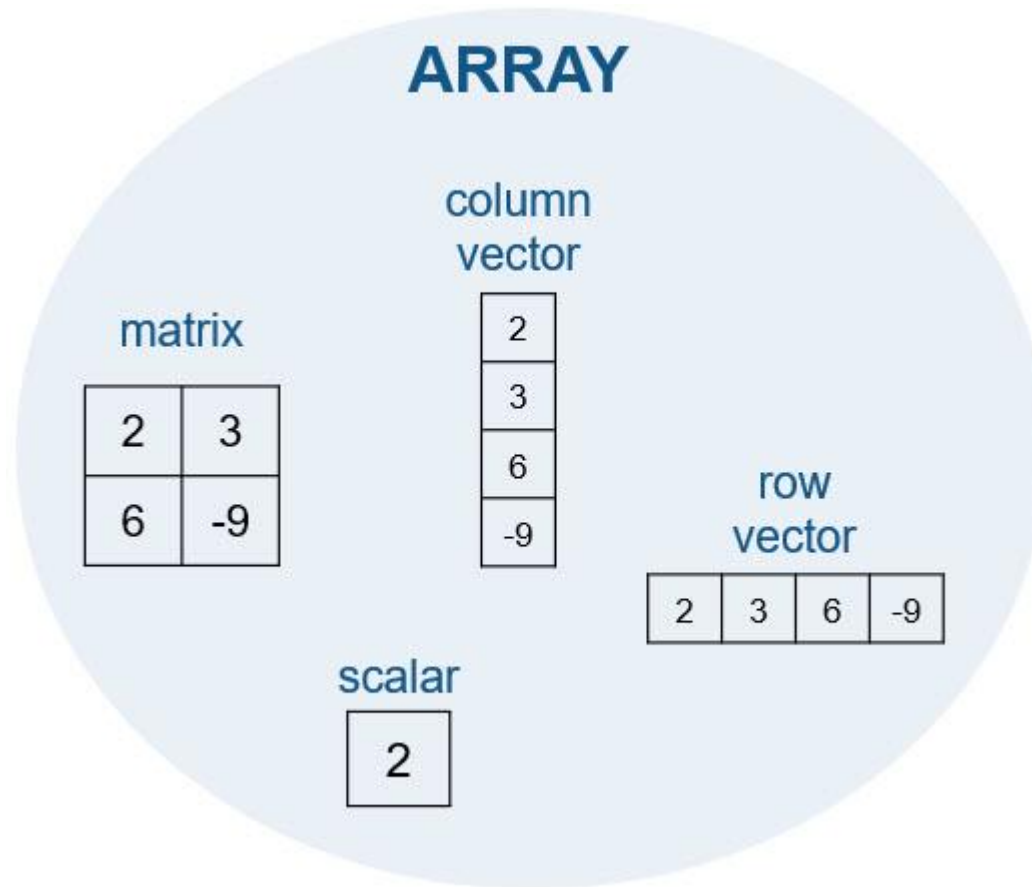


# Matrices and Arrays

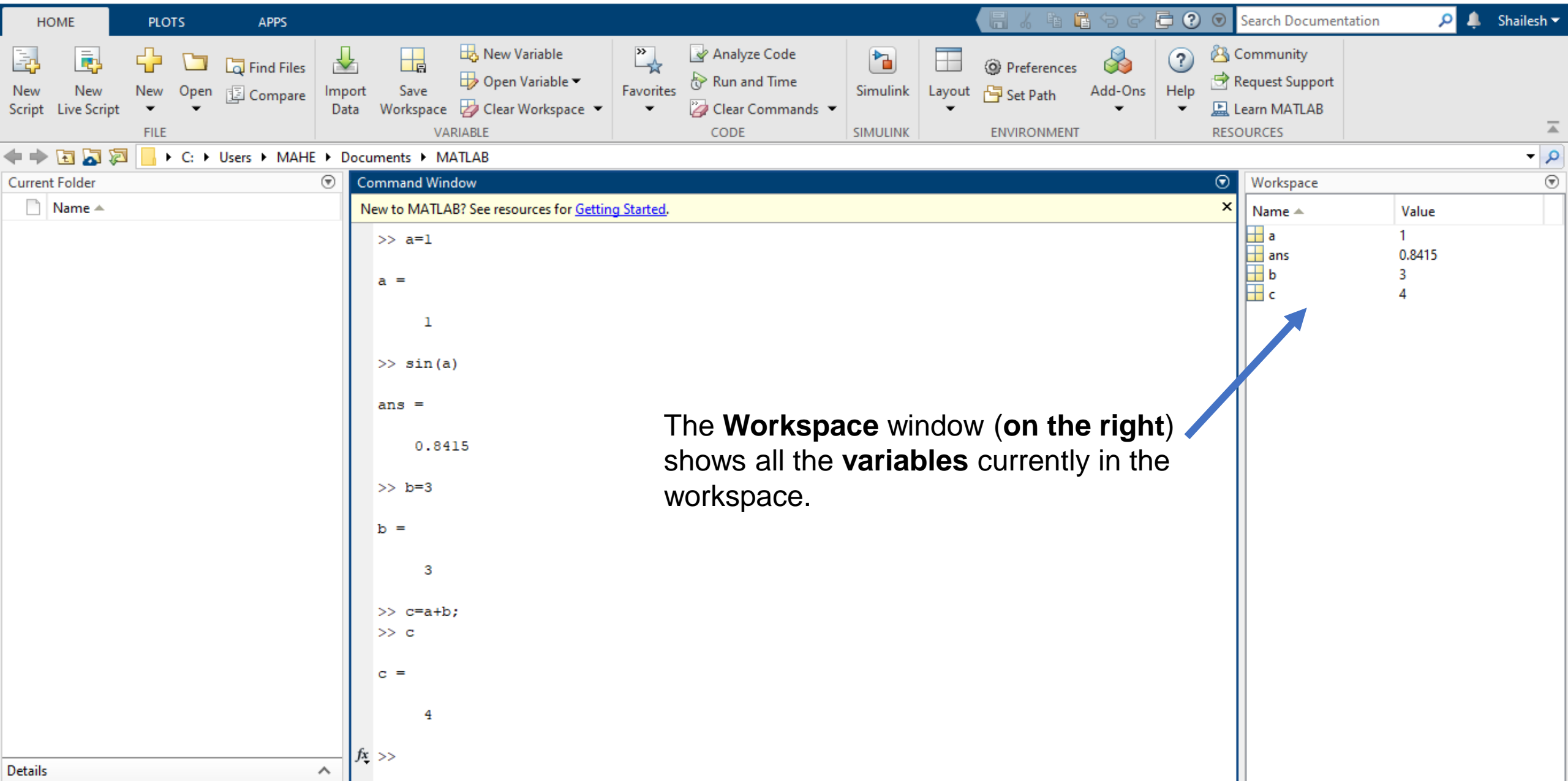


# Some useful commands

- **clc**; Clear command window. (**home**)
- **clear**; Clear variables and functions from memory.
- **format**; Set output format. **Short** (4 digits) **Long** (15 digits)
- Try : **help disp**;

# Creating variable at the command line

- Create a variable named **'a'** by typing this statement at the command line (`>>`):
- `>> a = 1 ;`
- When you do not specify an output variable, MATLAB uses the variable **ans**, short for answer, to store the results of your calculation.
- `>> sin(a)`
- `ans =`
- `0.8415`



HOME PLOTS APPS

Search Documentation Shailish

New Script New Live Script New Open Find Files Compare Import Data Save Workspace New Variable Open Variable Clear Workspace Favorites Analyze Code Run and Time Clear Commands Simulink Layout Preferences Set Path Add-Ons Help Community Request Support Learn MATLAB

FILE VARIABLE CODE SIMULINK ENVIRONMENT RESOURCES

Current Folder C: \ Users \ MAHE \ Documents \ MATLAB

Command Window

New to MATLAB? See resources for [Getting Started.](#)

```
>> a=1  
  
a =  
  
    1  
  
>> sin(a)  
  
ans =  
  
    0.8415  
  
>> b=3  
  
b =  
  
    3  
  
>> c=a+b;  
>> c  
  
c =  
  
    4  
  
fx >>
```

