2     8     7

>> A([1 4],[5 6])

ans =

     9    10
     8     7
```

2.7 Adding Elements to existing variables

- A variable that exists as a vector, or a matrix, can be changed by adding elements to it (remember that a scalar is a vector with one element).

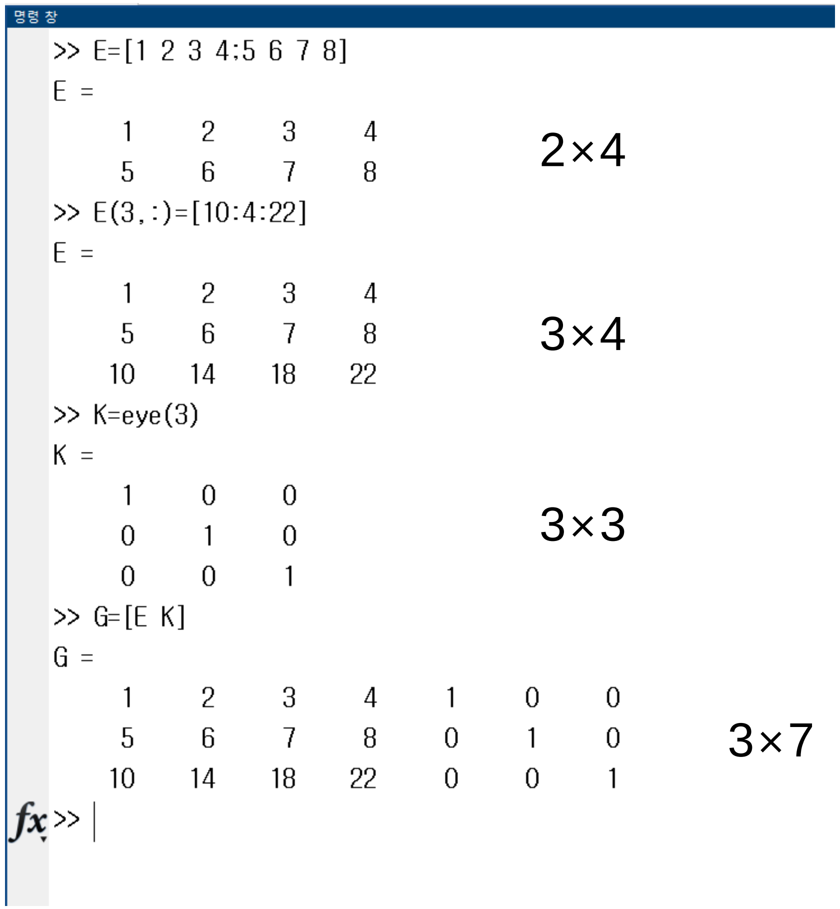
- A vector (a matrix with a single row or column) can be changed to have more elements, or it can be changed to be a two-dimensional matrix. Rows and/or columns can also be added to an existing matrix to obtain a matrix of different size. The addition of elements can be done by simply assigning values to the additional elements, or by appending existing variables.

- **Adding elements to a vector**:



```
>> Ve=[7 2 9]
Ve =
     7     2     9
>> De=[6 5 4]
De =
     6     5     4
>> [Ve De]
ans =
     7     2     9     6     5     4
>> Ve(6)=8
Ve =
     7     2     9     0     0     8
>> Na(7)=9
Na =
     0     0     0     0     0     0     9
>>
```

- **Adding elements to a matrix**: Rows and/or columns can be added to an existing matrix by assigning values to the new rows or columns. This can be done by assigning new values, or by appending existing variables. This must be done carefully since the size of the added rows or columns **must fit the existing matrix**.



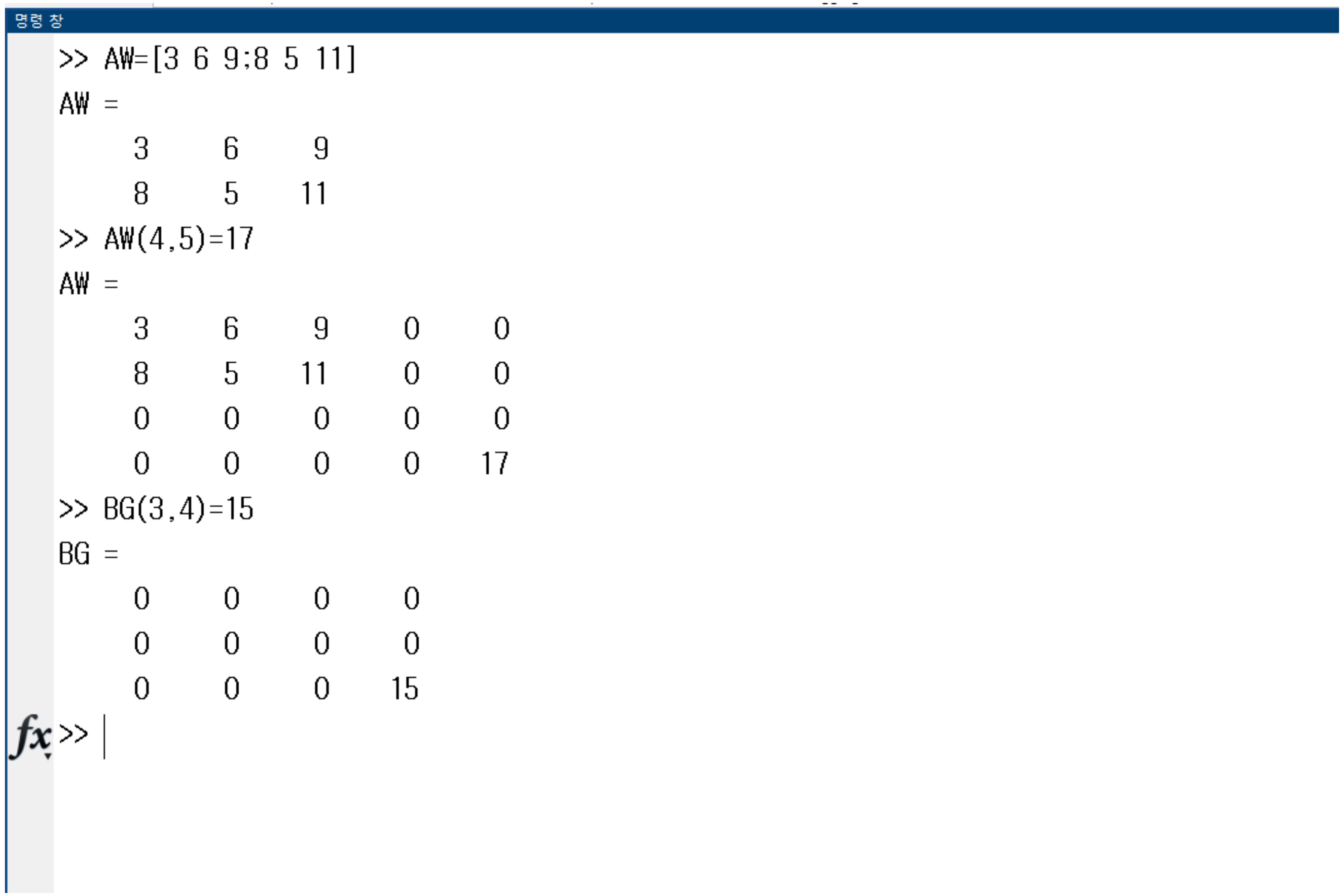
```
>> E=[1 2 3 4:5 6 7 8]
E =
      1      2      3      4
      5      6      7      8
>> E(3,:)=[10:4:22]
E =
      1      2      3      4
      5      6      7      8
     10     14     18     22
>> K=eye(3)
K =
      1      0      0
      0      1      0
      0      0      1
>> G=[E K]
G =
      1      2      3      4      1      0      0
      5      6      7      8      0      1      0
     10     14     18     22      0      0      1
fx >>
```

2×4

3×4

3×3

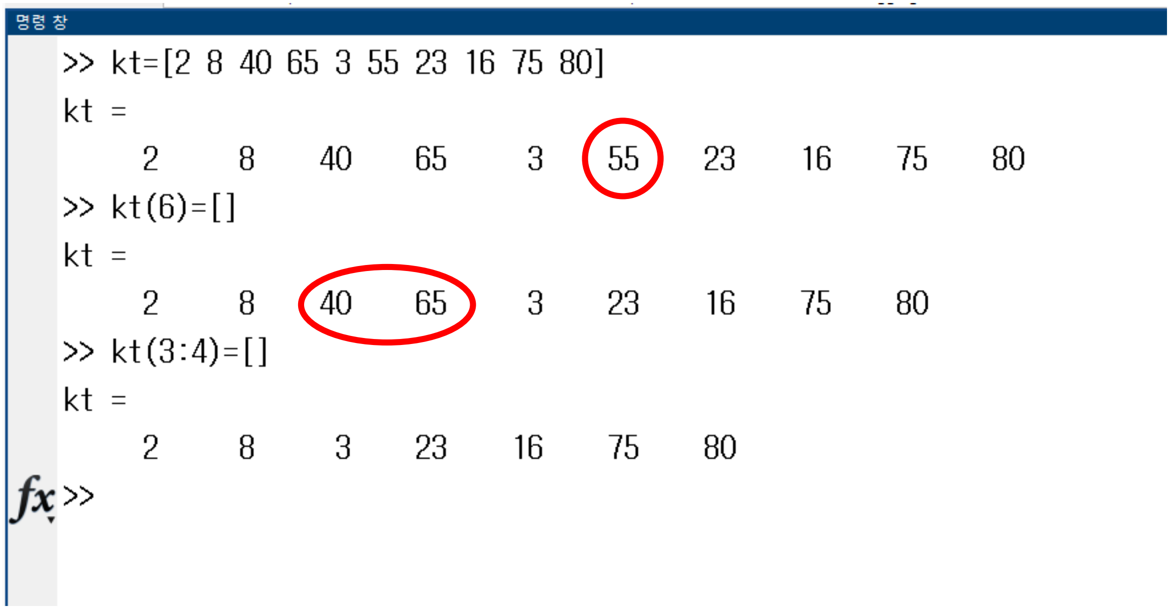
3×7



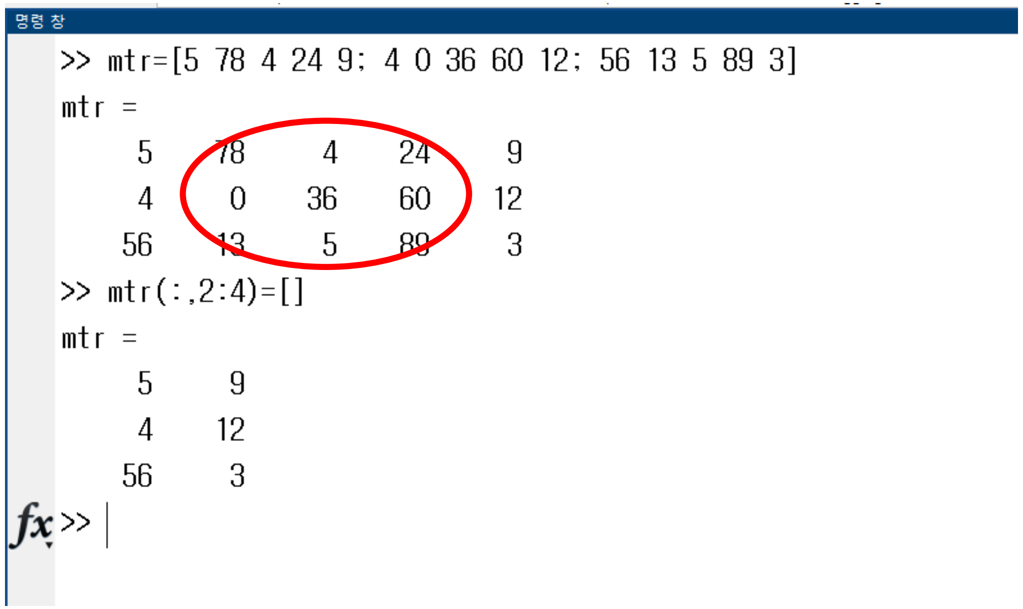
```
>> AW=[3 6 9:8 5 11]
AW =
      3      6      9
      8      5     11
>> AW(4,5)=17
AW =
      3      6      9      0      0
      8      5     11      0      0
      0      0      0      0      0
      0      0      0      0     17
>> BG(3,4)=15
BG =
      0      0      0      0
      0      0      0      0
      0      0      0     15
fx >>
```

2.8 Deleting elements

- An element, or a range of elements, of an existing variable can be deleted by reassigning nothing to these elements. This is done by using square brackets with nothing typed in between them. By deleting elements, a vector can be made shorter and a matrix can be made smaller.



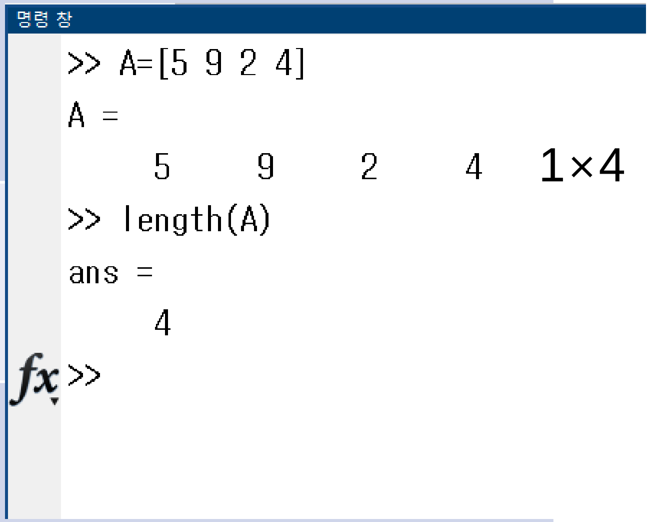
```
>> kt=[2 8 40 65 3 55 23 16 75 80]
kt =
     2     8    40    65     3    55    23    16    75    80
>> kt(6)=[]
kt =
     2     8    40    65     3    23    16    75    80
>> kt(3:4)=[]
kt =
     2     8     3    23    16    75    80
>>
```



