Introduction to MATLAB bootcamp

Week 1 Lecture 2

Polina Cherepanova cherepanova@princeton.edu
Polina Iamshchinina pi7127@princeton.edu

Rules for naming a variable

- A valid variable name starts with a letter, followed by letters, digits, or underscores. e.g. variable_name, var1,
- MATLAB® is case sensitive. So Var_1 and var_1 are different variables
- Variable name cannot start with a number
- Variable name cannot have special characters (like !, $, &, ^, %)
- Cannot use spaces
- The maximum length of a variable name is the value that the ”namelengthmax” command returns. Test this command and find what is the maximum number of characters which you can have in variable name
- Cannot use keywords (try iskeyword in command window)

Classify which of these is an acceptable MATLAB variable name

variablename

variable_name

variable name

variable_name$

variableName

for

variable_name_1

1_variable_name

MATLAB variable naming guidelines

1) Name should be descriptive
- Bad e.g. : a, var, x, y
- Good e.g. : length_vector, stimulus_threshold

2) By convention, start with lowercase
- Bad e.g. : Length_vector, Stimulusthreshold
- Good e.g. : length_vector

2) Use capitalization or underscores for readability
- Bad: stimulusabovethreshold
- Good: stimulus_above_threshold, or stimulusAboveThreshold

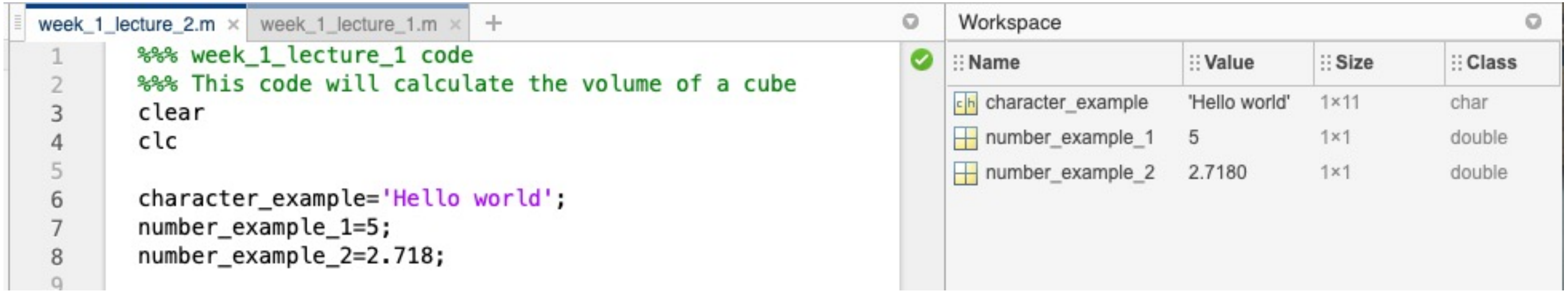
Modified from: https://github.com/mlnguyen/pni_summer_matlab2020

Variable types

- Great thing about MATLAB is that we do not need to initialize variables

- Variable types:
  - Numbers e.g. number_example=5;
  - Characters e.g. character_example='Hello world’;
  - Collection of numbers e.g. array_example=[1, 2, 3];
  - Collection of numbers and strings