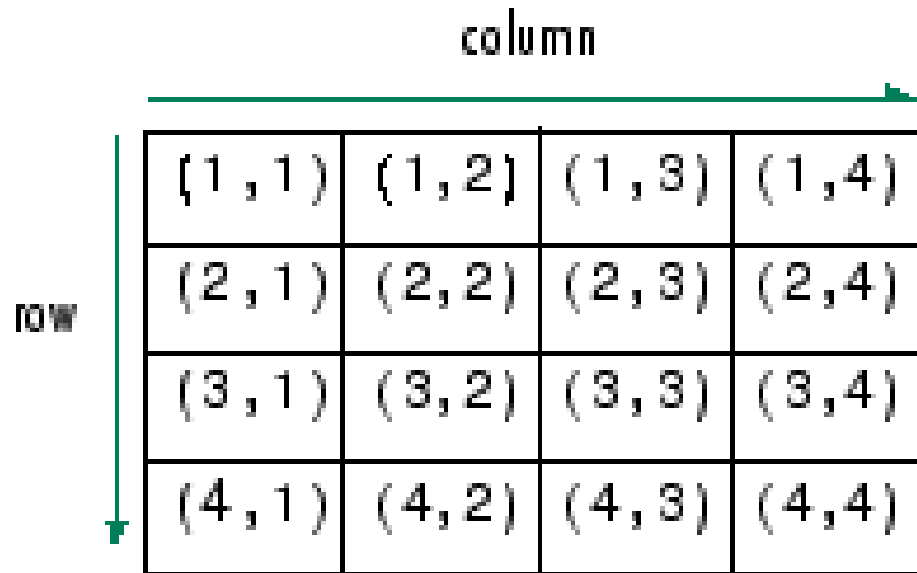
Workspace

Name	Value
a	1
ans	0.8415
b	3
c	4

The **Workspace** window (on the right) shows all the **variables** currently in the workspace.

# Matrices and Arrays

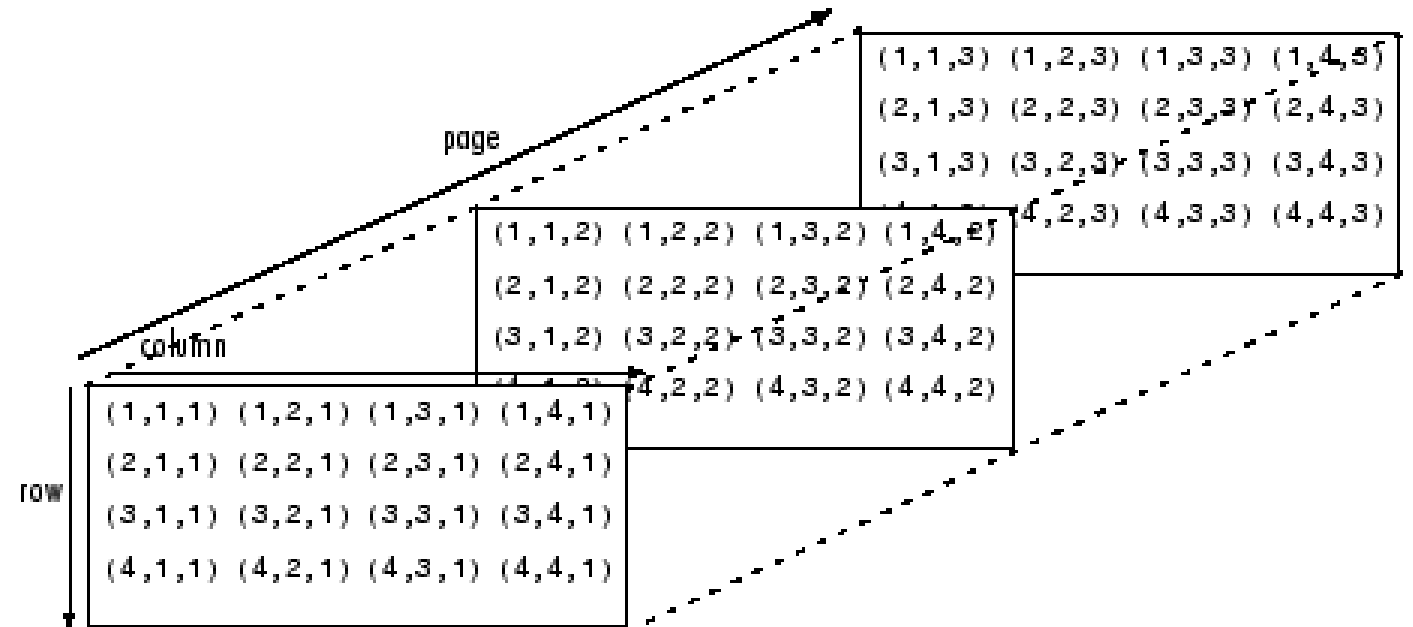
- All MATLAB variables are multidimensional arrays, no matter what type of data.