```
>> mtr=[5 78 4 24 9; 4 0 36 60 12; 56 13 5 89 3]
mtr =
     5    78     4    24     9
     4     0    36    60    12
    56    13     5    89     3
>> mtr(:,2:4)=[]
mtr =
     5     9
     4    12
    56     3
>>
```

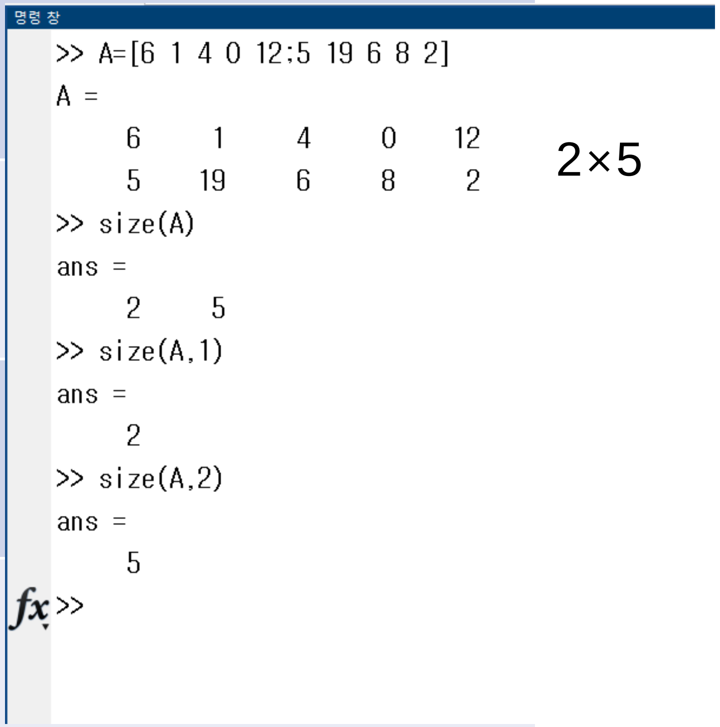
2.9 Built-in functions for handling arrays

- MATLAB has many built-in functions for managing and handling arrays. Some of these are listed below:

Function	Description	Example
length(A)	Returns the number of elements in the vector A.	
size(A)	Returns a row vector [m,n], where m and n are the size m×n of the array A.	
Reshape(A,m,n)	Creates a m by n matrix from the elements of matrix A. The elements are taken column after column. Matrix A must have m times n elements.	
diag(v)	When v is a vector, creates a square matrix with the elements of v in the diagonal.	
diag(A)	When A is a matrix, creates a vector from the diagonal elements of A.	

```
>> A=[5 9 2 4]
A =
     5     9     2     4
>> length(A)
ans =
     4
>>
```

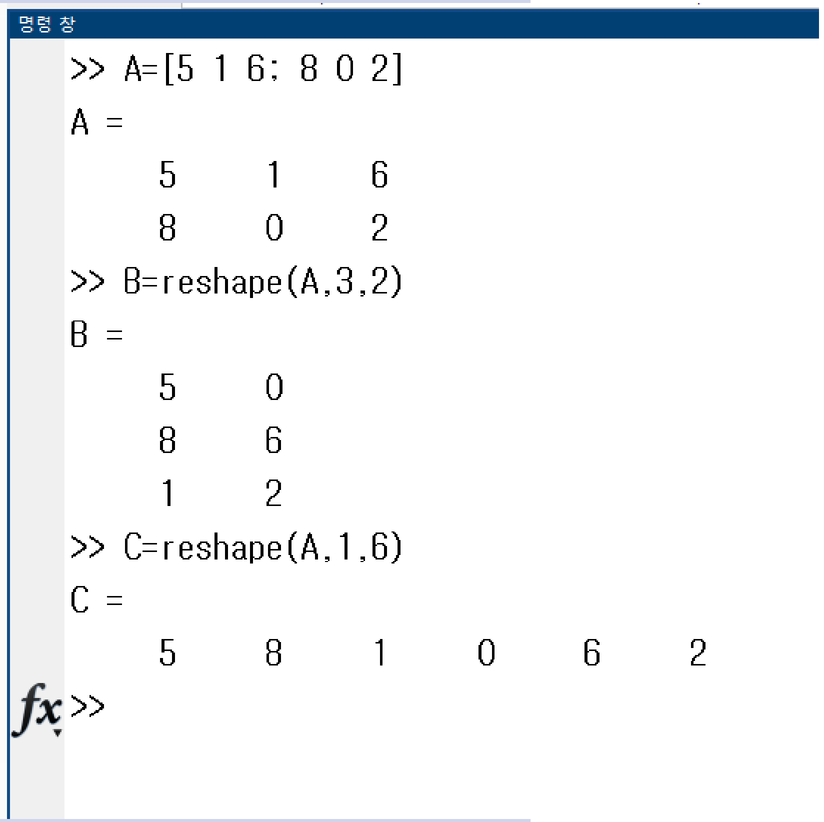
Function	Description	Example
length(A)	Returns the number of elements in the vector A.	
size(A)	Returns a row vector [m,n], where m and n are the size m×n of the array A.	
Reshape(A,m,n)	Creates a m by n matrix from the elements of matrix A. The elements are taken column after column. Matrix A must have m times n elements.	
diag(v)	When v is a vector, creates a square matrix with the elements of v in the diagonal.	
diag(A)	When A is a matrix, creates a vector from the diagonal elements of A.	