Variable types



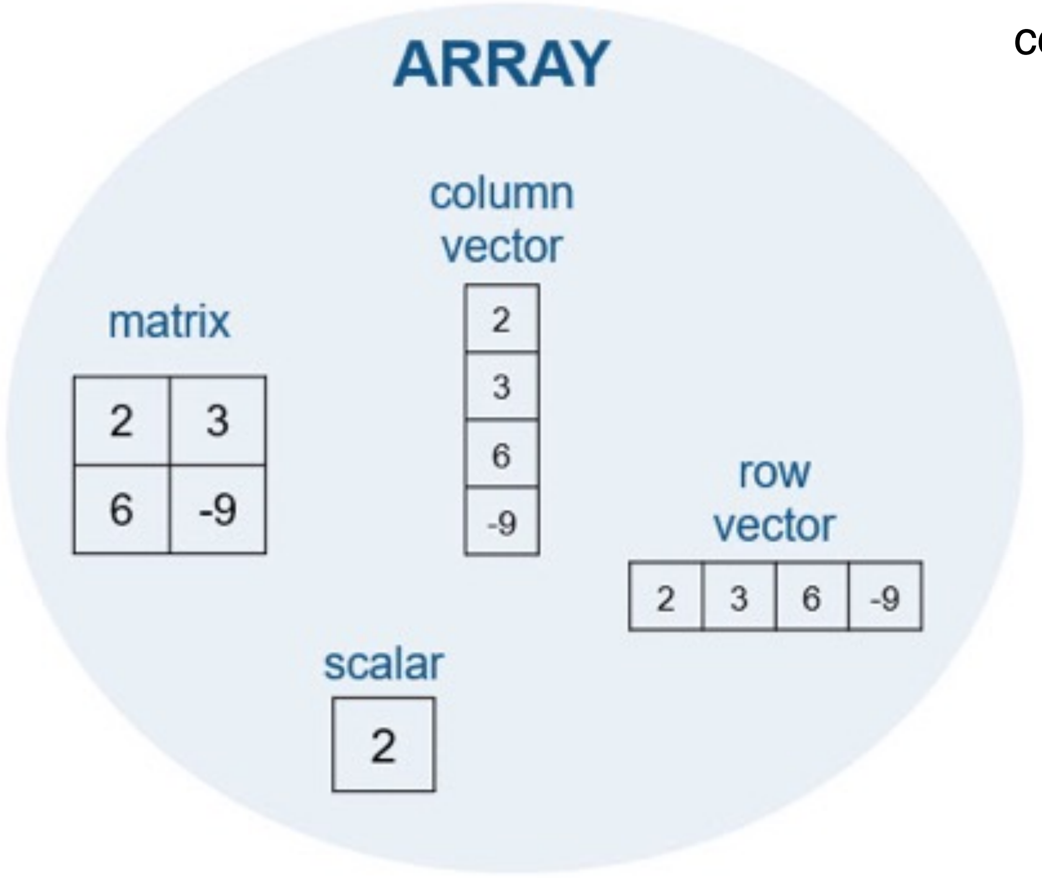
```
%%% week_1_lecture_1 code
%%% This code will calculate the volume of a cube
clear
clc

character_example='Hello world';
number_example_1=5;
number_example_2=2.718;
```

Name	Value	Size	Class
character_example	'Hello world'	1×11	char
number_example_1	5	1×1	double
number_example_2	2.7180	1×1	double

Arrays

- Very very very helpful and powerful!

```
row_vector=[11 12 13 14 15 16];
col_vector=[11; 12; 13; 14; 15; 16];
```



Selecting an element by index in a row/column vector

```
row_vector=[11 12 13 14 15 16];
```

```
>> row_vector(2)

ans =

12
```

```
>> row_vector(1:4)

ans =

11 12 13 14
```

```
>> row_vector(1:2:6)

ans =

11 13 15
```

Selecting an element by index in a row/column vector

```
row_vector=[11 12 13 14 15 16];
```

```
>> row_vector(6:-2:1)

ans =

16 14 12
```

```
>> row_vector(1:2:end)

ans =

11 13 15
```

```
>> row_vector(1:1:end-1)

ans =

11 12 13 14 15
```

Matrices

	Column 1	Column 2	Column 3	Column 4
Row 1	1	2	3	4
Row 2	5	6	7	8
Row 3	9	10	11	12

```
big_mat_eg=[1 2 3 4; 5 6 7 8; 9 10 11 12];
```

Length and size of the matrix

```
big_mat_eg=[1 2 3 4; 5 6 7 8; 9 10 11 12];
```

	Column 1	Column 2	Column 3	Column 4
Row 1	1	2	3	4
Row 2	5	6	7	8
Row 3	9	10	11	12

```
>> length(big_mat_eg)

ans =

4
```

```
>> size(big_mat_eg)

ans =

3    4
```

Selecting an element by index in a 2D matrix

```
big_mat_eg=[1 2 3 4; 5 6 7 8; 9 10 11 12];
```