A 4x4 grid representing a 2D array. A green arrow on the left points downwards and is labeled 'row'. A green arrow on the top points to the right and is labeled 'column'.

{ 1 , 1 }	{ 1 , 2 }	{ 1 , 3 }	{ 1 , 4 }
{ 2 , 1 }	{ 2 , 2 }	{ 2 , 3 }	{ 2 , 4 }
{ 3 , 1 }	{ 3 , 2 }	{ 3 , 3 }	{ 3 , 4 }
{ 4 , 1 }	{ 4 , 2 }	{ 4 , 3 }	{ 4 , 4 }

2D – Array



A 3D representation of an array with three stacked 4x4 grids. A solid arrow on the left points downwards and is labeled 'row'. A solid arrow on the top points to the right and is labeled 'column'. A dashed arrow points from the front grid to the back grid and is labeled 'page'.

{ 1 , 1 , 1 }	{ 1 , 2 , 1 }	{ 1 , 3 , 1 }	{ 1 , 4 , 1 }
{ 2 , 1 , 1 }	{ 2 , 2 , 1 }	{ 2 , 3 , 1 }	{ 2 , 4 , 1 }
{ 3 , 1 , 1 }	{ 3 , 2 , 1 }	{ 3 , 3 , 1 }	{ 3 , 4 , 1 }
{ 4 , 1 , 1 }	{ 4 , 2 , 1 }	{ 4 , 3 , 1 }	{ 4 , 4 , 1 }

{ 1 , 1 , 2 }	{ 1 , 2 , 2 }	{ 1 , 3 , 2 }	{ 1 , 4 , 2 }
{ 2 , 1 , 2 }	{ 2 , 2 , 2 }	{ 2 , 3 , 2 }	{ 2 , 4 , 2 }
{ 3 , 1 , 2 }	{ 3 , 2 , 2 }	{ 3 , 3 , 2 }	{ 3 , 4 , 2 }
{ 4 , 1 , 2 }	{ 4 , 2 , 2 }	{ 4 , 3 , 2 }	{ 4 , 4 , 2 }

{ 1 , 1 , 3 }	{ 1 , 2 , 3 }	{ 1 , 3 , 3 }	{ 1 , 4 , 3 }
{ 2 , 1 , 3 }	{ 2 , 2 , 3 }	{ 2 , 3 , 3 }	{ 2 , 4 , 3 }
{ 3 , 1 , 3 }	{ 3 , 2 , 3 }	{ 3 , 3 , 3 }	{ 3 , 4 , 3 }
{ 4 , 1 , 3 }	{ 4 , 2 , 3 }	{ 4 , 3 , 3 }	{ 4 , 4 , 3 }

3D – Array

Array Creation : `>> a = [1 2 3 4]` ; creating a **row** vector

`c = [3+4i, 4+3j; -i, 10j]` ; Array with **complex** numbers

# Examples (Try it on MATLAB command window)

- Creating a matrix that has multiple rows, separate the rows with semicolons.
- `>> a = [1 3 5; 2 4 6; 7 8 10]`
- Functions : **ones, zeros, eye**
- `z = zeros(5,1) ;` *%create a 5-by-1 column vector of zeros.*
- `O=ones(1,5);` *%create a 1-by-5 row vector of ones.*

# Matrix and Array Operations

- MATLAB allows you to process all the values in a matrix using a single arithmetic operator or function.
- `>> O = O+10;`
- `>> sin(a);`
- To transpose (**flip – interchanging rows and columns**) a matrix, use a single quote (`'`):
- Matrix multiplication : `a*a`

# Dot operator (.)

- Given : `>> a=[1 2 3;4 5 6; 7 8 9];`
- Try : `>> a.*a`
- To perform element-wise multiplication rather than matrix multiplication, use the `.*` operator.
- Try : `>> a./3` ; AND `>> a.^2;`



# Concatenation (append or join)

- $A=[a,a]$  ; concatenating 'a' with 'a' horizontally.
- $C = \text{horzcat}(a,a);$
- $A=[a;a]$  ; concatenating 'a' with 'a' vertically.
- $C = \text{vertcat}(a,a);$
- The pair of square brackets **[ ]** is the concatenation operator.

