M'hamed Bougara University - Boumerdès
Faculty of Sciences
Computer Science Department

Course: Programming Tools for Mathematics Matlab
L1

Dr RAHMOUNE Nabila

2023/2024

Chapter 1
Introduction to Matlab

Outline

- Introduction
- MATLAB Components
- Variables
- Vectors and matrices
- Applications

What Is MATLAB?

MATLAB (MATrix LABoratory) developed by The Mathworks, Inc. (http://www.mathworks.com)

- high-performance language for technical computing
- computation, visualization and programming in an easy-to-use environment

Typical uses include:

- Math and computation
- Algorithm development
- Modelling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including Graphical User Interface building

Why MATLAB?

- Easy to do very rapid prototyping
- Fast implementation and debugging
- Quick to learn, and good documentation
- A good library of image processing functions
- Excellent display capabilities
- Widely used for teaching and research in universities and industry

MATLAB Components

- **The MATLAB language**
- a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features.

- **The MATLAB working environment**
- the set of tools and facilities that you work with as the MATLAB user or programmer, including tools for developing, managing, debugging, and profiling

- **Handle Graphics**
- the MATLAB graphics system. It includes high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics.

MATLAB Components

- **The MATLAB function library.**
- a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms as well as special image processing related functions

- **The MATLAB Application Program Interface (API)**
- a library that allows you to write C and Fortran programs that interact with MATLAB. It include facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

Some Elements and Characteristics

- Everything in MATLAB is a matrix
- MATLAB is an interpreted language, no compilation needed (but possible)
- MATLAB does not need any variable declarations, no dimension statements, has no packaging, no storage allocation, no pointers
- Programs can be run step by step, with full access to all variables, functions etc.

Online Help

The help command	>> help
The help window	>> helpwin
The lookfor command	>> lookfor

```
» help cd
  CD Change current working directory.
  CD directory-spec sets the current directory to the one specified.
  CD .. moves to the directory above the current one.
  CD, by itself, prints out the current directory.
  WD = CD returns the current directory as a string.
   Use the functional form of CD, such as CD('directory-spec'),   when the
    directory specification is stored in a string.
```

Calculations at the Command Line

MATLAB as a calculator

```
» -5/(4.8+5.32)^2
ans =
   -0.0488
»(3+4i)*(3-4i)
ans =
    25
»cos(pi/2)
ans =
  6.1230e-017
»exp(acos(0.3))
ans =
    3.5470
```

Assigning Variables

```
» a = 2;
» b = 5;
» a^b
ans =
   32
» x = 5/2*pi;
» y = sin(x)
y =
    1
» z = asin(y)
z =
   1.5708
```

Semicolon suppresses screen output

Results assigned to "ans" if name not specified

() parentheses for function inputs

Numbers stored in double-precision floating point format

»cmd_line

Working with Variables

CD / PWD, LS / DIR - navigating directories

WHAT - displays the files within a directory (grouped by type)

! - invoke operating system

WHICH - identifies the object referenced by given name (function / variable)

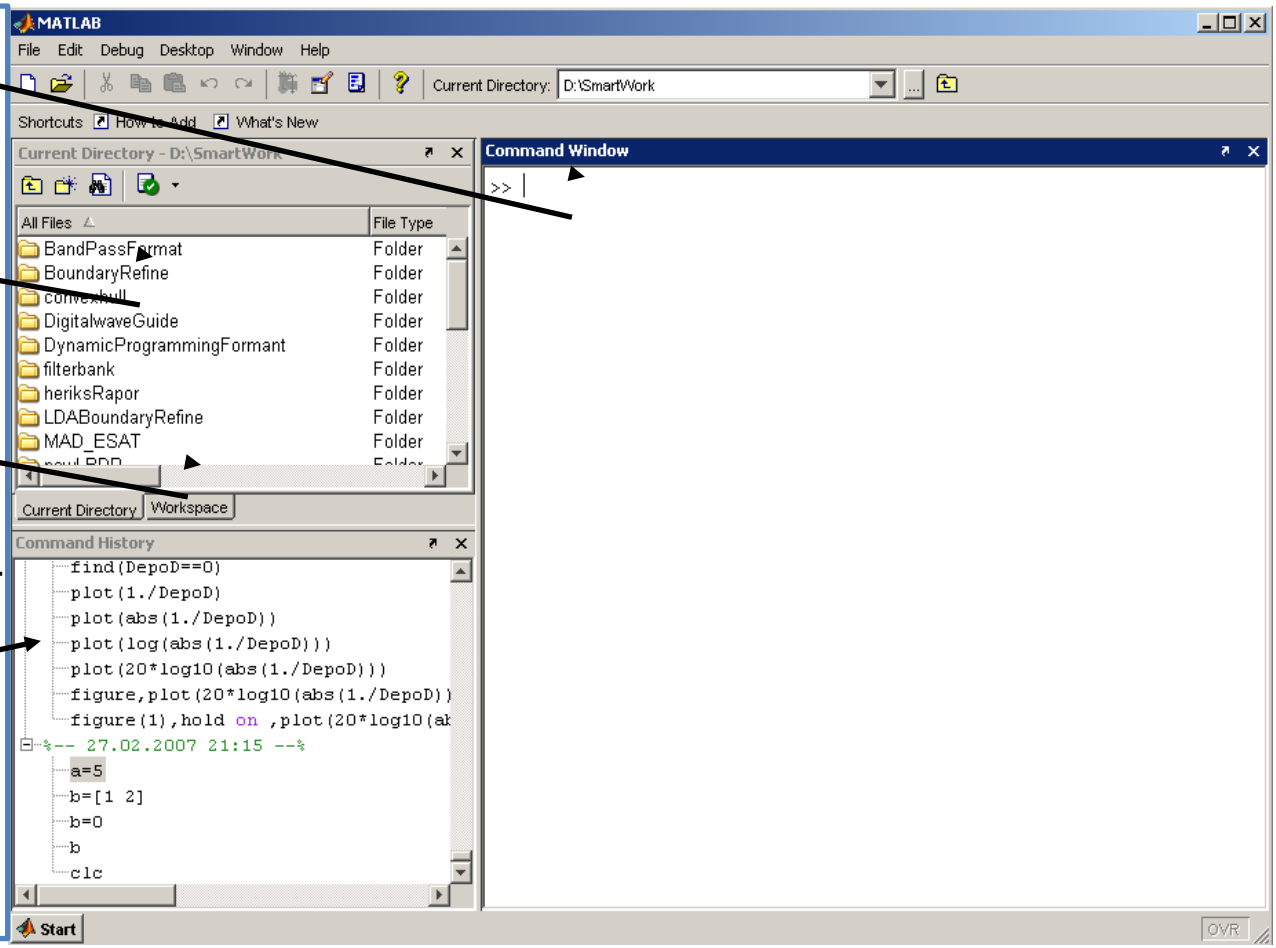
CLEAR - remove function / variable from memory

WHOS - lists workspace variables and details (size, memory usage, data type)