	Column 1	Column 2	Column 3	
Row 1	1 $_{1}$	2 $_{4}$	3 $_{7}$	4 $_{10}$
Row 2	5 $_{2}$	6 $_{5}$	7 $_{8}$	8 $_{11}$
	9 $_{3}$	10 $_{6}$	11 $_{9}$	12 $_{12}$

```
>> big_mat_eg(:,1)

ans =

1
5
9
```

```
>> big_mat_eg(2,:)

ans =

5 6 7 8
```

```
>> big_mat_eg(:,3:end)

ans =

3     4
7     8
11    12
```

Selecting an element by index in a 2D matrix

```
matrix_example=[31 32 33; 34 35 36];
```

	Column 1	Column 2	Column 3
Row 1	31 $_{1}$	32 $_{3}$	33 $_{5}$
Row 2	34 $_{2}$	35 $_{4}$	36 $_{6}$

```
>> matrix_example(1:3)

ans =

31 34 32
```

```
>> matrix_example(1:2:end)

ans =

31 32 33
```

```
>> matrix_example(end:-1:1)

ans =

36 33 35 32 34 31
```

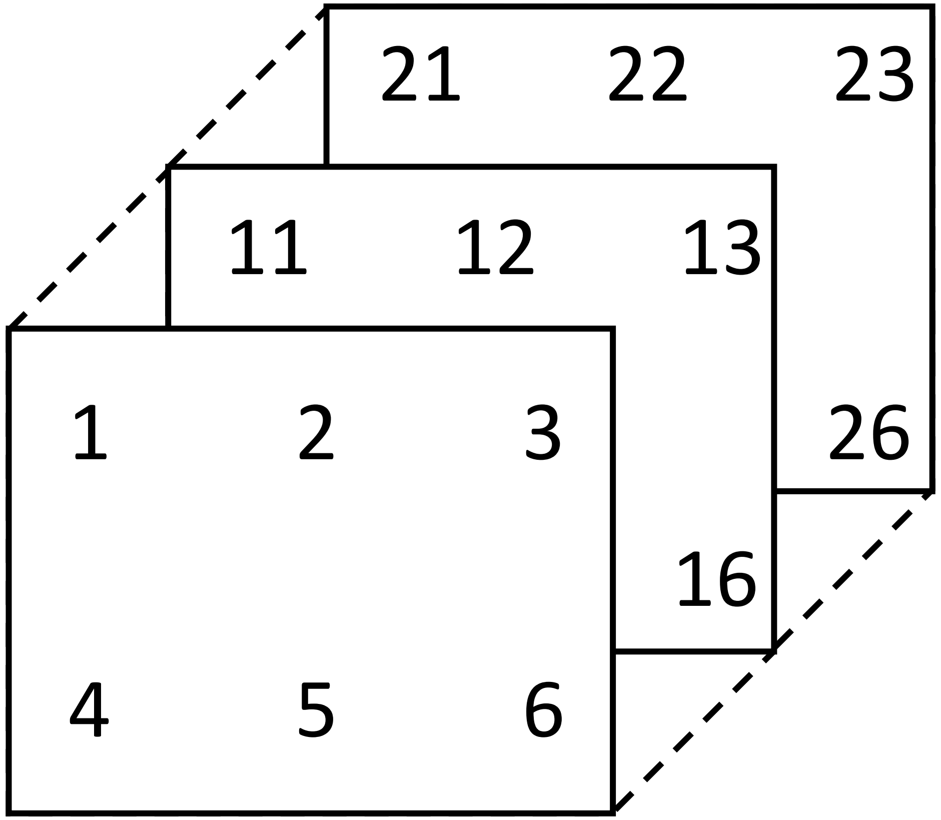
3 Dim matrix (can then be generalized to N Dim)

What will be a good example of a 3 dim matrix?

```
three_dim_mat(:,:,1)=[1 2 3; 4 5 6];
three_dim_mat(:,:,2)=[11 12 13; 14 15 16];
three_dim_mat(:,:,3)=[21 22 23; 24 25 26];
```

```
>> three_dim_mat(4)

ans =

5
```

```
>> three_dim_mat(9)

ans =

12
```