```
>> A=[6 1 4 0 12;5 19 6 8 2]
A =
     6     1     4     0    12
     5    19     6     8     2
>> size(A)
ans =
     2     5
>> size(A,1)
ans =
     2
>> size(A,2)
ans =
     5
>>
```

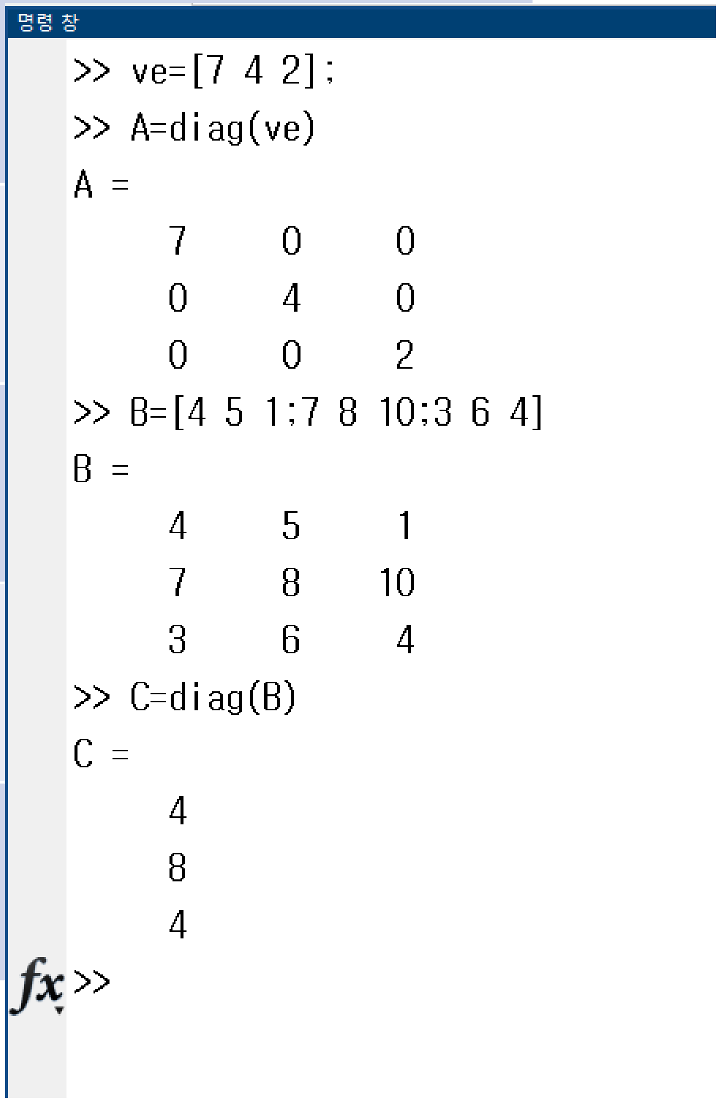
2×5

Function	Description	Example
length(A)	Returns the number of elements in the vector A.	
size(A)	Returns a row vector [m,n], where m and n are the size m×n of the array A.	
Reshape(A,m,n)	Creates a m by n matrix from the elements of matrix A. The elements are taken column after column. Matrix A must have m times n elements.	
diag(v)	When v is a vector, creates a square matrix with the elements of v in the diagonal.	
diag(A)	When A is a matrix, creates a vector from the diagonal elements of A.	



```
>> A=[5 1 6; 8 0 2]
A =
     5     1     6
     8     0     2
>> B=reshape(A,3,2)
B =
     5     0
     8     6
     1     2
>> C=reshape(A,1,6)
C =
     5     8     1     0     6     2
>>
```

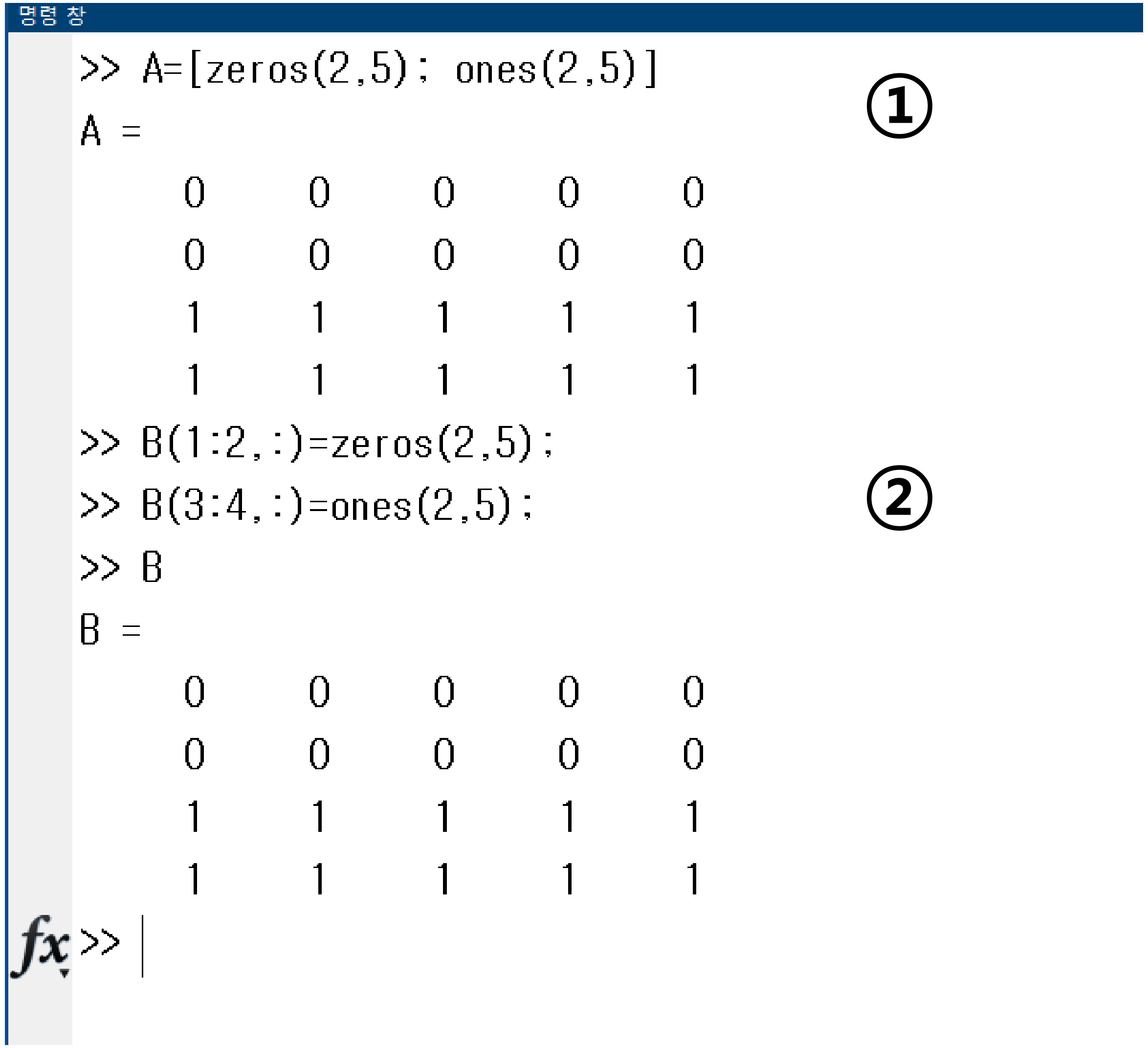
Function	Description	Example
length(A)	Returns the number of elements in the vector A.	
size(A)	Returns a row vector [m,n], where m and n are the size m×n of the array A.	
Reshape(A,m,n)	Creates a m by n matrix from the elements of matrix A. The elements are taken column after column. Matrix A must have m times n elements.	
diag(v)	When v is a vector, creates a square matrix with the elements of v in the diagonal.	
diag(A)	When A is a matrix, creates a vector from the diagonal elements of A.	