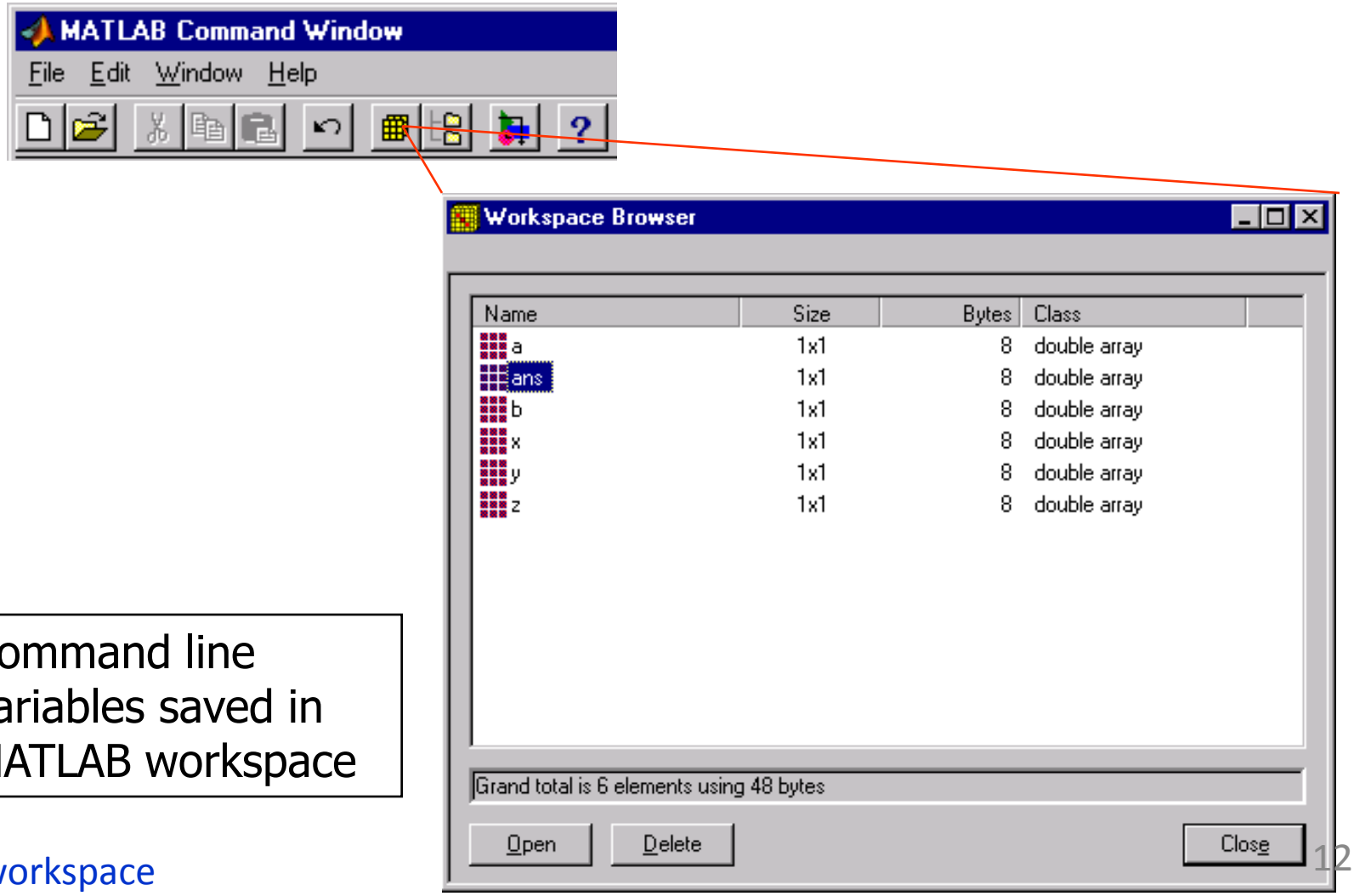
SIZE - returns the size of matrix

Matlab Main Screen

- **Command Window**
  - type commands
- **Current Directory**
  - View folders and m-files
- **Workspace**
  - View variables
  - Double click on a variable to see it in the Array Editor
- **Command History**
  - view past commands
  - save a whole session using diary



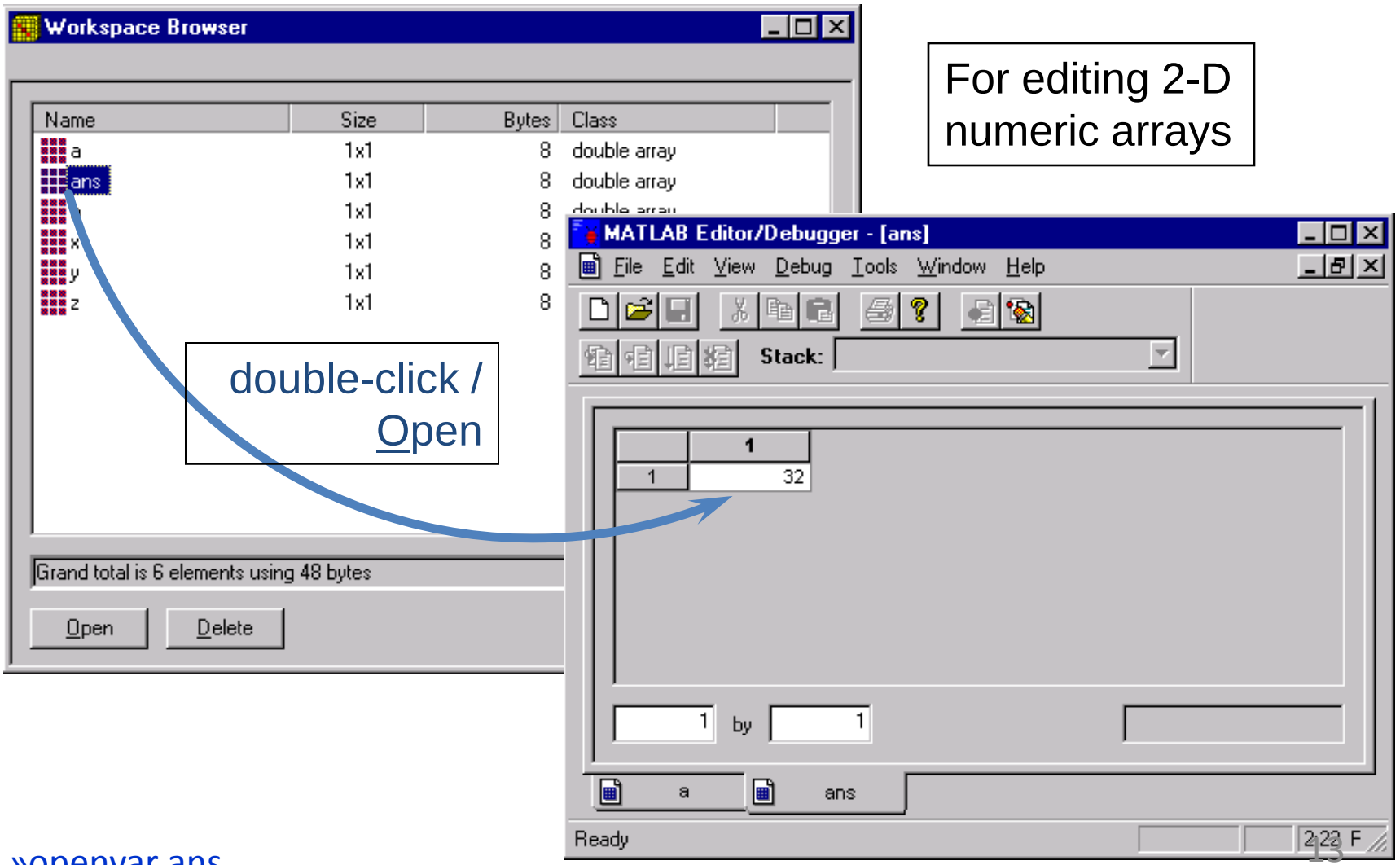
Workspace Browser



Command line variables saved in MATLAB workspace

»workspace

Array Editor



For editing 2-D numeric arrays

»openvar ans

Some MATLAB Workspace Functions

- You can see the variables in your workspace by looking at the *Workspace* window or by using the **whos** command:

```
>> whos
```

Name	Size	Bytes	Class	Attributes
area	1x1	8	double	
height	1x1	8	double	
Width	1x1	8	double	

- If you want to remove a variable from the MATLAB you can use the **clear** command, e.g.,

```
>> clear width
```

removes width from the workspace.

- If you want to get rid of all the variables in your workspace just type:

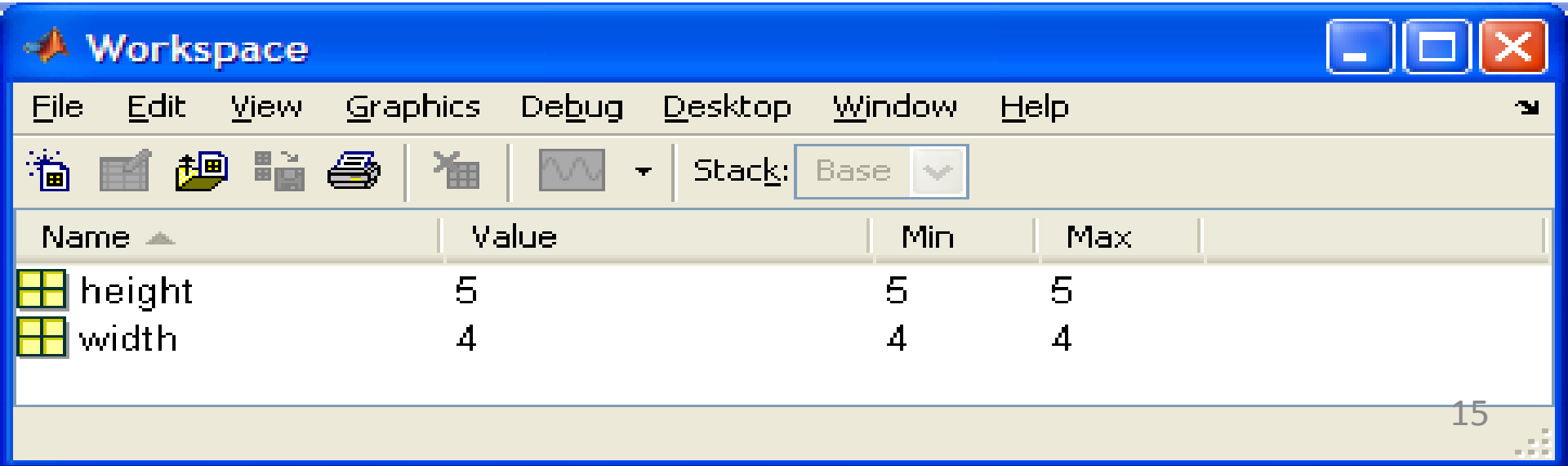
```
>> clear
```

Workspace Browser

For example, after entering:

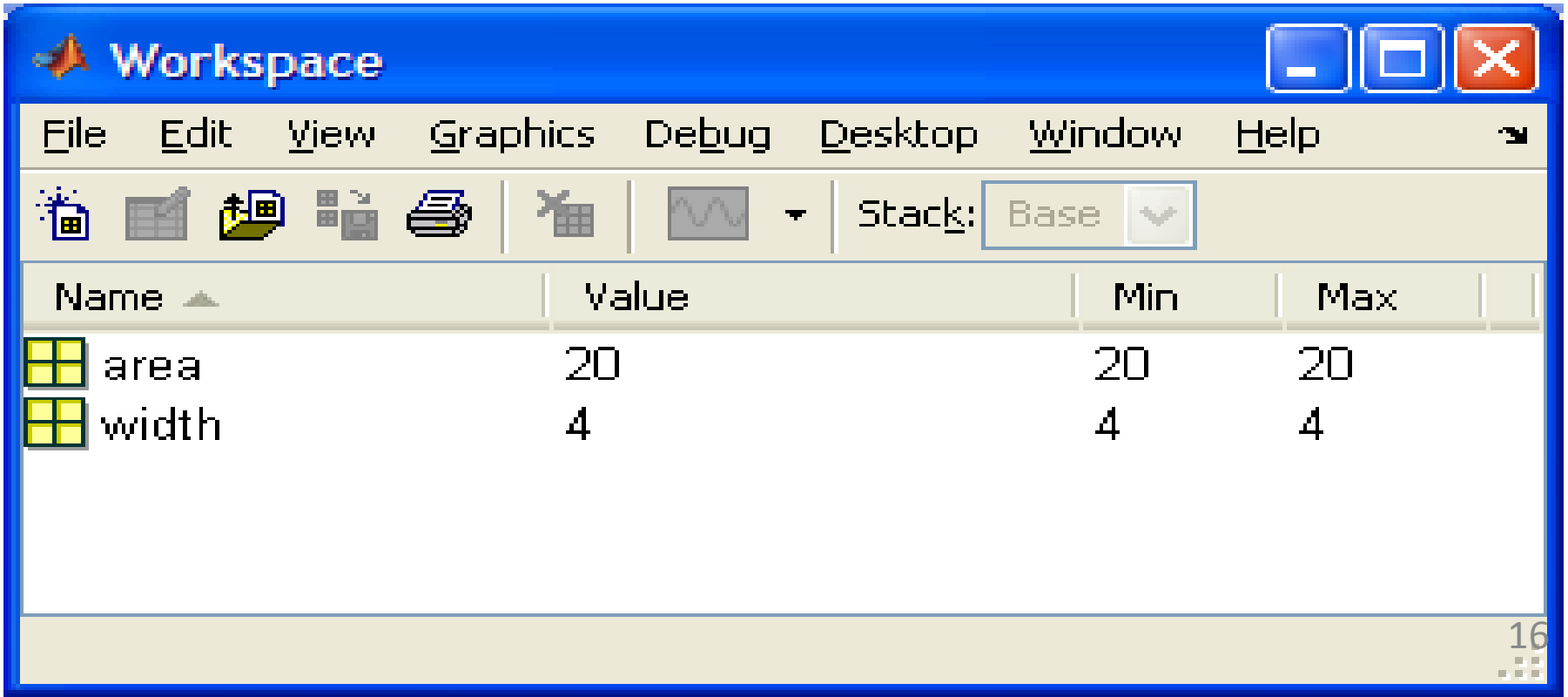
```
>> height = 5
height =
5
>> width = 4
width =
4
```

the *Workspace* window will contain both **height** and **width** as scalars:



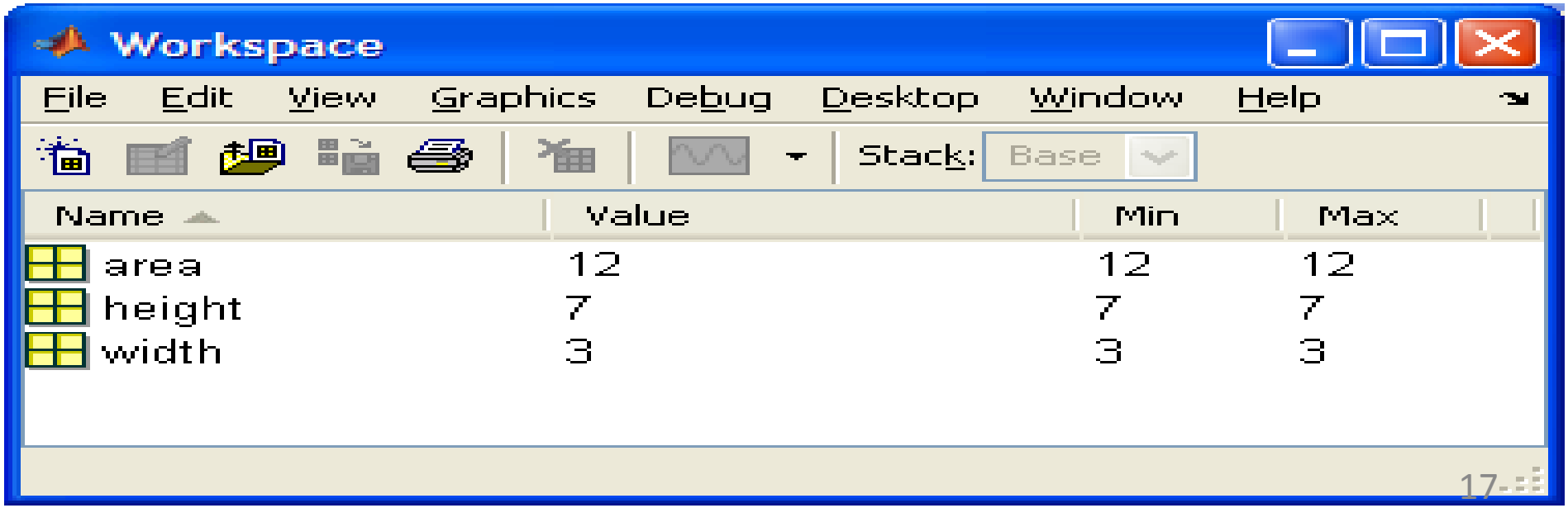
Workspace(cont)

If you want it back! Press the up-arrow key until the earlier command:
>> height = 5
appears in the Command Window. Now press enter



Workspace(cont)

MATLAB suppresses the output of a command if you finish the command with a semi-colon - **;**.
Example:
>> height = 7;
If you look at **height** in your *Workspace* window you will see its value has changed to 7:



Matlab Optimization

MATLAB is matrix-oriented, so what would take several statements in C or another language can usually be accomplished in just a few lines using MATLAB's built-in matrix and vector operations.

C:

```
float  A[10][10], B[10][10], C[10][10] ;
for (int i=0;i<10;i++)
{   for (int j=0; j<10;j++)
    { C[i][j] = A[i][j] + B[i][j]; }
}
```

MATLAB:

```
C = A + B
```

Operators

==	Equal to
~=	Not equal to
<	Strictly smaller
>	Strictly greater
<=	Smaller than or equal to
>=	Greater than equal to
&	And operator
|	Or operator

Variables

MATLAB variables are created when they are used with an assignment statement.

All variables are created as matrices with "some" type (unless specified)

Syntax :

variable name = a value (or an expression)

For example:

```
>> x = expression
```

where expression is a combination of numerical values, mathematical operators, variables, and function calls.

Variables

All variables are created as matrices with "some" type (unless specified)

No need for types. i.e.,

~~int a;~~
~~double b;~~
~~float c;~~

C/C++

```
int a;
a=1;
double b;
b=2+4;
```

Matlab

```
>>a=1;
>>b=2+4;
```

```
>> a=1
a =
     1
>> b=2+4
b =
     6
```

```
myNumberVariable = 3.14
myStringVariable = 'hello world!'
```

Variables

- Special variables:
  ans : default variable name for the result
  pi: π = 3.1415926……
  eps: $\in$ = 2.2204e-016, smallest amount by which 2 numbers can differ.
  Inf or **inf** : ∞, infinity
  NaN or **nan**: not-a-number

- Commands involving variables:
  who: lists the names of defined variables
  whos: lists the names and sizes of defined variables
  clear: clears all varialbes, reset the default values of special variables.
  clear *name*: clears the variable *name*
  clc: clears the command window
  clf: clears the current figure and the graph window.

Variables

Rules on Variable Names

There are a number of rules as to the variable names you can use:

1. must start with a letter, followed by letters, digits, or underscores. **X12**, **rate_const**, **Flow_rate** are all acceptable variable names but not **vector-A** (since **-** is a reserved character);

2. are case sensitive, e.g., **FLOW**, **flow**, **Flow**, **FlOw** are all different variables;

3. must not be longer than 31 characters;

4. must not be a reserved word (i.e., special names which are part of MATLAB language);

Variables

Naming Variables

Don’t use these names for anything :

`i, j:` can be used to indicate complex numbers*

`Pi:` has the value 3.1415...

`ans:` stores the result of the last unassigned value

`Inf, -Inf:` infinities

`NaN:` "Not Number"

Use `ii, jj, kk,` etc. for loop counters.

Variables Types

MATLAB is a "weakly typed" language

- No need to initialize variables!

MATLAB supports various types; the most popular ones are

- 3.84
  - 64-bit double (default)
- 'A'
  - 16-bit char

Most variables you'll deal with are vectors, matrices, doubles or chars

Other types are also supported: complex, symbolic, 16-bit and 8-bit integers (uint16 & uint8), etc.