# Saving and Loading Workspace Variables

- Save variables in your workspace to a MATLAB specific file format called a MAT-file using the save command.
- To save the workspace to a MAT-file named **filename.mat**, use the command:
- `>> save filename`
- `>> load filename`
- Saving the variable **"k"** to a new MAT-file called **justk.mat**:
- `>> save justk k`

# Using Built-in Functions and Constants

- `>> a = pi` ( $\pi$ )
- <https://in.mathworks.com/help/phased/ref/physconst.html>
- `>> a = sin(30)`
- `>> a = sin(pi/6)`
- `>> a = sqrt(16)`
- You can perform calculations within the square brackets.
- `x = [abs(-4) 4^2];`
- `y = [sqrt(10) pi^2]`

# Creating Evenly-Spaced Vectors

- $X = [1\ 2\ 3\ 4]$  – Vector contains evenly spaced number
- Shortcut method -  $X = 1:4$  – “**:**” operator – **start\_value : end\_value**
- $A = 1:2:10$  – **start\_value : increment : end\_value**
- `linspace(start_value , end_value, number_of_elements)`
- **If you wanted to create an evenly-spaced vector from 1 to  $2\pi$  with 100 elements, would you use linspace or `:`?**

# Array Creation Functions

- `x = rand(2)` - output will be a 2-by-2 matrix of random numbers.
- `X = ones(1,3)` or `X = zeros (2,3)`
- `X = rand(size(X));`

# Array Indexing – Single element

- Indexing is used to access selected elements of an array
- $A = [1 \ 2 \ 3 \ 4; 5 \ 6 \ 7 \ 8; 9 \ 10 \ 11 \ 12; 13 \ 14 \ 15 \ 16];$
- $A(\text{row}, \text{col})$
- Single subscript traverses down each column in order –  $A(4,2) = A(8)$
- What is the output for  $A(4,5)$ ?
- **You can specify elements outside the current dimensions.**
- **The size of the array increases to accommodate the newcomers.**

# Array Indexing – Multiple elements

- To refer to multiple elements of an array, colon operator is used, which allows you to specify a range of the form **start : end**.
- Try : `A(1:3,2)`
- Try : `A(3,:)`
- Try : `A(2,2:end)`
- Example : Extract the Third, Sixth, and Eighth elements from A.

# Indexing Vectors

- `A = [16 5 9 4 2 11 7 14];`
- Swap the two halves of `v` to make a new vector:
- `A2 = A([5:8 1:4])`
- Extracting portions of an array: `A(2:end-1)`
- `A(1:2:end)` % Extract all the odd elements
- `A(end:-1:1)` % Reverse the order of elements



# Indexing Vectors

- `A = [16 5 9 4 2 11 7 14];`
- `A([2 3 4]) = [10 15 20]` % Replace some elements of A
- `A([2 3]) = 30` % Replace second and third elements by 30

# Indexing Matrices with Two Subscripts

- $A(\text{row\_x}, \text{col\_y})$  - Extract the element in row\_x, column\_y
- $A(\text{row\_n}, :)$  – Extract nth row
- $A(:, \text{end})$  – Extract last column
- $A(2:4, 1:2)$

# Linear Indexing

- When you index into the matrix A using only one subscript,
- MATLAB treats A as if its elements were strung out in a long column vector, by going down the columns consecutively.
- $A(14) = A(2,4)$
- $A([6 \ 12 \ 15]) = 11 \ 15 \ 12$

1 16	5 2	9 3	13 13
2 5	6 11	10 10	14 8
3 9	7 7	11 6	15 12
4 4	8 14	12 15	16 1

**A**

# Linear Indexing

- **sub2ind** that converts from row and column subscripts to linear indices.
- `idx = sub2ind(size(A), [2 3 4], [1 2 4])`
- `ans =`
- `2 7 16`
- `A(idx)`
- `ans =`
- `5 7 1`

1 16	5 2	9 3	13 13
2 5	6 11	10 10	14 8
3 9	7 7	11 6	15 12
4 4	8 14	12 15	16 1

**A**

# Logical Indexing

- $A(A > 12)$  extracts all the elements of A that are greater 12 into a logical array (**column vector**).
- Many MATLAB functions that start with **is** return logical arrays and are very useful for logical indexing.
- **$B(\text{isnan}(B)) = 0$**  - To replace all NaN elements of the matrix B with zero
- **$\text{Strvalue}(\text{isspace}(\text{Strvalue})) = \text{'\_'}$**  - replace all the spaces in a string matrix Strvalue with underscores.