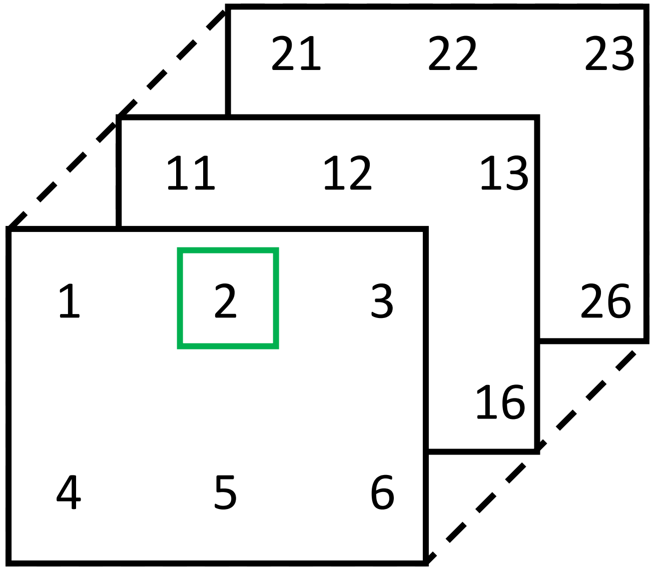


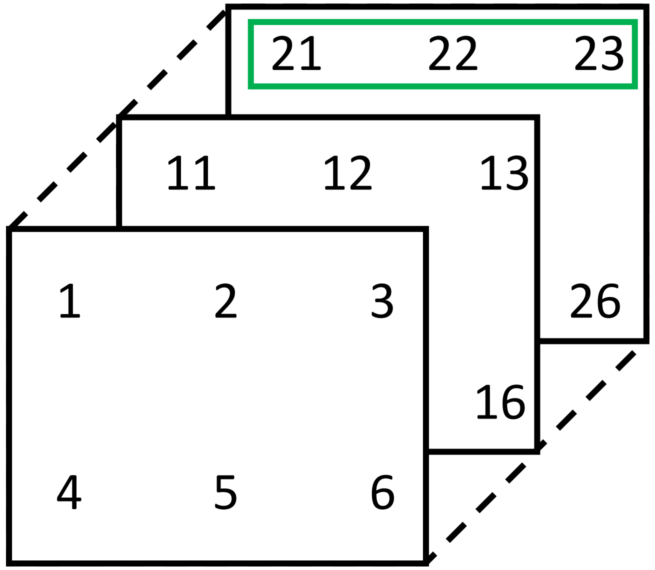
3 Dim matrix (can then be generalized to N Dim)

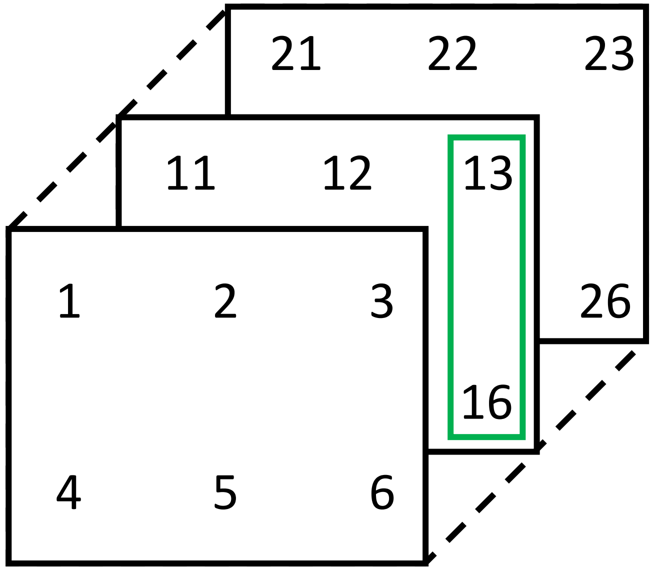
```
>>three_dim_mat(1,2,1)

ans =

5
```

```
>> three_dim_mat(1,:,3)

ans =

21 22 23
```