Variables Types: Scalars

- A variable can be given a value explicitly
  - `a = 10`
  - Shows up in workspace!
- Or as a function of explicit values and existing variables
  - `c = 1.3 * 45 - 2 * a`
- To suppress output, end the line with a semicolon
  - `cooldude = 13/3;`

Variables Types: Strings

A = ‘vision and geometry’

strfind(A, ‘geometry')

strcmp(A,'computer vision') B =

strcat(A,' 12345')

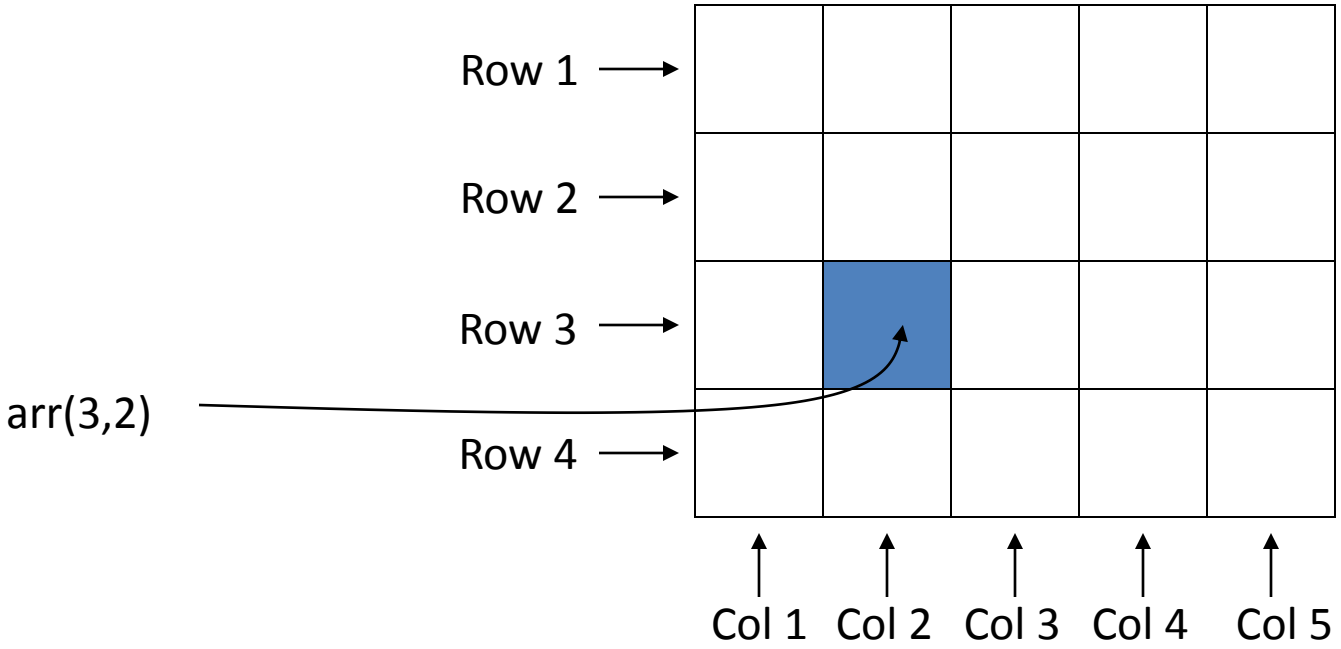
c = [A,' 12345']

D = sprintf('I am %02d years old.\n',9) int2str,

str2num, str2double

Variables Types: Arrays

Array: A collection of data values organized into rows and columns, and known by a single name.



Variables Types: Arrays

Arrays

The fundamental unit of data in MATLAB

Scalars are also treated as arrays by MATLAB (1 row and 1 column).

Row and column indices of an array start from 1.

Arrays can be classified as **vectors** and **matrices**.

Variables Types: Arrays

Arrays

The fundamental unit of data in MATLAB

Scalars are also treated as arrays by MATLAB (1 row and 1 column).

Row and column indices of an array start from 1.

Arrays can be classified as **vectors** and **matrices**.

Vector: Array with one dimension

Matrix: Array with more than one dimension

Size of an array is specified by the number of rows and the number of columns, with the number of rows mentioned first (For example: n x m array).

Total number of elements in an array is the product of the number of rows and the number of columns.

User interaction

Reading : input(‘message’)
Writing : disp(‘message’)

Exercise: Calculate the sum of two integers entered by the user.

Solution 1 :

```
v1= input(' Enter the first value' );
v2= input(' Enter the second value' );
sm=v1+v2;
disp([‘the sum of the two values is ', num2str(sm)])
```

User interaction

Reading : input(‘message’)
Writing : disp(‘message’)

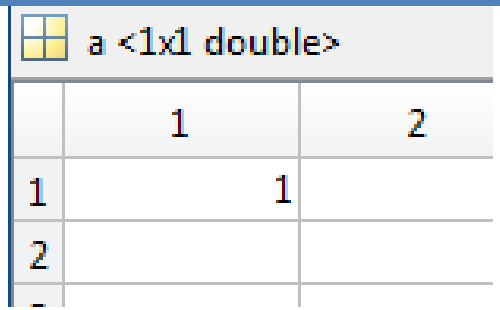
Exercise: Calculate the sum of two integers entered by the user.

Solution 2 :

```
v= input(‘Enter the two values : ', 's');
v = str2num(v) ;
sm=v(1)+v(2);
disp([‘the sum of the two values is ', num2str(sm)])
```

Creating variables

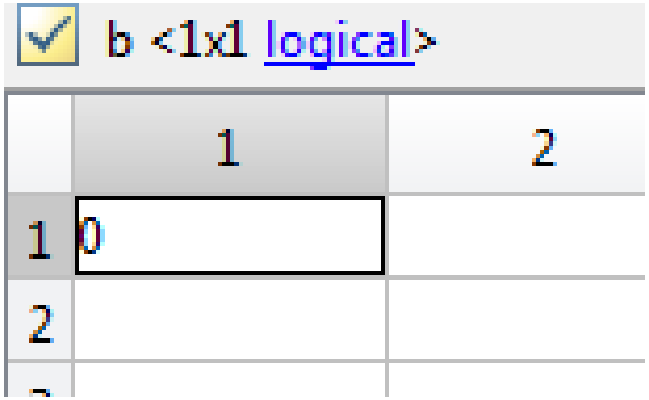
```
a = 1;
```




```
>> whos a
  Name        Size              Bytes  Class     Attributes

  a           1x1                   8  double
```

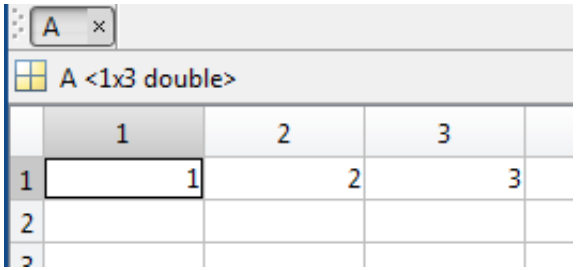
```
b = false;
```



Creating variables

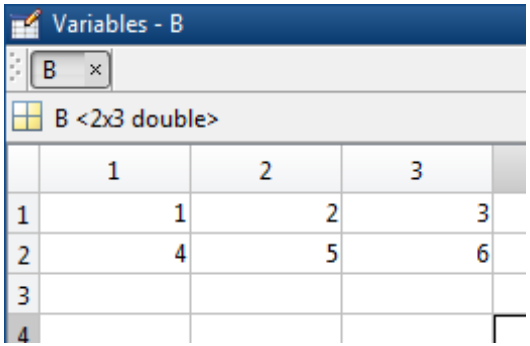
A = [1, 2, 3]

```
>> A=[1 2 3]
A =
     1     2     3
```



B = [1,2,3;4,5,6]

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



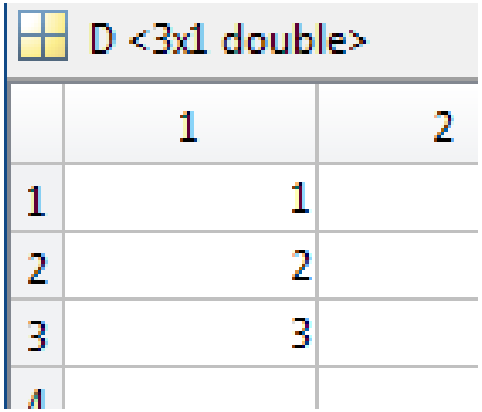
C=[1 2 3;4 5 6;7 8 9]

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

Creating variables

D=[1 ; 2 ; 3]

```
>> D= [1 ;2 ;3]
D =
     1
     2
     3
```



E=[1 2 3]’

```
>> E=[1 2 3]'
E =
     1
     2
     3
```

Creating variables

```
>> A=1:10
A =
     1     2     3     4     5     6     7     8     9    10
>> B= 0:2:10
B =
     0     2     4     6     8    10
```

```
>> 1:0.5:5
ans =
    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000    4.0000    4.5000    5.0000
```

Arrays : Vectors and Matrix

a vector

```
x = [1 2 5 1]
```

```
x =
     1     2     5     1
```

a matrix

```
y = [1 2 3; 5 1 4; 3 2 -1]
```

```
y =
     1     2     3
     5     1     4
     3     2    -1
```

transpose

```
y = x'
```

```
y =
     1
     2
     5
     1
```

Vectors

- A row vector in MATLAB can be created by an explicit list, starting with a left bracket, entering the values separated by spaces (or commas) and closing the vector with a right bracket.

- A column vector can be created the same way, and the rows are separated by semicolons.

Example:

```
>> x = [ 0   0.25*pi   0.5*pi   0.75*pi   pi ]
x =
         0    0.7854    1.5708    2.3562    3.1416
>> y = [ 0; 0.25*pi; 0.5*pi; 0.75*pi; pi ]

y =
         0
    0.7854
    1.5708
    2.3562
    3.1416
```

x is a row vector.

y is a column vector.