```
>> ve=[7 4 2];
>> A=diag(ve)
A =
     7     0     0
     0     4     0
     0     0     2
>> B=[4 5 1;7 8 10;3 6 4]
B =
     4     5     1
     7     8    10
     3     6     4
>> C=diag(B)
C =
     4
     8
     4
>>
```

※ Sample Problem 2-1: Create a matrix

- Using the ones and zeros commands, create a 4×5 matrix in which the first two rows are 0s and the next two rows are 1s.

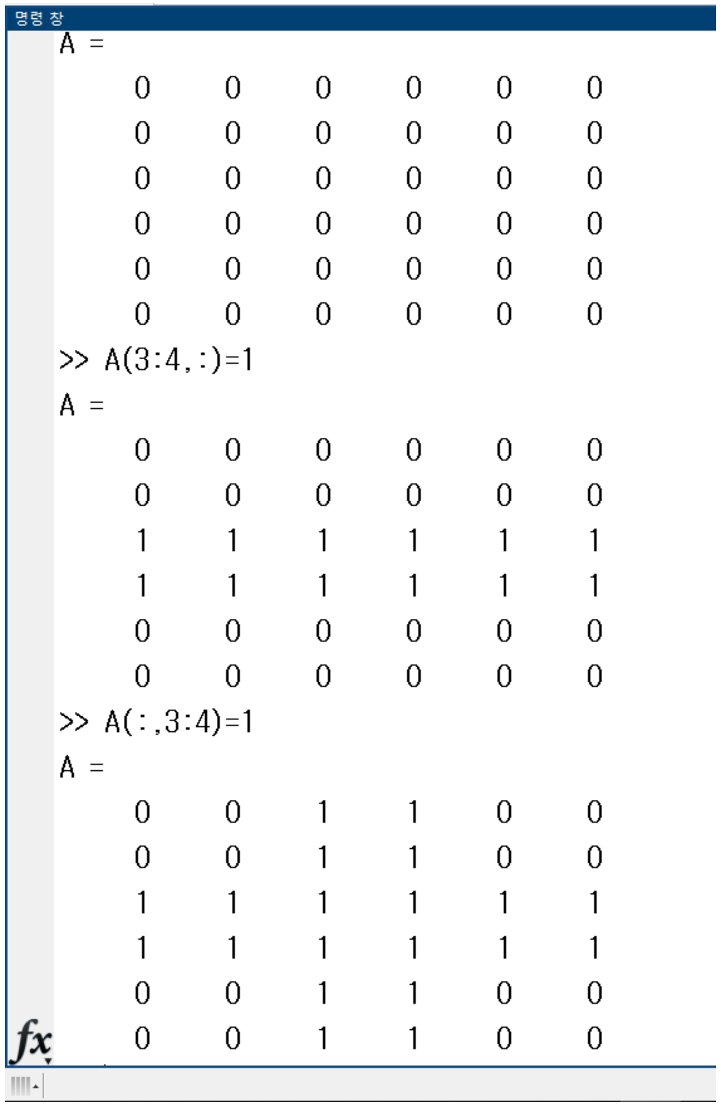


```
>> A=[zeros(2,5); ones(2,5)]
A =
     0     0     0     0     0
     0     0     0     0     0
     1     1     1     1     1
     1     1     1     1     1
>> B(1:2,:)=zeros(2,5);
>> B(3:4,:)=ones(2,5);
>> B
B =
     0     0     0     0     0
     0     0     0     0     0
     1     1     1     1     1
     1     1     1     1     1
>> 
```

※ Sample Problem 2-2: Create a matrix

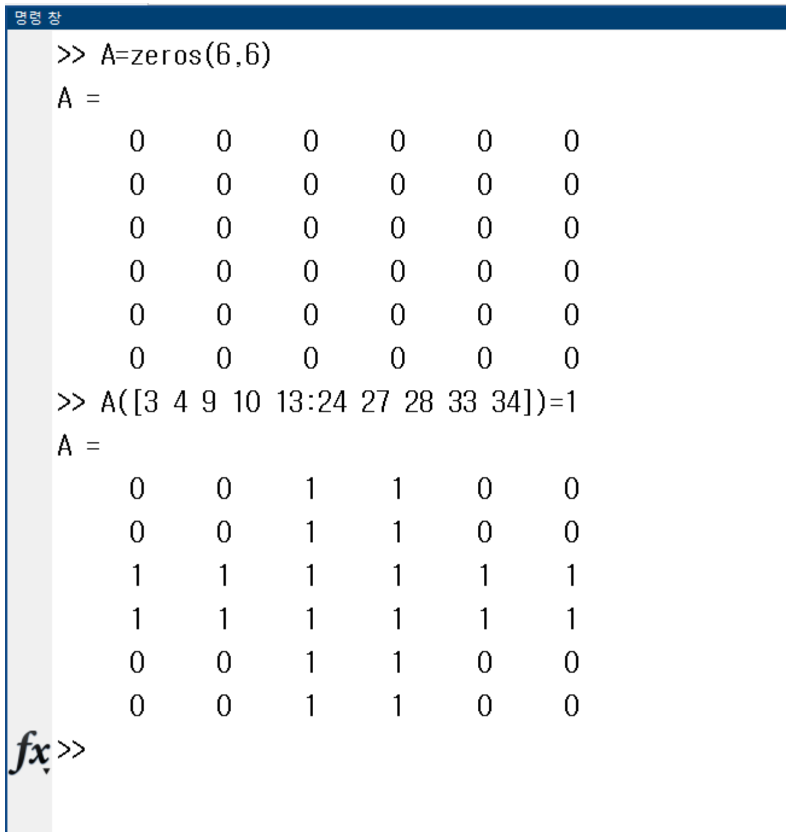
- Create a 6×6 matrix in which the middle two rows and the middle two columns are 1s and the rest of the entries are 0s.

①



```
A =

     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0

>> A(3:4,:)=1

A =

     0     0     0     0     0     0
     0     0     0     0     0     0
     1     1     1     1     1     1
     1     1     1     1     1     1
     0     0     0     0     0     0
     0     0     0     0     0     0

>> A(:,3:4)=1

A =

     0     0     1     1     0     0
     0     0     1     1     0     0
     1     1     1     1     1     1
     1     1     1     1     1     1
     0     0     1     1     0     0
     0     0     1     1     0     0
```

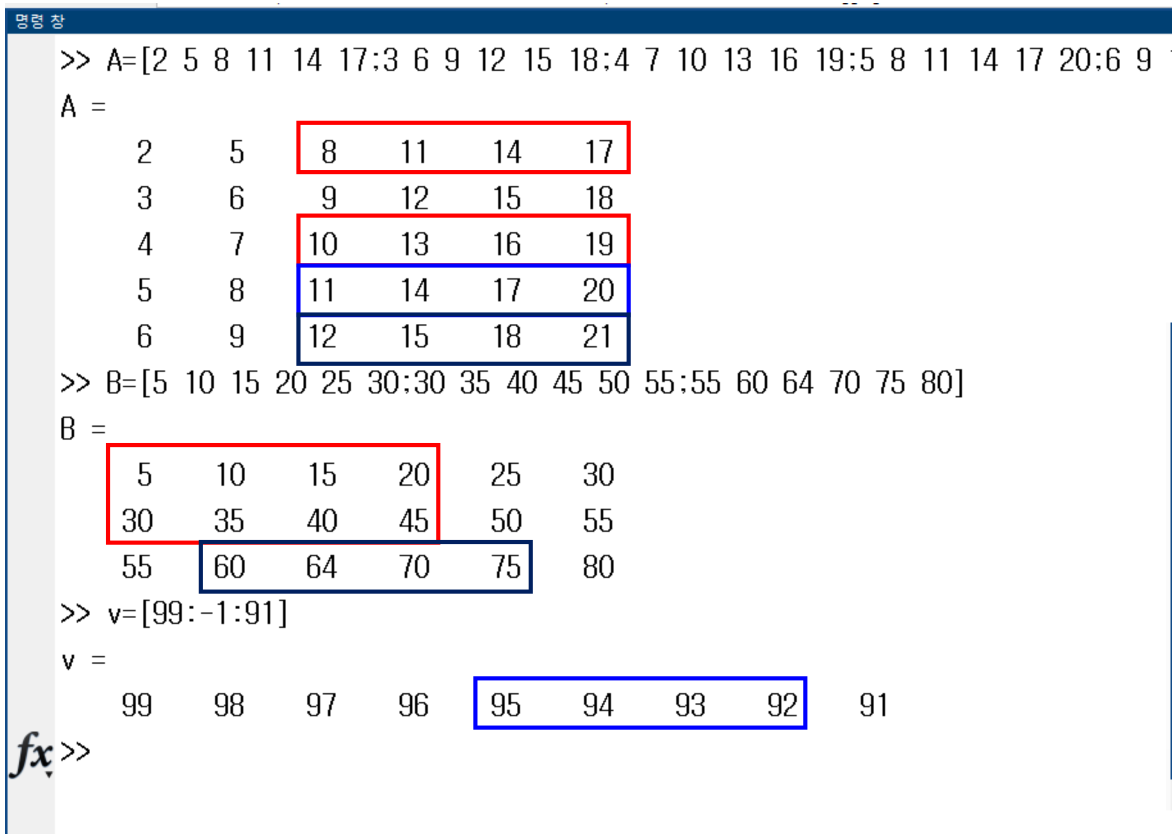
②



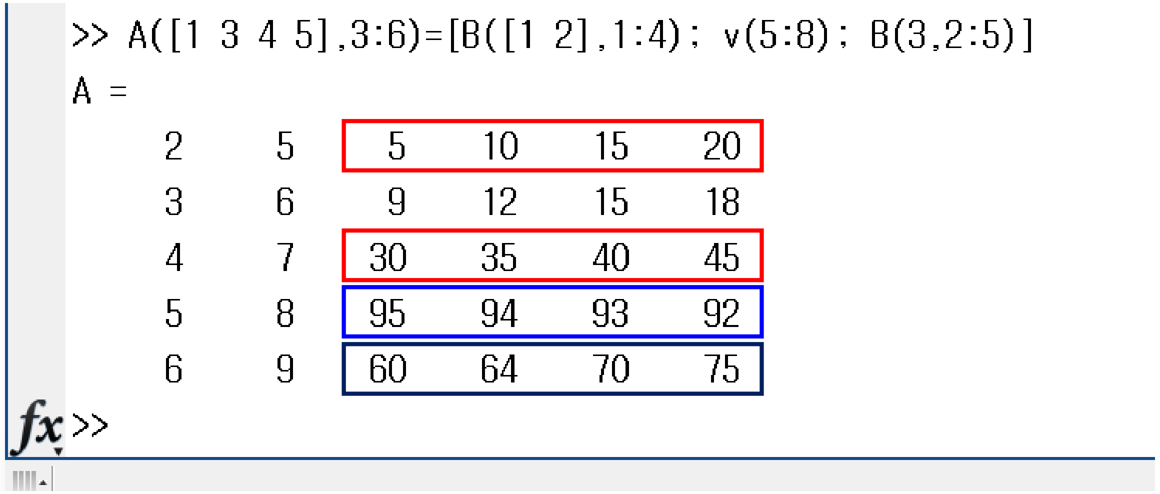
```
>> A=zeros(6,6)