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

ans =
13
16
```

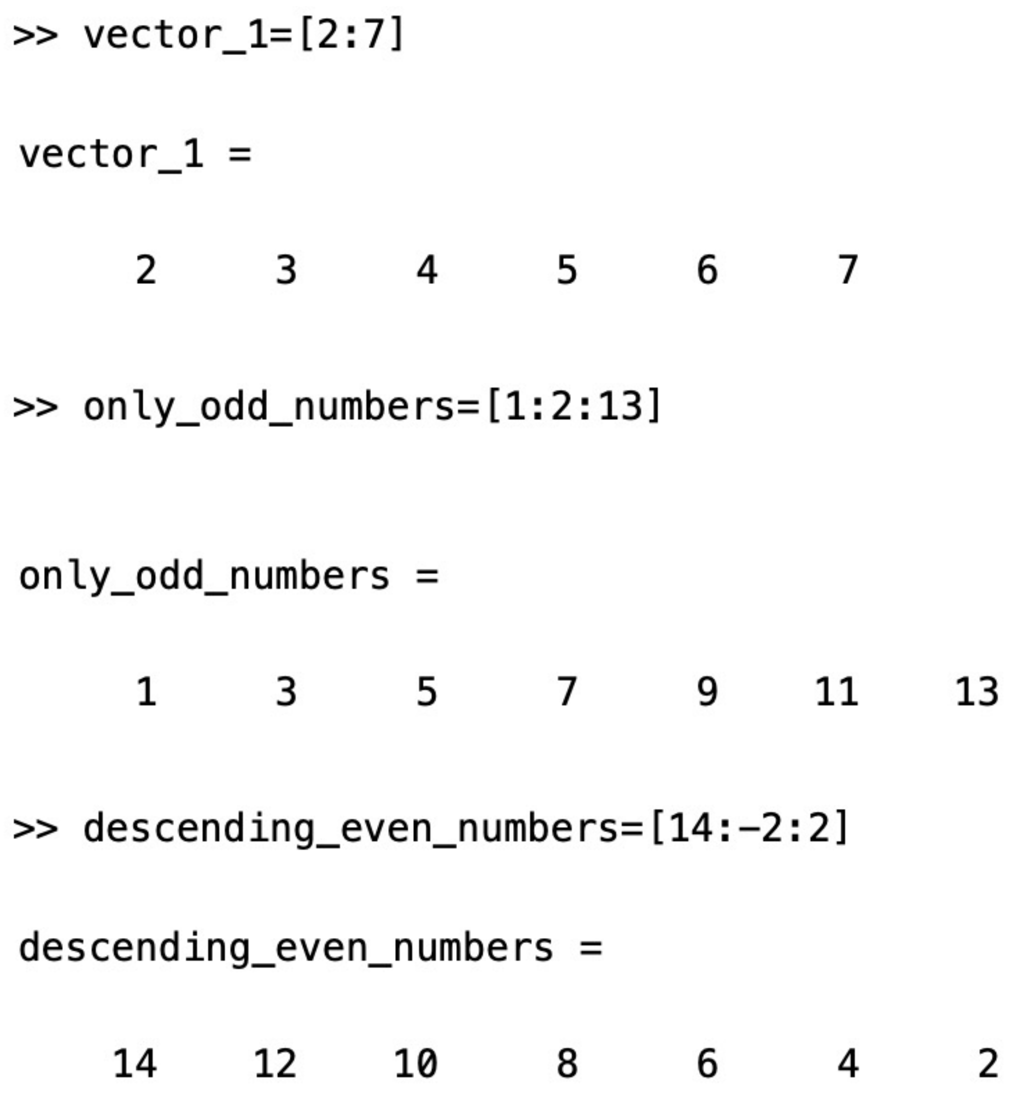






Defining matrices in MATLAB

To define a matrix:
matrix_name=[start_value : step_value : end_value];



```
>> vector_1=[2:7]

vector_1 =

     2     3     4     5     6     7

>> only_odd_numbers=[1:2:13]


only_odd_numbers =

     1     3     5     7     9    11    13

>> descending_even_numbers=[14:-2:2]

descending_even_numbers =

    14    12    10     8     6     4     2
```

Defining matrices in MATLAB

Another way to define a matrix is to use the function linspace/linearly spaced vector (https://www.mathworks.com/help/matlab/ref/linspace.html)

```
matrix_name=linspace(start_num, end_num, num_elements)
```

Here is an example to generate an array with 7 elements between -3 and 3 (including both of them).