# Find function

- Logical indexing is closely related to the **find** function.
- **FIND** returns a vector containing the linear indices of **each nonzero element** in **array** X.
- $A(A > 5)$  is equivalent to  $A(\text{find}(A > 5))$
- If X is a vector, then **find** returns a vector with the same orientation as X.
- If X is a multidimensional array, then **find** returns a **column vector** of the **linear indices** of the result.

# Find function

- $X =$ 

16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1
- $k = \text{find}(X < 10, 5)$  ; Find the first five elements that are less than 10
- $X(k)$  ; View the corresponding elements of X.
- SEE - **ind2sub** - Convert linear indices to subscripts

# Changing Values in Arrays

- $A(1,4)=0.5$
- $A(1,\text{end})=0.2$



# Performing Array Operations on Vectors

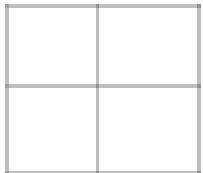
- Adding a **scalar value** to all the **elements** of an **array**.
- **Add/Sub** any two arrays of the same size.
- **Multiply** or **divide** all the elements of an array by a scalar.
- Basic statistical functions in MATLAB can be applied to a vector to produce a single output. **Max, Min, Sum, Sqrt**
- `.*` operator performs elementwise multiplication and allows to multiply the corresponding elements of two equally sized arrays.
- Ex :  $z = [3 \ 4] .* [10 \ 20] = ?$
- Try :  $x = [1 \ 2; 3 \ 4; 5 \ 6; 7 \ 8] .* [1; 2; 3; 4]$

# Compatible Array Sizes for Basic Operations

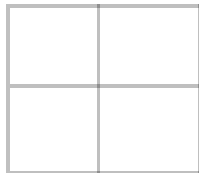
- Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are either the same or one of them is 1.
- In the simplest cases, two array sizes are compatible if they are the same or if one is a scalar.

## Two inputs which are the same size

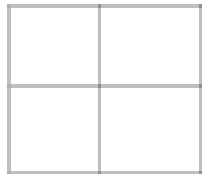
A: 2-by-2



B: 2-by-2

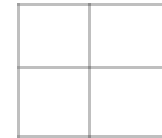


Result: 2-by-2



## One input is a scalar

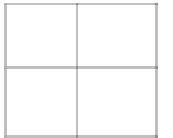
A: 2-by-2



B: 1-by-1

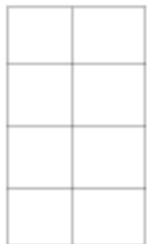


Result: 2-by-2

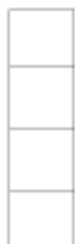


## Matrix and column vector with the same number of rows.

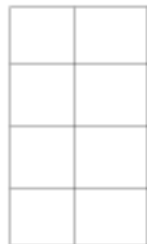
A: 4-by-2



B: 4-by-1



Result: 4-by-2



## Column vector, Row vector

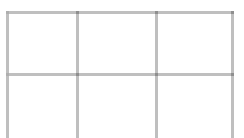
B: 2-by-1



A: 1-by-3



Result: 2-by-3



# Multidimensional Arrays

- Creating Multidimensional Arrays
- `>> A = [1 2 3; 4 5 6; 7 8 9];`
- Add a second page : `A(:,:,2) = [10 11 12; 13 14 15; 16 17 18]`
- `B = cat(3,A,[3 2 1; 0 9 8; 5 3 7])`
- The `cat` function can be a useful tool for building multidimensional arrays.
- Expand a multidimensional array `>> B(:,:,4) = 0`

# Accessing Elements

- Try :  $C = A(:, [1 \ 3], :)$ 
  - Use the index vector  $[1 \ 3]$  in the second dimension to access only the first and last columns of each page of A.
- Try :  $D = A(2:3, :, :)$ 
  - To find the second and third rows of each page, use the colon operator to create your index vector.