A =

     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0

>> A([3 4 9 10 13:24 27 28 33 34])=1

A =

     0     0     1     1     0     0
     0     0     1     1     0     0
     1     1     1     1     1     1
     1     1     1     1     1     1
     0     0     1     1     0     0
     0     0     1     1     0     0

>>
```

※ Sample Problem 2-3: matrix manipulation

- Create the three arrays in the Command Window, and then, by writing one command, replace the last four columns of the first and third rows of *A* with the first four columns of the first two rows of *B*, the last four columns of the fourth row of *A* with the elements 5 through 8 of *v*, and the last four columns of the fifth row of *A* with columns 3(wrong!!; 2) through 5 of the third row of *B*.



```
>> A=[2 5 8 11 14 17;3 6 9 12 15 18;4 7 10 13 16 19;5 8 11 14 17 20;6 9 12 15 18 21]
A =
     2     5     8    11    14    17
     3     6     9    12    15    18
     4     7    10    13    16    19
     5     8    11    14    17    20
     6     9    12    15    18    21
>> B=[5 10 15 20 25 30;30 35 40 45 50 55;55 60 64 70 75 80]
B =
     5    10    15    20    25    30
    30    35    40    45    50    55
    55    60    64    70    75    80
>> v=[99:-1:91]
v =
    99    98    97    96    95    94    93    92    91
fx >>
```



```
>> A([1 3 4 5],3:6)=[B([1 2],1:4); v(5:8); B(3,2:5)]
A =
     2     5     5    10    15    20
     3     6     9    12    15    18
     4     7    30    35    40    45
     5     8    95    94    93    92
     6     9    60    64    70    75
fx >>
```

2.10 Strings and Strings as Variables

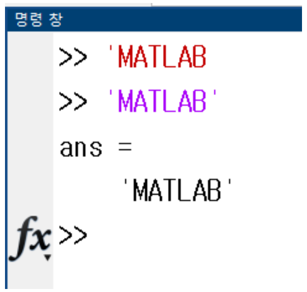
- A string is an array of characters. It is created by typing the characters within single quotes.

- Strings can include letters, digits, other symbols, and spaces. Examples of strings: 'ad ef ', '3%fr2', '{edcba:21!', 'MATLAB’.

- A string that contains a single quote is created by typing two single quotes within the string.

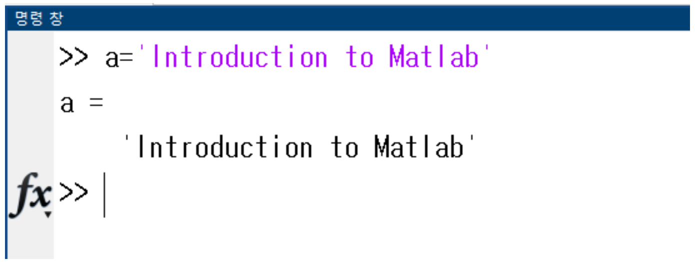
- When a string is being typed in, the color of the text on the screen changes to maroon when the first single quote is typed. When the single quote at the end of the string is typed, the color of the string changes to purple.

- When strings are being used in formatting plots (labels to axes, title, and text notes), characters within the string can be formatted to have a specified font, size, position (uppercase, lowercase), color, etc.



```
>> 'MATLAB
>> 'MATLAB'
ans =
    'MATLAB'
fx >>
```

- Strings can also be assigned to variables by simply typing the string on the right side of the assignment operator.



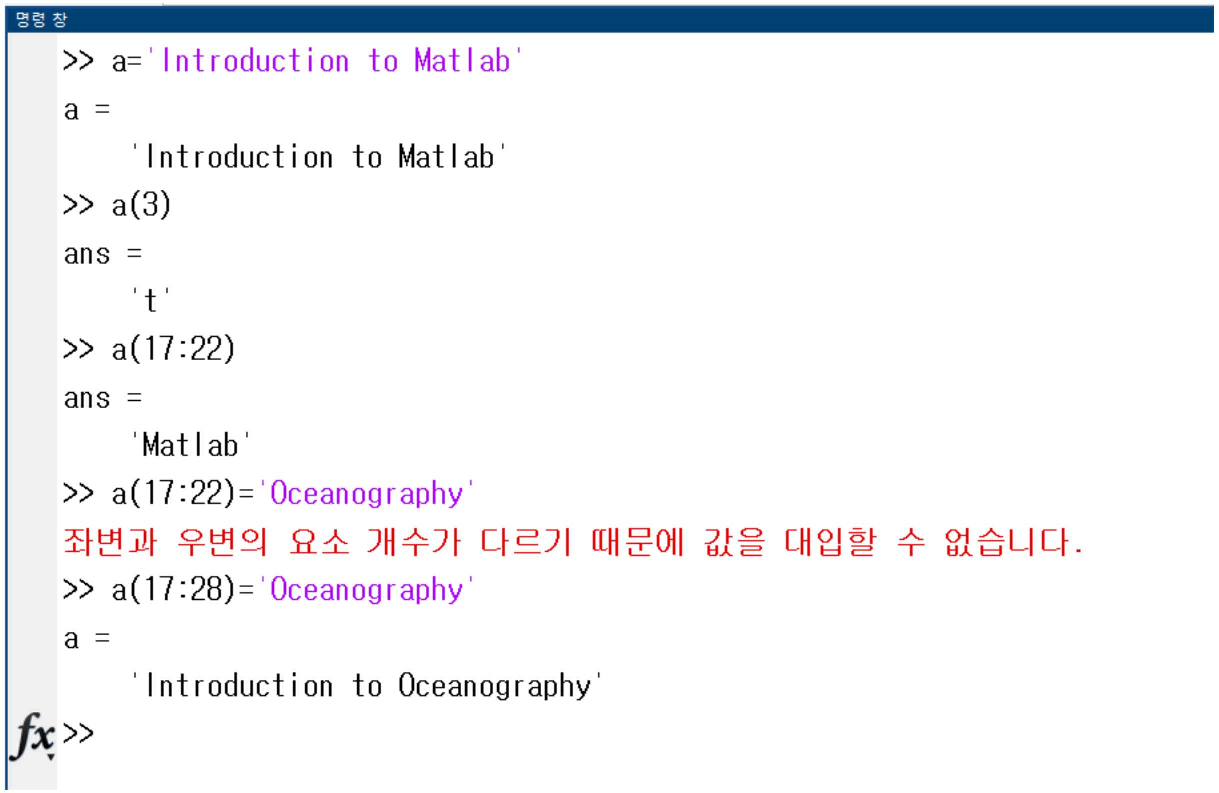
```
>> a='Introduction to Matlab'
a =
    'Introduction to Matlab'
fx >>
```

- When a variable is defined as a string, the characters of the string are stored in an array just as numbers are.

- Each character, including a space, is an element in the array.

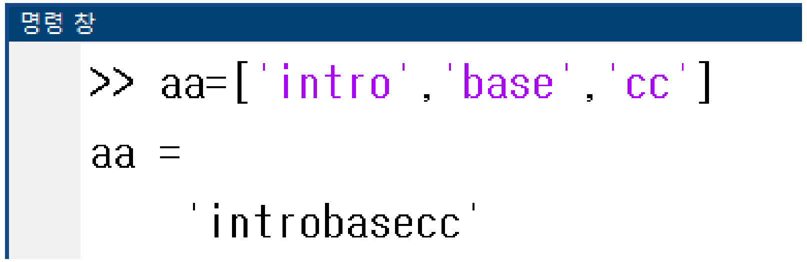
- This means that a one-line string is a row vector in which the number of elements is equal to the number of characters. The elements of the vectors are addressed by position.

- As with a vector that contains numbers, it is also possible to change specific elements by addressing them directly.



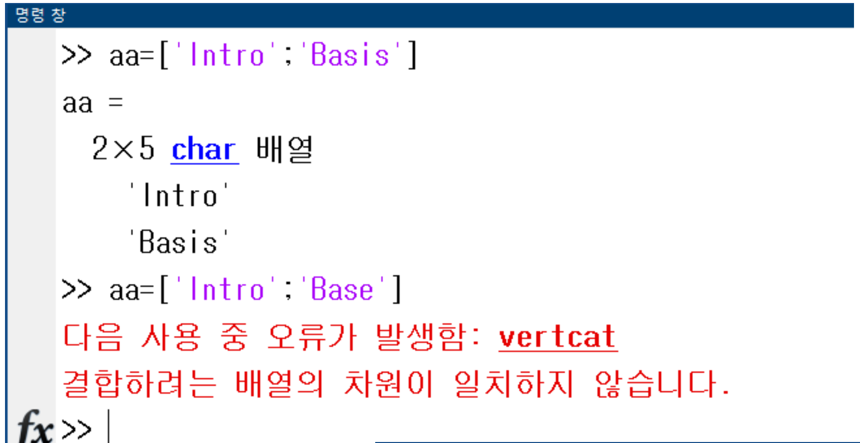
```
>> a='Introduction to Matlab'
a =
    'Introduction to Matlab'
>> a(3)
ans =
    't'
>> a(17:22)
ans =
    'Matlab'
>> a(17:22)='Oceanography'
좌변과 우변의 요소 개수가 다르기 때문에 값을 대입할 수 없습니다.
>> a(17:28)='Oceanography'
a =
    'Introduction to Oceanography'
>>
```

- Strings can also be placed in a matrix. As with numbers, this is done by typing a semicolon ; (or pressing the **Enter** key) at the end of each row. Each row must be typed as a string, which means that it must be enclosed in single quotes.


명령 창

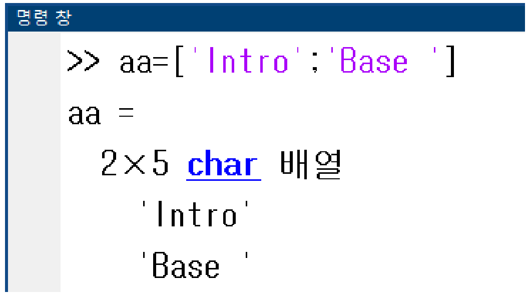
```
>> aa=['intro','base','cc']
aa =
    'introbasecc'
```

- In addition, as with a numerical matrix, all rows must have the same number of elements. This requirement can cause problems when the intention is to create rows with specific wording. Rows can be made to have the same number of elements by adding spaces.


명령 창