Vectors

- row = [1 2 3.2 4 6 5.4];
- row = [1, 2, 4, 7, 4.3, 1.1];

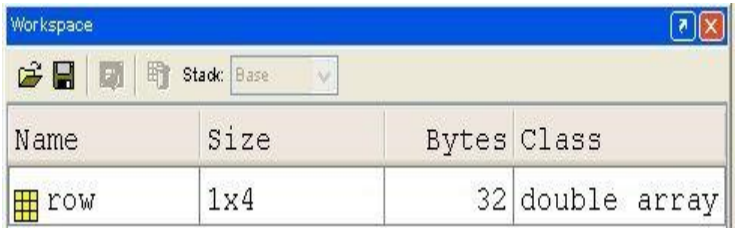
- Command window:

```
>> row=[1 2 5.4 -6.6]

row =

    1.0000    2.0000    5.4000   -6.6000
```

Workspace:



- col = [1; 2; 3.2; 4; 6; 5.4];

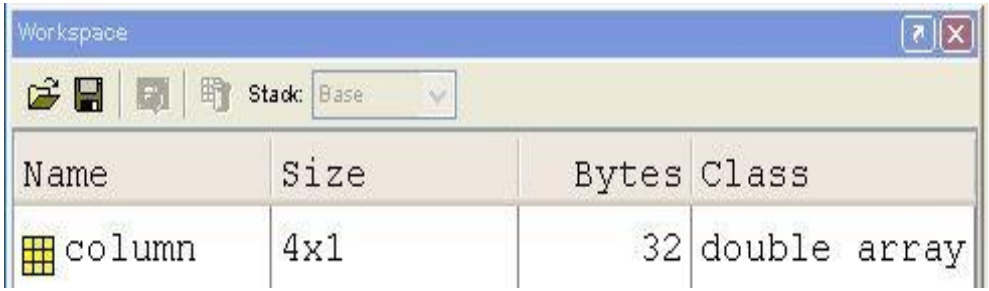
Command window:

```
>> column=[4;2;7;4]

column =

     4
     2
     7
     4
```

Workspace:



Vectors

Vector Addressing – A vector element is addressed in MATLAB with an integer index enclosed in parentheses.
Example:

```
>> x(3)
ans =
    1.5708
```

← 3rd element of vector x

- The colon notation may be used to address a block of elements.

  (start : increment : end)

  start is the starting index, increment is the amount to add to each successive index, and end is the ending index. A shortened format (start : end) may be used if increment is 1.

- Example:

```
>> x(1:3)
ans =
         0    0.7854    1.5708
```

← 1st to 3rd elements of vector x

NOTE: MATLAB index starts at 1.

Vectors

Some useful commands:

x = start:end	create row vector x starting with start, counting by one, ending at end
x = start:increment:end	create row vector x starting with start, counting by increment, ending at or before end
length(x)	returns the length of vector x
y = x’	transpose of vector x
dot (x, y)	returns the scalar dot product of the vector x and y.

Array Operations

Scalar-Array Mathematics

For addition, subtraction, multiplication, and division of an array by a scalar simply apply the operations to all elements of the array.

Example:

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

>> g = 2*f – 1
g =
     1     3
     5     7
```

Each element in the array f is multiplied by 2, then subtracted by 1.

Array Operations

Element-by-Element Array-Array Mathematics.

Operation	***Algebraic Form***	***MATLAB***
Addition	a + b	a + b
Subtraction	a – b	a – b
Multiplication	a x b	a .* b
Division	a ÷ b	a ./ b
Exponentiation	a^b	a .^ b

- **Example:**

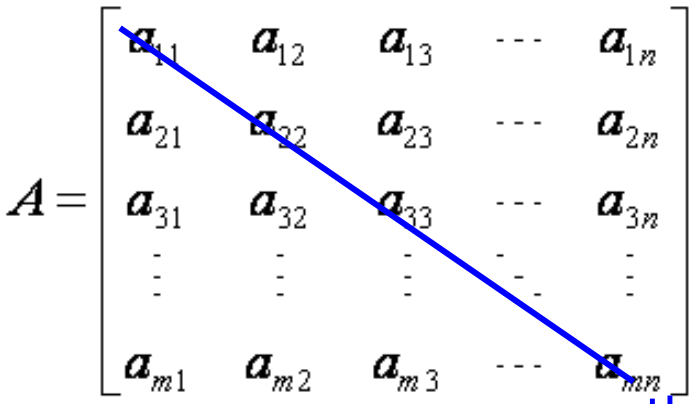
```
>> x = [ 1 2 3 ];
>> y = [ 4 5 6 ];
>> z = x .* y
z =
     4    10    18
```

Each element in x is multiplied by the corresponding element in y.

Matrices

- A Matrix array is two-dimensional, having both multiple rows and multiple columns, similar to vector arrays:
  - it begins with [, and end with]
  - spaces or commas are used to separate elements in a row
  - semicolon or enter is used to separate rows.

A is an m x n matrix.

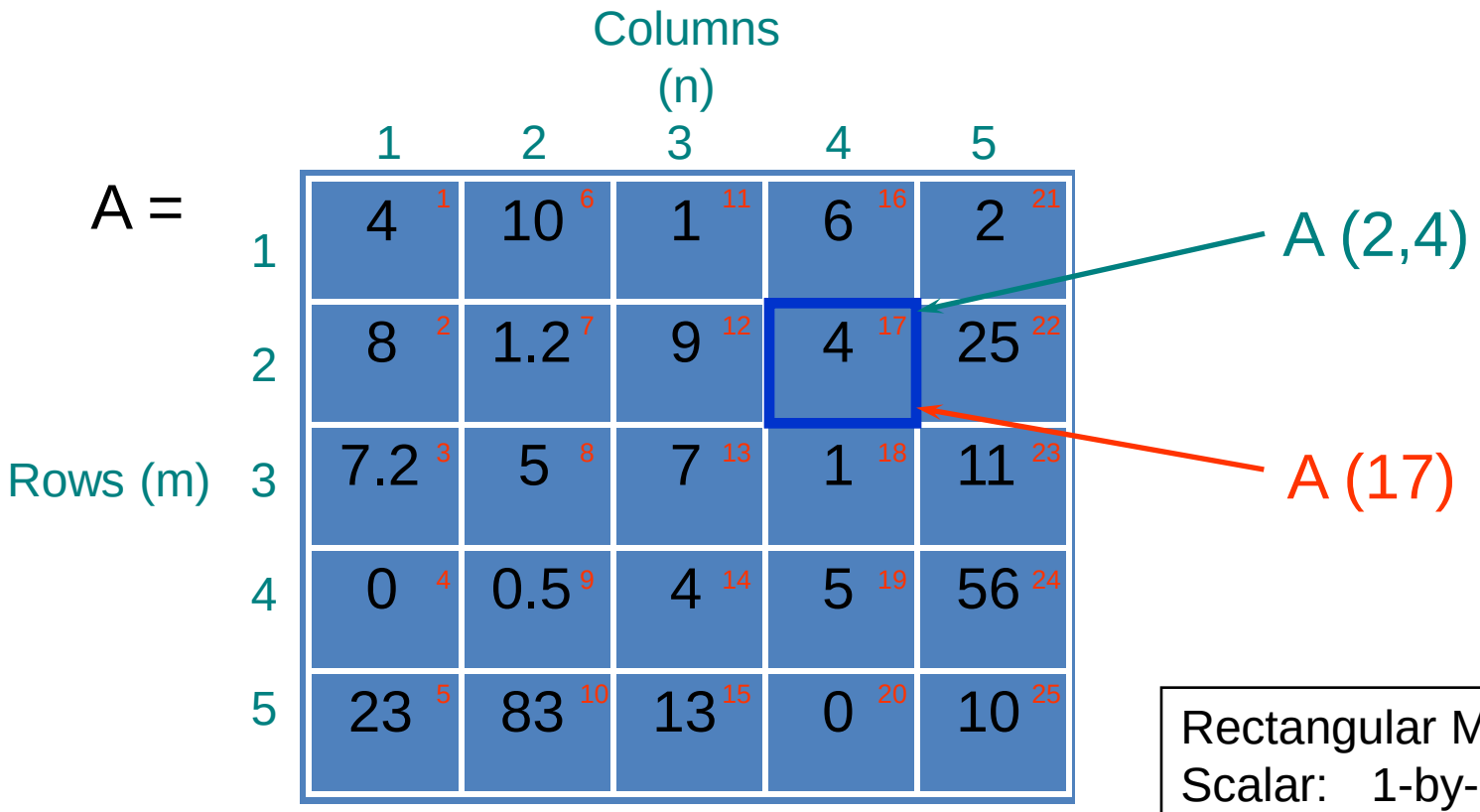


the main diagonal

•Example:

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

Matrices



Matrix elements can be EITHER numbers OR characters

Rectangular Matrix:
Scalar: 1-by-1 array
Vector: m-by-1 array
1-by-n array
Matrix: m-by-n array

Matrices

Matrix Addressing:

-- *matrixname(row, column)*

-- **colon** may be used in place of a row or column reference to select the entire row or column.

- Example:

```
>> f(2,3)

ans =

    6

>> h(:,1)

ans =

    2

    1
```

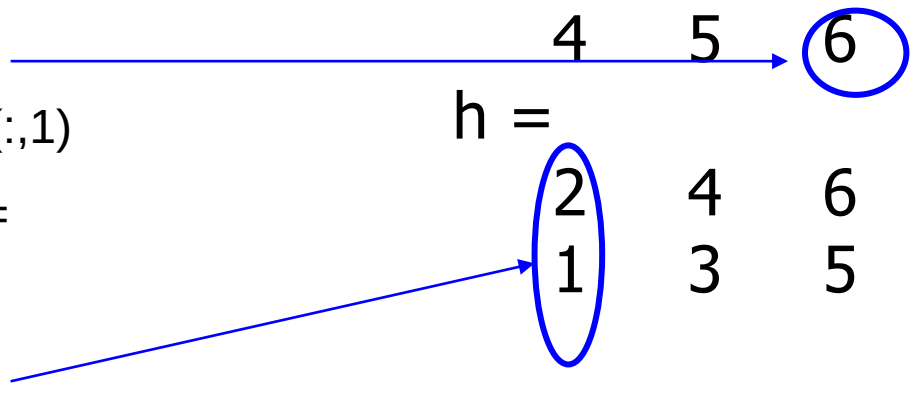
recall:

f =

1	2	3
4	5	6

h =

2	4	6
1	3	5



Matrices

Some useful commands:

zeros(n)	returns a n x n matrix of zeros
zeros(m,n)	returns a m x n matrix of zeros
ones(n)	returns a n x n matrix of ones
ones(m,n)	returns a m x n matrix of ones
size (A)	for a m x n matrix A, returns the row vector [m,n] containing the number of rows and columns in matrix.
length(A)	returns the larger of the number of rows or columns in A.

Matrices

More commands

Transpose	B = A’
Identity Matrix	eye(n) ➔ returns an n x n identity matrix eye(m,n) ➔ returns an m x n matrix with ones on the main diagonal and zeros elsewhere.
Addition and subtraction	C = A + B C = A – B
Scalar Multiplication	B = αA, where α is a scalar.
Matrix Multiplication	C = A*B
Matrix Inverse	B = inv(A), A must be a square matrix in this case. rank (A) ➔ returns the rank of the matrix A.
Matrix Powers	B = A.^2 ➔ squares each element in the matrix C = A * A ➔ computes A*A, and A must be a square matrix.
Determinant	det (A), and A must be a square matrix.

A, B, C are matrices, and m, n, α are scalars.

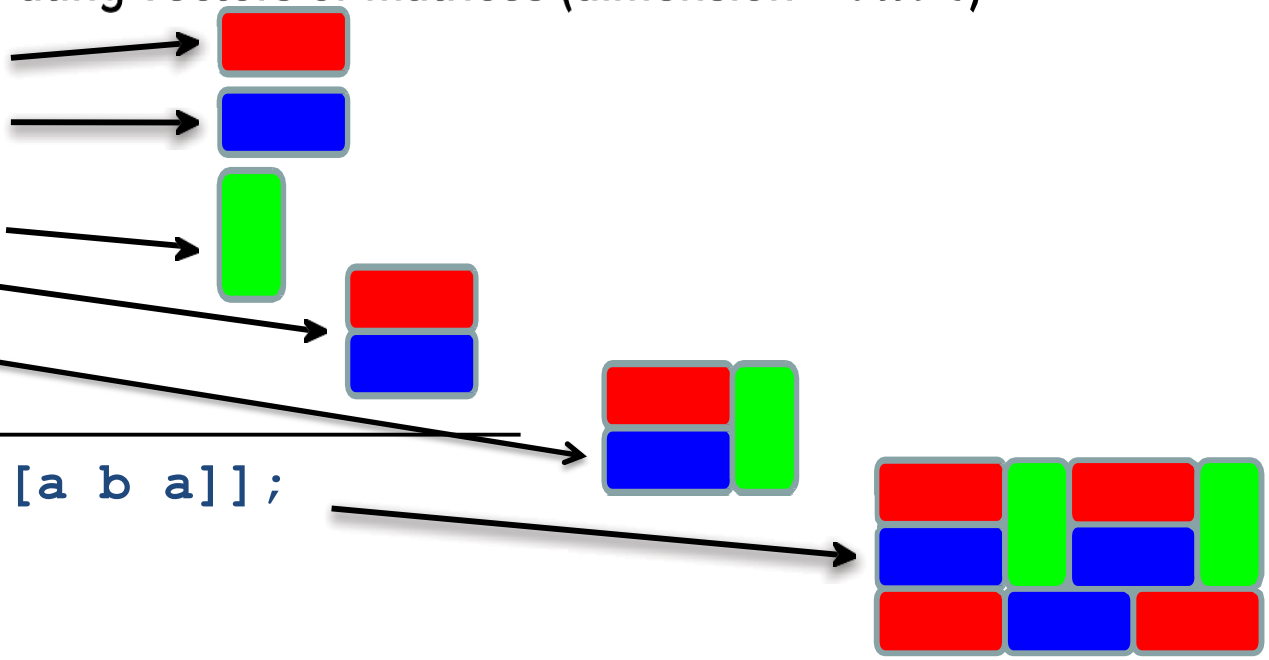
Matrices

- Make matrices like vectors
  - Element by element
    - `a= [1 2;3 4];`

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- By concatenating vectors or matrices (dimension matters)

```
a = [1 2];
b = [3 4];
c = [5;6]
d = [a;b];
e = [d c];
f = [[e e];[a b a]];
```



```
str = ['Hello I am '];
```

Strings are character vectors

Entering Numeric Arrays

Row separator:
semicolon (;)

Column separator:
space / comma (,)

```
»  a=[1 2;3 4]
a =
     1     2
     3     4
»  b=[-2.8, sqrt(-7), (3+5+6)*3/4]
b =
   -2.8000   0 + 2.6458i   10.5000
»  b(2,5) = 23
b =
   -2.8000  0 + 2.6458i  10.5000   0        0
         0         0        0   0  23.0000
```

Use square brackets [ ]

Matrices must be rectangular. (Set undefined elements to zero)

Any MATLAB expression can be entered as a matrix element

Entering Numeric Arrays

Scalar expansion

Creating sequences:
colon operator (:)

Utility functions for creating matrices.
(Ref: Utility Commands)

```
»   w=[1 2;3 4] + 5
w =
    6    7
    8    9
»   x = 1:5
x =
    1    2    3    4    5
»   y = 2:-0.5:0
y =
  2.0000   1.5000   1.0000   0.5000   0
»   z = rand(2,4)
z =
  0.9501   0.6068   0.8913   0.4565
  0.2311   0.4860   0.7621   0.0185
```

Numerical Array Concatenation

Use [ ] to combine existing arrays as matrix "elements"

Row separator:
semicolon (;)

Column separator:
space / comma (,)

The resulting matrix must be rectangular.

```
»  a=[1 2;3 4]
a =
     1     2
     3     4
»  cat_a=[a, 2*a; 3*a, 4*a; 5*a, 6*a]
cat_a =
     1     2     2     4
     3     4     6     8
     3     6     4     8
     9    12    12    16
     5    10     6    12
    15    20    18    24
```

Use square brackets [ ]

4*a

Size and length

- You can tell the difference between a row and a column by:
  - Looking in the workspace
  - Displaying the variable in the command window
  - Using the size function

```
>> size(row)

ans =

     1     4
```

```
>> size(column)

ans =

     4     1
```

sz = size(A) returns a row vector whose elements are the lengths of the corresponding dimensions of A. For example, if A is a 3-by-4 matrix, then size(A) returns the vector [3 4].

```
>> length(row)

ans =

     4
```

```
>> length(column)

ans =

     4
```

Size and length

L = **length(X)** returns the length of the largest array dimension in X.

- For vectors, the length is simply the number of elements.
- For arrays with more dimensions, the length is **max(size(X)).**
- The length of an empty array is zero.

```
>>v = 5:10
v = 1×6
     5     6     7     8
9    10
L = length(v)
L = 6
```

```
X = zeros(3,7);
L = length(X)
L = 7
```

Referencing and access to elements of a vector

Access to the elements of a vector is done using the following syntax:

- parentheses ( ) are used for consultation

  <vectorName(position)>

  % position can be a simple number, a list of numbers (position vector)

- hooks [ ] for creation

Referencing and access to elements of a vector

```
V= 5  -1  13   -6  7

>> V(3)        % 3rd position
 ans =13
 >> V(2:4)     % from second position to fourth
 ans =  -1    13    -6

 >> V(4:-2:1)  % from the 4th position to the 1st with step = -2
 ans =   -6    -1

 >> V(3:end)     % from the 3rd position to the last
  ans =    13    -6    7

 >> V([1,3,4])   % 1st, 3rd and 4th position only
  ans =   5    13    -6
```

Referencing and access to elements of a vector

```
V= [5   -1        13    -6   7]
```

```
>> V(1) = 8      % give the value 8 to the first element
```

```
>> V(6) = -3     % add a sixth element with value -3
```

```
>> V(2) = []     % delete the second element
```

```
>> V(3:5) = []   % delete from 3rd to 5th element
```

Array and Matrix Index

- Matrix indices begin from 1 (not 0!!!)
- Matrix indices must be positive integers

```
A =
     1     2     3
     4     5     6
     7     8     9
```

```
>> A(1,3)
ans =
     3
```

```
>> A(1)
ans =
     1
```

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

A(**1**) ↔ $A(\mathbf{1},1)$
A(**2**) ↔ $A(\mathbf{2},1)$

Column-Major Order

```
>> A(0)
Subscript indices must either be real positive integers or logicals.
>> A(1,4)
Index exceeds matrix dimensions.
>>
```

Matrix Index (example)

Given:

```
A =

     3     5     3
     6     8     2
     2     7     3
```

```
>> A(3,2)

ans =

     7
```

```
>> A(2,:)

ans =

     6     8     2
```

```
>> A(1:2,2)

ans =

     5
     8
```

A(-2), A(0)

Error: ??? Subscript indices must either be real positive integers or logicals.

A(4,2)
Error: ??? Index exceeds matrix dimensions.

Matrix Index (example)

```
A =
     1     2     3
     4     5     6
     7     8     9
```

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

ans =

     5     6
```

```
>> A(2,1:end)

ans =

     4     5     6
```

```
>> A(2,:)

ans =

     4     5     6
```

```
>> A(2,1:2:3)

ans =

     4     6
```

```
>> A(2,[1 3])

ans =

     4     6
```

```
>> A(:)

ans =

     1
     4
     7
     2
     5
     8
     3
     6
     9
```

Generating Vectors from functions

zeros(M,N) MxN matrix of zeros

```
x = zeros(1,3)
x =
     0     0     0
```

ones(M,N) MxN matrix of ones

```
x = ones(1,3)
x =
     1     1     1
```

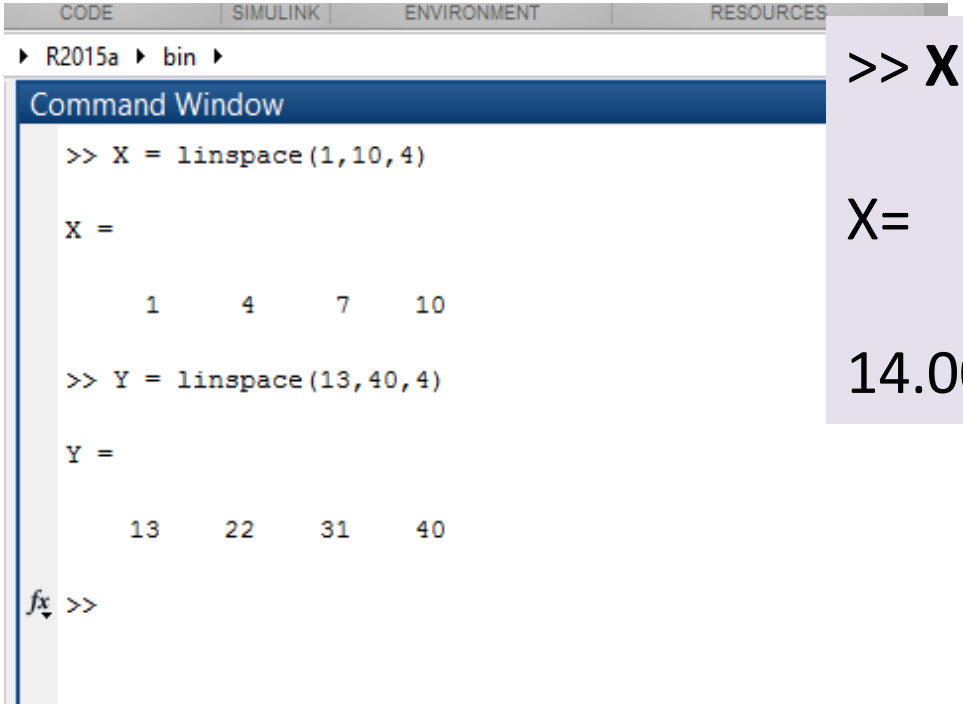
rand(M,N) MxN matrix of uniformly distributed random numbers on (0,1)

```
x = rand(1,3)
x =
  0.9501   0.2311   0.6068
```

Generating Vectors from functions

```
>> X = linspace(1,10,4) % a vector of four elements from 1 to 10
```

```
>> Y = linspace(13,40,4)     % a vector of four elements from 13 to 40
```




```
>> X = linspace(14,10,4)

X=

14.0000   12.6667   11.3333   10.0000
```

Matrix Concatenation

```
X=[1  2], Y=[3  4]

>> A=[X  Y]

     A=   1  2  3  4

>> A=[X;Y]

     1   2
     3   4
```

```
>> x=[1  2],y=[4  5], z=[  0  0]

>> A=[ x  y]

      1    2    4    5

 >> B  =  [x  ;  y]

         1  2
         4  5

  >>   C=[X   Y;  Z]
```

```
C = [x y ;z]
Error:
??? Error using ==> vertcat CAT arguments dimensions are not consistent.
```

Numerical Array Concatenation - [ ]

Use [ ] to combine existing arrays as matrix “elements”

Row separator:
semicolon (;)

Column separator:
space / comma (,)

The resulting matrix must be rectangular.

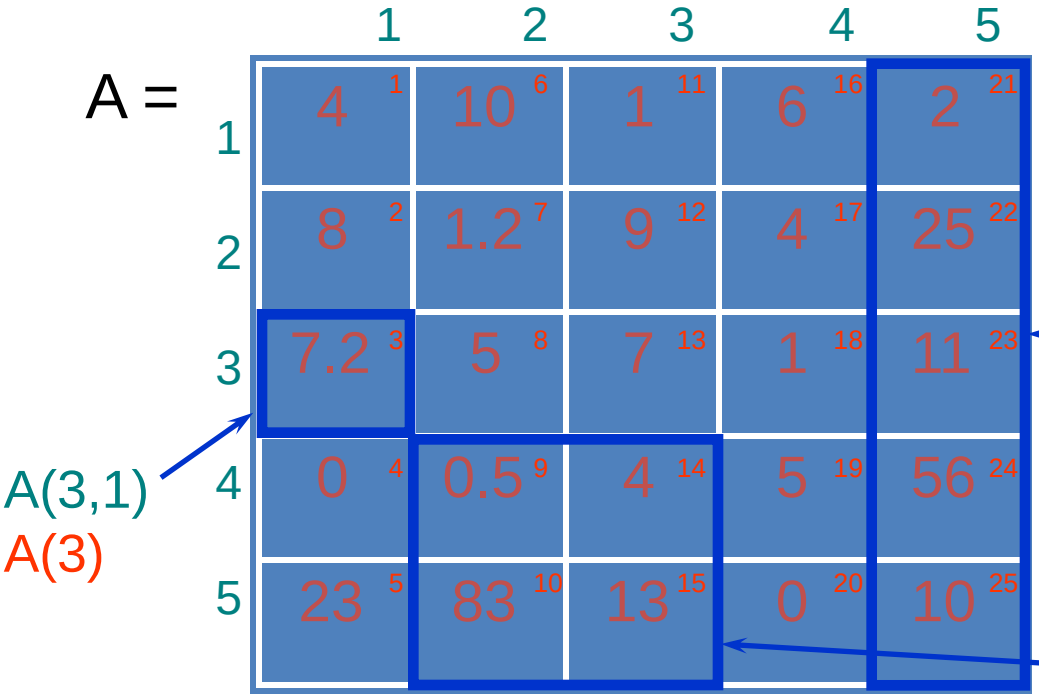
```
»   a=[1 2;3 4]
a =
    1    2
    3    4
»   cat_a=[a, 2*a; 3*a, 4*a; 5*a, 6*a]
cat_a =
    1    2    2    4
    3    4    6    8
    3    6    4    8
    9   12   12   16
    5   10    6   12
   15   20   18   24
```

Use square brackets [ ]

4*a

»num_cat

Array Subscripting / Indexing



- Use () parentheses to specify index
- colon operator (:) specifies range / ALL
- [ ] to create matrix of index subscripts
- ‘end’ specifies maximum index value

Exercise: Creating Matrices

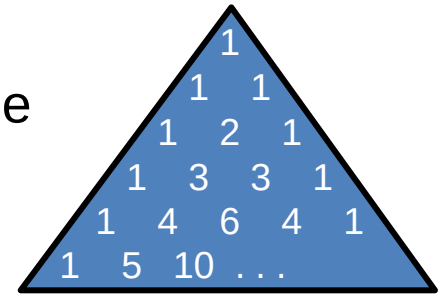
Create a 5x5 Pascal matrix ("P_mat")