# Manipulating Arrays - Reshape

1	2	3	4	5
9	0	6	3	7
8	1	5	0	2

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

$B = \text{reshape}(A, [6 \ 5])$

Use the reshape function to rearrange the elements of the 3-D array into a 6-by-5 matrix.

$B = 6 \times 5$

1	3	5	7	5
9	6	7	5	5
8	5	2	9	3
2	4	9	8	2
0	3	3	8	1
1	0	6	4	3

**reshape** operates columnwise, creating the new matrix by taking consecutive elements down each column of A, starting with the first page then moving to the second page.

# Manipulating Arrays

- $P1 = \text{permute}(M, [2 \ 1 \ 3])$  - interchange row and column subscripts on each page
- $P2 = \text{permute}(M, [3 \ 2 \ 1])$  - interchange row and page subscripts of  $M$ .
- $M(:, :, 1) = [1 \ 2 \ 3; 4 \ 5 \ 6; 7 \ 8 \ 9];$
- $M(:, :, 2) = [0 \ 5 \ 4; 2 \ 7 \ 6; 9 \ 3 \ 1];$

# Repeat copies of array

- Create a 3-by-2 matrix whose elements contain the value 10.
- `>> A = repmat(10,3,2)`
- $A = 3 \times 2$
- $\begin{bmatrix} 10 & 10 \\ 10 & 10 \\ 10 & 10 \end{bmatrix}$

# Repeat copies of array

- **Example :**

- Try : `A = diag([100 200 300]);`

- Try : `B = repmat(A,2,3);`

- **Example :**

- Try : `A = [1 2; 3 4];`

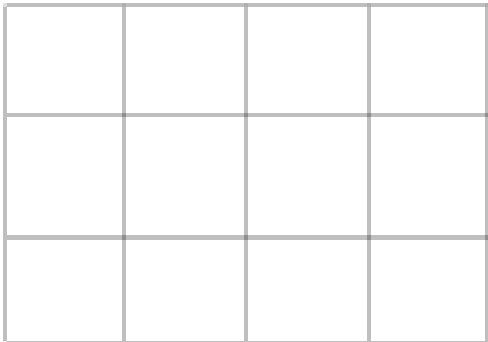
- Try : `B = repmat(A,[2 3 2]);`



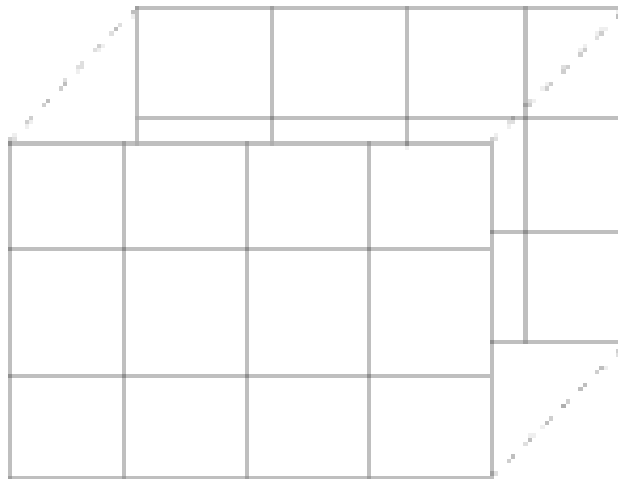
# Compatible Array Sizes for Basic Operations

- One input is a matrix, and the other is a 3-D array with the same number of rows and columns.