```
>> aa=['Intro';'Basis']
aa =
  2×5 char 배열
    'Intro'
    'Basis'
>> aa=['Intro';'Base']
다음 사용 중 오류가 발생함: vertcat
결합하려는 배열의 차원이 일치하지 않습니다.
>> 
```

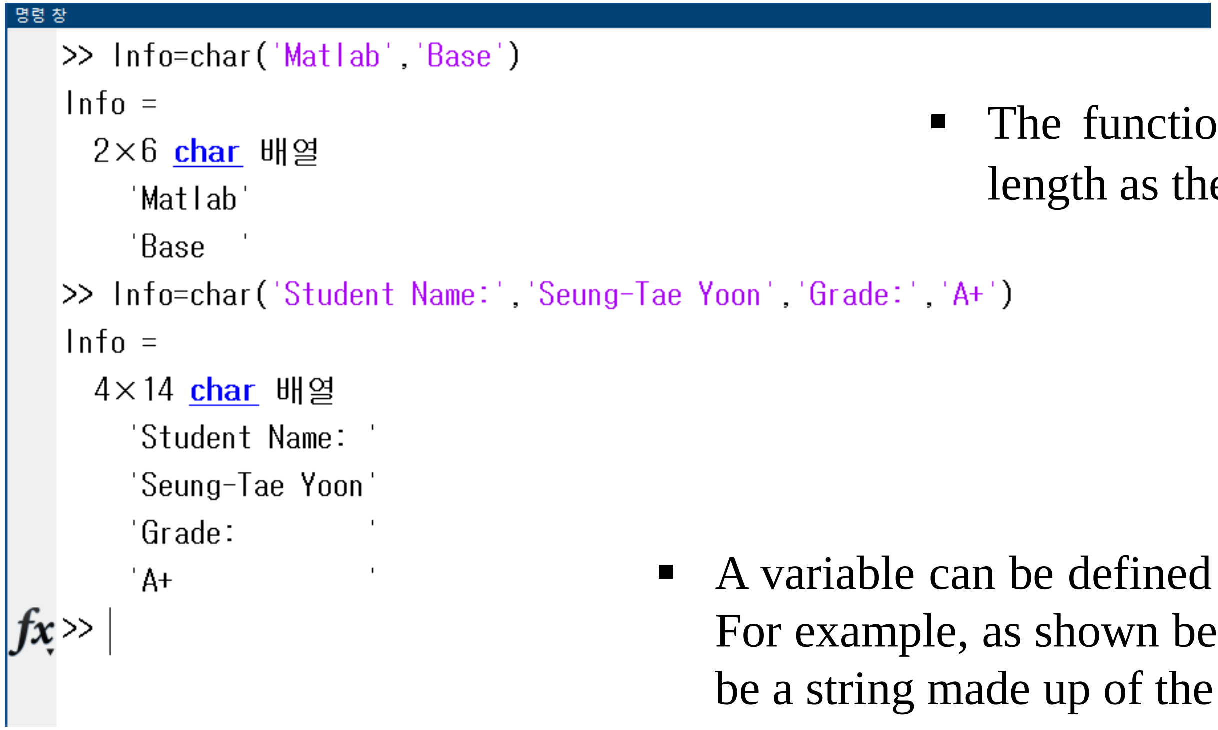
- MATLAB has a built-in function named `char` that creates an array with rows having the same number of characters from an input of rows not all of the same length. MATLAB makes the length of all the rows equal to that of the longest row by adding spaces at the end of the short lines.


명령 창

```
>> aa=['Intro';'Base ']
aa =
  2×5 char 배열
    'Intro'
    'Base '
```

- In the `char` function, the rows are entered as strings separated by a comma according to the following format:

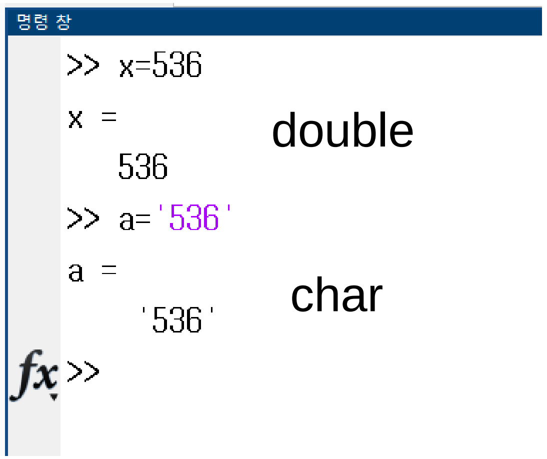
```
variable_name = char('string 1','string 2','string 3')
```