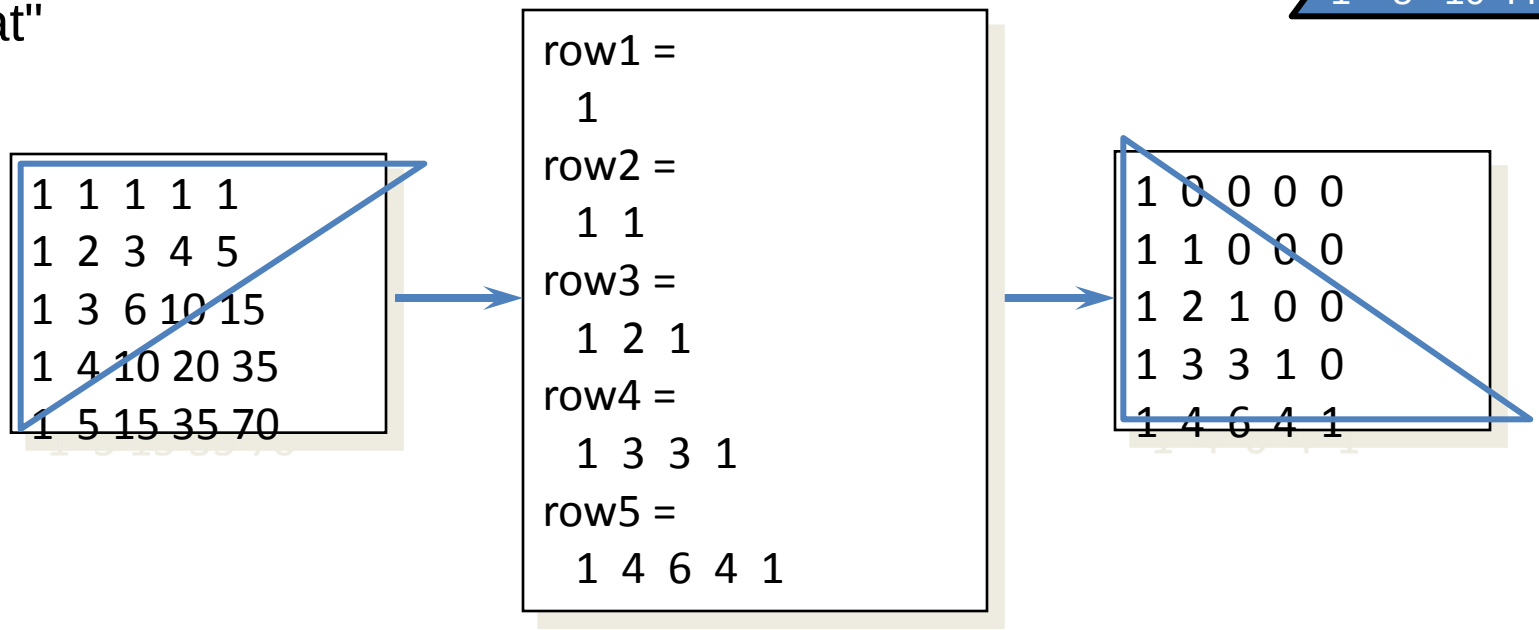
Extract elements to create row vectors for the first 5 rows of Pascal's Triangle

(Look for patterns in the element locations in "P_mat")

Combine the vectors into a single matrix with zeros above the diagonal

"P_mat"





Solution: Creating Matrices

```
» P_mat = pascal(5)                % Create Pascal's Matrix

» row1 = P_mat(1)                  % Extract Rows
» row2 = P_mat(2:4:6) .....
» row5 = P_mat(5:4:21)

» new_mat = row1                   % Create New Matrix
» new_mat(2,1:2) = row2 .....
» new_mat(5,1:5) = row5
» % NOTE: GENERALIZED FORM:
» % Nmax = length(P_mat);
» % rowN = P_mat(N:Nmax-1:(N-1)*Nmax+1)
» % new_mat(N,1:N) = rowN
```

Solution: Creating Matrices

```
» P_mat = pascal(5)                    % Create Pascal's Matrix

%  Rearrange Matrix - Pascal Triangle in upper triangle
%  Flip Matrix Horizontally (LEFT-RIGHT):
%  Could also use FLIPLR():
» temp = P_mat(1:5, 5:-1:1)

% Extract diagonals (rows of Pascal's Triangle)
» row1 = diag(temp,4)'
» row2 = diag(temp,3)'
» row3 = diag(temp,2)'
» row4 = diag(temp,1)'
» row5 = diag(temp)'
% Create New Matrix as before
```

Solution: Creating Matrices

```
»  P_mat = pascal(5)                % Create Pascal's Matrix

% Rearrange Matrix - Pascal Triangle in lower triangle
%  Flip Matrix Vertically (UP-DOWN):
%  Could also use FLIPUD():
»  temp1 = P_mat(5:-1:1, 1:5)

% Extract lower triangular matrix:
»  temp2 = tril(temp1)

% Create New Matrix by sorting columns of "temp2":
»  new_mat = sort(temp2)
```

Matrices Operations

Add: +
Subtract: -

```
>> u = [-2, 6, 1] ;
>> v = [ 3, -1, 4] ;
```

```
>> u+2
Ans =
      0   8   3

>> u+v
ans =

       1     5      5
```

```
>> u-2
Ans =
      -4   4   -1

>> u-v
ans =

       -5     7      -3
```

Matrices Operations

Given A and B:

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

A =

     1     2     3
     4     5     6
     7     8     9
```

```
>> B = [3 5 2; 5 2 8; 3 6 9]

B =

     3     5     2
     5     2     8
     3     6     9
```

Addition

```
>> X = A + B

X =

     4     7     5
     9     7    14
    10    14    18
```

Subtraction

```
>> Y = A - B

Y =

    -2    -3     1
    -1     3    -2
     4     2     0
```

Product

```
>> Z = A * B

Z =

    22    27    45
    55    66   102
    88   105   159
```

Transpose

```
>> T = A'

T =

     1     4     7
     2     5     8
     3     6     9
```

Matrices Operations

Divide: **./** % element by element

Multiply: **.*** % element by element

Power: **.^** (e.g. .^2 means squared) % element by element

You can use round brackets to specify the order in which operations will be performed

Matrices Operations

```
>> u = [-2, 6, 1] ;
>> v = [ 3, -1, 4] ;
```

```
>> u*2
ans =
      -4   12   2

>> u.*2
ans =

       -4     12      2

>> u.*v
 ans=
         -6     -6    4
```

```
>> u/2
  ans =
  -1.0000   3.0000 0.5000
>> u./2
  ans =
  -1.0000   3.0000 0.5000
>> u./v
  ans =
  -0.6667   -6.0000      0.2500
```

```
>> u.^2
ans =
   4   36       1
>> u.^v
ans =
  -8.0000   0.1667     1.0000
```

Matrices Operations

```
A = [1 2 3; 5 1 4; 3 2 1]
    A =
          1    2    3
          5    1    4
          3    2   -1
```

```
x = A(1,:)

x=
     1   2   3
```

```
y = A(3 ,:)

y=
     3  2  -1
```

```
b = x .* y

b=
     3  4 -3
```

```
c = x . / y

c=
   0.33   0.5   -3
```

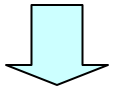
```
d = x .^2

d=
     1    4    9
```

```
K= x^2
Erorr:
 ??? Error using ==> mpower  Matrix must be square.
B=x*y
Erorr:
??? Error using ==> mtimes Inner matrix dimensions must agree.
```

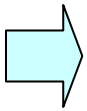
Matrices Operations

```
A = [1 2 3; 5 1 4; 3 2 1]
    A =
          1     2     3
          5     1     4
          3     2    -1
```



```
x = A(1,:)        y = A(3 ,:)

x=                y=
    1   2   3         3  4  -1
```



```
b = x .* y

b=
     3  8 -3
```

```
c = x . / y

c=
   0.33   0.5   -3
```

```
d = x
.^y

d=
       1    16    0.33
```

Logical functions

- ✓any (vecA) returns 1 if one of the elements of vecA is not null
- ✓all (vecA) returns 1 if all elements of vecA are null

- ✓isequal (x,y) retourns 1 if x and y are equal
- ✓ischar (x) retourns 1 if x is a character

...

Special function

Commande	Signification
Sum(V)	La somme des éléments d'un vecteur V
prod (V)	Le produit des éléments d'un vecteur V
mean (V)	La moyenne des éléments d'un vecteur V
sort (V)	Ordonne les éléments du vecteur V par ordre croissant
length(V)	la taille du vecteur V
max(V)	plus grand élément du vecteur V
min(V)	Plus petit élément du vecteur V
fliplr(V)	renverse l'ordre des éléments du vecteur V

Applications (1)

Questions	
Create the line vector in two ways (9 7 5 3 1).	X=[9 7 5 3 1]; X=9:-2:1;
Create the column vector in two ways (10 9.5 9 8.5 8).	Y=[10; 9.5; 9; 8.5; 8]; Y=(10:-0.5:8)’;
Create the line vector (9 7 5 3 1 9 7 5 3 1 9 7 5 3 1) from the previous vectors.	X3=[X X X];
Create the column vector (10 9.5 9 8.5 8 10 9.5 9 8.5 8 10 9.5 9 8.5 8) from the previous vectors.	Y3=[Y;Y;Y];

Applications (2)

Questions
Enter polynomials p = $2x^5 - x^2+5x$ et q = x^2+1.
Calculate the roots of p and q.
Define the vector V containing 50 values ranging from 1 to 3.
Evaluate the polynomial P at the points in V.

```
p=zeros(1,6);
q=zeros(1,3);
 p(1)=2; p(4)=-1; p(5)=5;
q(1)=1; q(3)=1;
  roots(p) ;
   roots(q) ;

 V=linspace(1,3,50) ;

  Polyval(p,v) ;
```

Applications (3)

Compute :

$$\sum_{n=0}^{100} \sin^n x$$

For $x=\frac{\pi}{5}$

Solution :

1- create a vector pow with integer from 0 to 100,

2- create a second vector t each element is **sin ∏/5 power n**.

3- calculate sum of t' elements

```
>> pow = 0 : 100 ;
>> t = sin ( pi /5) .^ pow ;
>> s = sum( t )
```

Applications (4)

1. Construct the vector $(1, \frac{1}{2}, \frac{1}{3} \ldots\ldots, \frac{1}{1000})$
2. Deduce the vector from it .
3. Use this final vector for calculations $\sum_{k=1}^{1000} \frac{1}{k^2}$

Solution :

1. Créating inverse of number vector from 1 to 1000

```
>> u = 1. / (1:1000) ;
```

2. Élévation square vector element :

```
>> v = u.^2 ;
```

3. function sum(v) sum vector éléments.

```
>>somme = sum( v )
```