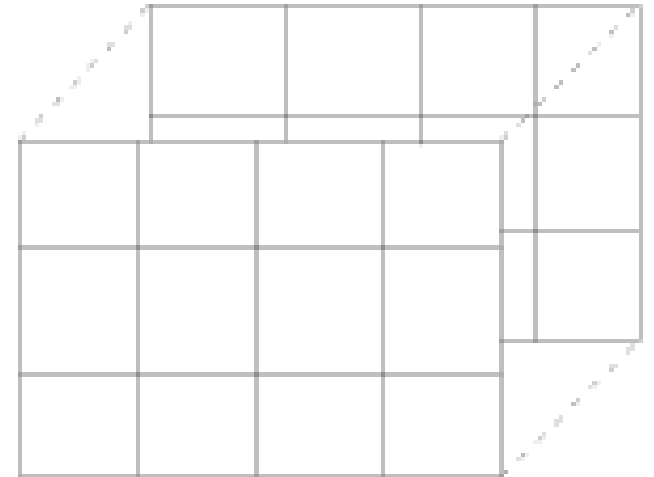
A: 3-by-4



B: 3-by-4-by-2



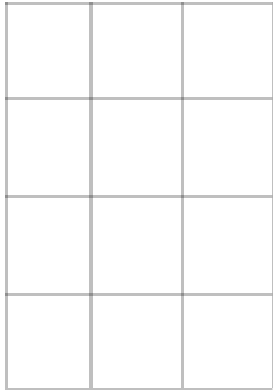
Result: 3-by-4-by-2



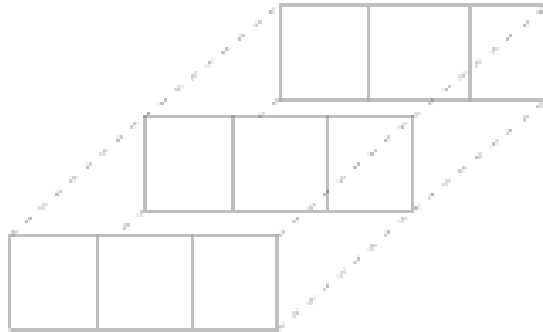
# Compatible Array Sizes for Basic Operations

- One input is a matrix, and the other is a 3-D array. The dimensions are all either the same or one of them is 1.

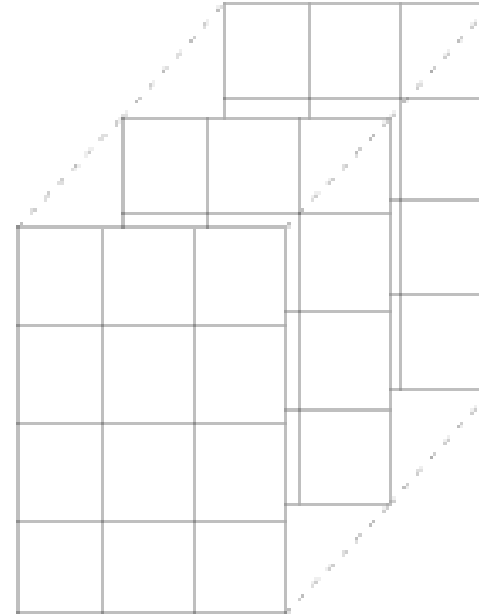
A: 4-by-3



B: 1-by-3-by-3



Result: 4-by-3-by-3



# Cell Array

- A cell array is a data type with **indexed data containers** called **cells**, where each cell can contain any type of data.
- **Cell** arrays commonly contain either **text**, **combinations of text and numbers**, or **numeric arrays** of different sizes.
- $C = \{ '2017-08-16', [56 \ 67 \ 78] \}$
- $C(1,3) = \{ 'Hello world' \}$  - Adding new element
- $C(2,:) = \{ '2017-08-17', [58 \ 69 \ 79], 'dd' \};$  - Adding a new row.

# Table Array

- Table array with named variables that can contain different types.
  - column-oriented data in variables
- LastName = {'Sanchez';'Johnson';'Li';'Diaz';'Brown'};
- Age = [38;43;38;40;49];
- Employed = logical([1;0;1;0;1]);
- Height = [71;69;64;67;64];
- Weight = [176;163;131;133;119];
- BloodPressure = [124 93; 109 77; 125 83; 117 75; 122 80];
- **T = table(LastName, Age, Employed, Height, Weight, BloodPressure);**

# Table Array

- `meanHeight = mean(T.Height);` - **Performing calculations**
- `T.BMI = (T.Weight*0.453592)./(T.Height*0.0254).^2` – **Adding a new column BMI**
- `T.Properties.Description = 'Patient data, including body mass index (BMI) calculated using Height and Weight';`

# Structure Array

- A structure array is a data type that groups related data using data containers called fields.
  - Each field can contain any type of data.
- Access data in a field using dot notation of the form **structName.fieldName**.
- Rectangle.L = 10;
- Rectangle.W = 5;
- Rectangle.Area = Rectangle.L \* Rectangle.W
- **s = struct(obj)** creates a scalar structure with field names and values that correspond to properties of obj.