```
>> Info=char('Matlab','Base')
Info =
  2×6 char 배열
    'Matlab'
    'Base  '
>> Info=char('Student Name:','Seung-Tae Yoon','Grade:','A+')
Info =
  4×14 char 배열
    'Student Name: '
    'Seung-Tae Yoon'
    'Grade:        '
    'A+            '
>>
```

- The function `char` creates an array with four rows with the same length as the longest row by adding empty spaces to the shorter lines.

- A variable can be defined as either a number or a string made up of the same digits. For example, as shown below, x is defined to be the number 536, and y is defined to be a string made up of the digits 536.

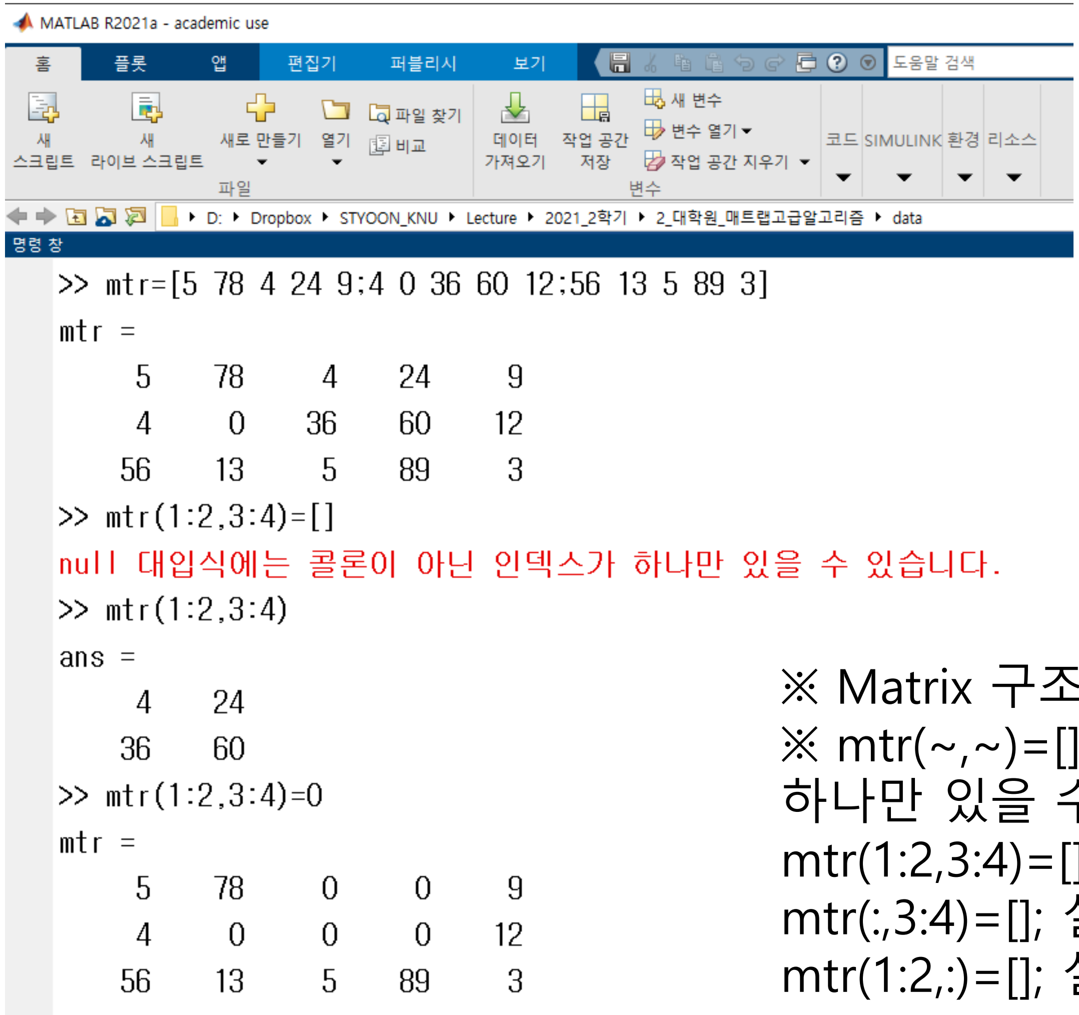


```
>> x=536
x =
   536
>> a='536'
a =
    '536'
>>
```

double

char

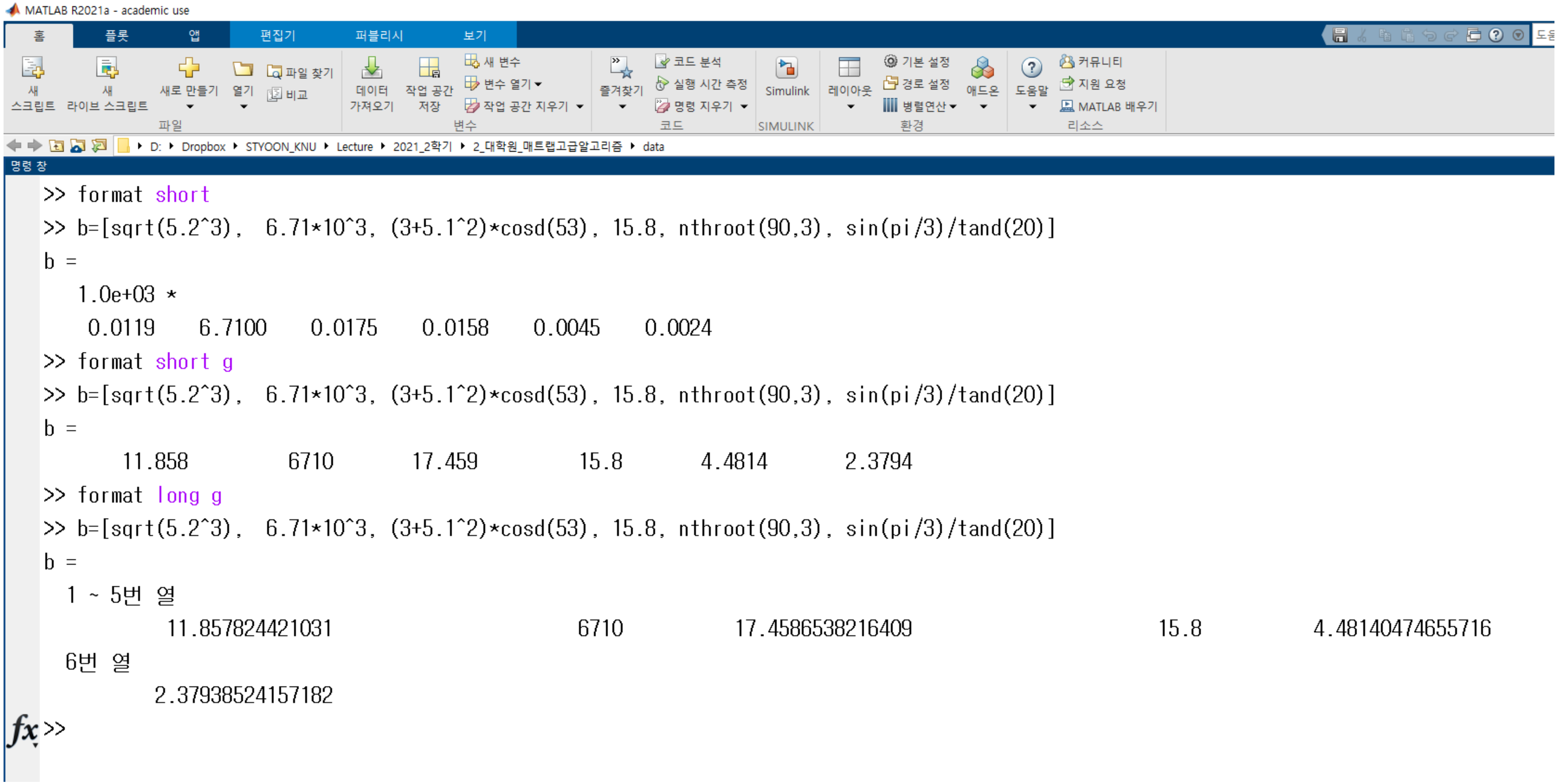
※ Deleting elements



```
>> mtr=[5 78 4 24 9;4 0 36 60 12;56 13 5 89 3]
mtr =
     5    78     4    24     9
     4     0    36    60    12
    56    13     5    89     3
>> mtr(1:2,3:4)=[]
null 대입식에는 콜론이 아닌 인덱스가 하나만 있을 수 있습니다.
>> mtr(1:2,3:4)
ans =
     4    24
    36    60
>> mtr(1:2,3:4)=0
mtr =
     5    78     0     0     9
     4     0     0     0    12
    56    13     5    89     3
```

※ Matrix 구조가 흐트러지는 상황은 실행 안됨.
※ mtr(~,~)=[]와 같은 null 대입식에는 콜론이 아닌 인덱스가 하나만 있을 수 있음.
mtr(1:2,3:4)=[]; 실행 안됨.
mtr(:,3:4)=[]; 실행 됨.
mtr(1:2,:)=[]; 실행 됨.

※ Output format



```
>> format short
>> b=[sqrt(5.2^3),  6.71*10^3, (3+5.1^2)*cosd(53), 15.8, nthroot(90,3), sin(pi/3)/tand(20)]
b =
   1.0e+03 *
    0.0119    6.7100    0.0175    0.0158    0.0045    0.0024
>> format short g
>> b=[sqrt(5.2^3),  6.71*10^3, (3+5.1^2)*cosd(53), 15.8, nthroot(90,3), sin(pi/3)/tand(20)]
b =
        11.858          6710        17.459          15.8        4.4814        2.3794
>> format long g
>> b=[sqrt(5.2^3),  6.71*10^3, (3+5.1^2)*cosd(53), 15.8, nthroot(90,3), sin(pi/3)/tand(20)]
b =
  1 ~ 5번 열
          11.857824421031                      6710          17.4586538216409                      15.8          4.48140474655716
  6번 열
         2.37938524157182
>>
```

2.11 Problems

1. Create a variable `a` that is a row vector with the following elements: 9, 1, 3^2, 7/4, 0, 2.25×8.5, 0.8, and $\sin(\pi / 8)$.

2. Create a variable `b` that is a row vector with the following elements: $\sqrt{5.2^3}$, 6.71×10^3, $(3 + 5.1^2)\cos 53°$, 15.8, $\sqrt[3]{90}$, and $\dfrac{\sin(\pi/3)}{\tan 20°}$.

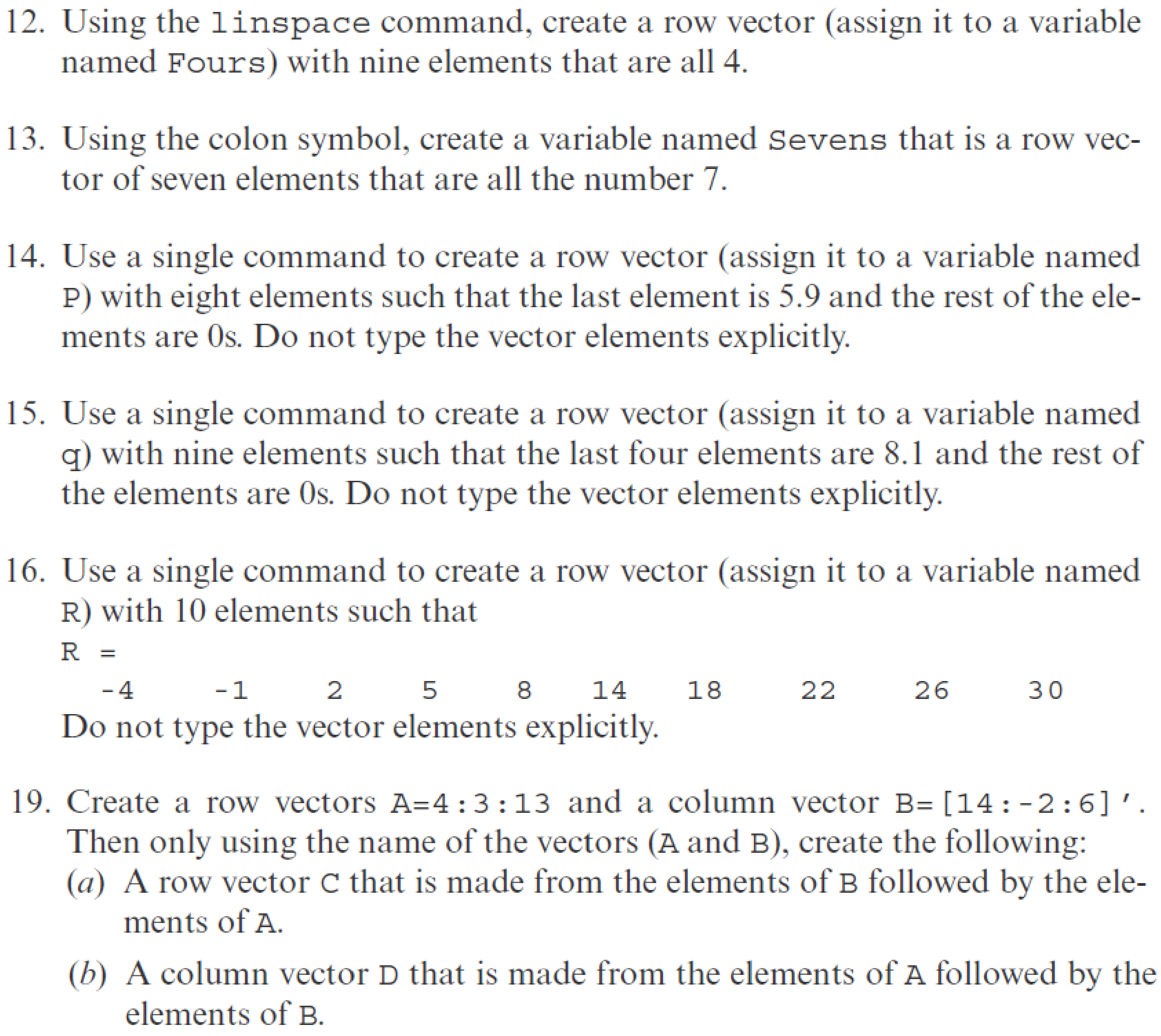
3. Create a variable `c` that is a column vector with the following elements: 2.1×10^{-2}, $\sin(1.7\pi)$, 28.5, $2.7^{4/3}$, and e^3.

4. Create a variable `d` that is a column vector with the following elements: $0.75 \times 5.2^{0.7}$, 11.1, $\sqrt[3]{60}$, $\tan(10\pi / 11)$, $\cos^2 5°$, and 0.116.

5. Define the variables $x = 3.4$ and $y = 5.8$, and then use them to create a row vector (assign it to a variable named `e`) that has the following elements: x/y, $x+y$, x^y, $x \times y$, y^2, and x.

6. Define the variables $c = 4.5$ and $d = 2.8$, and then use them to create a column vector (assign it to a variable named `f`) that has the following elements: d^2, c, $(c+d)$, c^d, and d.

7. Create a variable `g` that is a row vector in which the first element is 3 and the last element is 27, with an increment of 4 between the elements (3, 7, 11, … , 27).



12. Using the `linspace` command, create a row vector (assign it to a variable named `Fours`) with nine elements that are all 4.

13. Using the colon symbol, create a variable named `Sevens` that is a row vector of seven elements that are all the number 7.

14. Use a single command to create a row vector (assign it to a variable named `P`) with eight elements such that the last element is 5.9 and the rest of the elements are 0s. Do not type the vector elements explicitly.

15. Use a single command to create a row vector (assign it to a variable named `q`) with nine elements such that the last four elements are 8.1 and the rest of the elements are 0s. Do not type the vector elements explicitly.

16. Use a single command to create a row vector (assign it to a variable named `R`) with 10 elements such that

```
R =
  -4     -1     2     5     8    14    18     22     26     30
```

Do not type the vector elements explicitly.

19. Create a row vectors `A=4:3:13` and a column vector `B=[14:-2:6]'`. Then only using the name of the vectors (`A` and `B`), create the following:
    (*a*) A row vector `C` that is made from the elements of `B` followed by the elements of `A`.

    (*b*) A column vector `D` that is made from the elements of `A` followed by the elements of `B`.

21. Create a row vector `vC=2:3:38` that has 13 elements. Then, create the following new vectors by assigning elements of `vC` to the new vectors:
    (*a*) A vector (name it `vCodd`) that contains all the elements with odd index of `vC`; i.e., `vCodd = 2 8 14 ... 38`.
    (*b*) A vector (name it `vCeven`) that contains all the elements with even index of `vC`; i.e., `vCeven = 5 11 17 ... 35`.

    In both parts use vectors of odd and even numbers to address the elements of `vC` that are assigned to `vCodd`, and `vCeven`, respectively. Do not enter the elements of the vectors explicitly.

22. Create two row vectors `vD=20:4:44` and `vE=50:3:71`. Then, create the following new vectors by assigning elements of `vD` and `vE` to the new vectors:
    (*a*) A vector (name it `vDE`) that contains the 2nd through the 5th elements of `vD` and the 4th through 7th elements of `vE`; i.e., `vDE = 24 28 32 36 59 62 65 68`.
    (*b*) A vector (name it `vED`) that contains elements 6, 5, 4, 3, and 2 of `vE` and elements 4, 3, 2, and 1 of `vD`; i.e., `vED = 65 62 59 56 53 32 28 24 20`.

    In both parts use vectors to address the elements of `vD` and `vE` that are assigned to `vDE` and `vED`, respectively. Do not enter the elements of the vectors explicitly.

23. Create a nine-element row vector `vF=5:7:61`. Then create a vector (name it `vFrev`) that consist of the elements of `vF` in reverse order. Do it by using a vector to address the elements of `VF`. (Do not type the elements of `vF` vector explicitly.)

24. Create the following matrix by assigning vectors with constant spacing to the rows (use the `linspace` command for the third row). Do not type individual elements explicitly.

```
A =
    1.0000 2.0000 3.0000 4.0000 5.0000 6.0000 7.0000
    7.0000 6.0000 5.0000 4.0000 3.0000 2.0000 1.0000
    2.0000 3.1667 4.3333 5.5000 6.6667 7.8333 9.0000
```

27. Create the following matrix by typing one command. Do not type individual elements explicitly.

```
D =
     1      1      1      1
     1      1      1      1
     1      1      1      1
     8      6      4      2
```

29. Create the following matrix by typing one command. Do not type individual elements explicitly.

```
F =
     0      0      0      0      0      0
     0      0      0      0      0      0
     0      0      8      6      4      2
```

30. Create the following matrix by typing one command. Do not type individual elements explicitly.

```
G =
     1      1      1      1      1
     1      1      1      1      1
     1      1      1      1      1
     0      0      0      1      1
     0      0      0      1      1
     0      0      0      1      1
```

32. Create the following three row vectors:

`a=[5 8 -1 0 2]`, `b=[4 1 9 -2 3]`, and `c=[-3 5 0 6 1]`.

(*a*) Use the three vectors in a MATLAB command to create a 3×5 matrix in which the rows are the vectors `c`, `b`, and `a`, respectively.

(*b*) Use the three vectors in a MATLAB command to create a 5×3 matrix in which the columns are the vectors `c`, `b`, and `a`, respectively.

39. Create the following matrix `G`:

```
G =
      0.1    0.2    0.3    0.4    0.5    0.6    0.7
       10      9      8      7      6      5      4
        0    0.2    0.4    0.6    0.8    1.0    1.2
        5      3      1     -1     -3     -5     -7
```

(*a*) Create a 3×4 matrix `Ma` from the first, third, and fourth rows and the first two and last two columns of matrix `G`.

(*b*) Create a 3×3 matrix `Mb` from the first three rows and the second, fourth, and sixth columns of matrix `G`.

44. Using the `zeros`, `ones`, and `eye` commands, create the following arrays by typing one command:

(*a*) $\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$ (*b*) $\begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$ (*c*) $\begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$

45. Use the `eye`, `ones`, and `zeros` commands to create the following arrays:

$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $B = [1 \;\; 1]$ $C = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Using the variables `A`, `B`, and `C`, write a command that creates the following matrix `D`:

$$D = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

In next time,

- Mathematical operations with Arrays

If you have any questions,
Please contact styoon@knu.ac.kr