```
>> test=linspace(-3,3,7)

test =
     -3 -2 -1 0 1 2 3
```

Defining matrices in MATLAB

You can use built-in functions to define matrices. For e.g.

1. To generate matrix with all 1s: ones(num_row, num_column)
2. To generate matrix with all 0s: zeros(num_row, num_column)
3. To generate matrix with all NaNs: nan(num_row, num_column)

```
>> ones(2,3)

ans =

1 1 1
1 1 1
```

```
>> zeros(2,4)

ans =

0 0 0 0
0 0 0 0
```

```
>> nan(2,2)

ans =

NaN NaN
NaN NaN
```

Defining matrices in MATLAB

- To generate matrix with random numbers: rand(num_row, num_col)

```
>> rand(2,3)

ans =

0.0782 0.1067 0.0046
0.4427 0.9619 0.7749
```

```
>> rand(2)

ans =

0.8173 0.0844
0.8687 0.3998
```

- To generate matrix with random integers: randi(max_integer,num_row, num_col)

```
>>  randi(9,2,3)

ans =

3 4 2
8 9 3
```

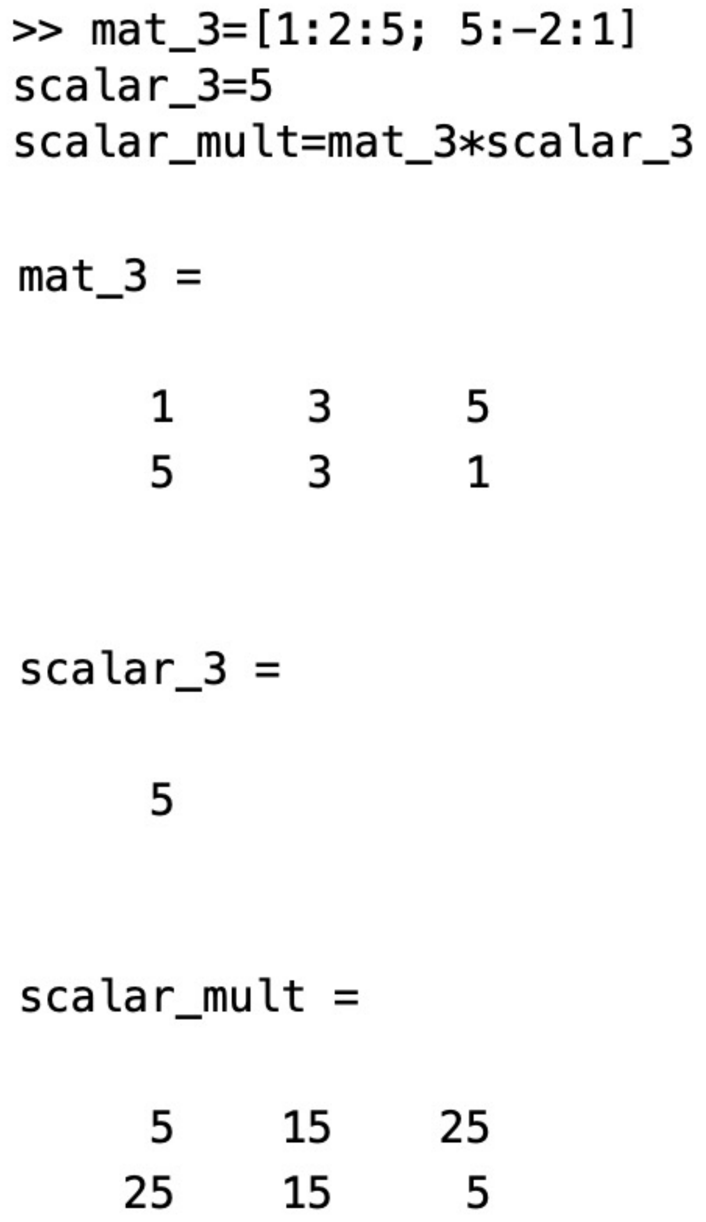
Matrix operations: Addition and subtraction

```
>> mat_1=[6:2:10;20:-2:16]
>> mat_2=[3:5;-5:-3]

mat_1 =

6 8 10
20 18 16


mat_2 =

3 4 5
-5 -4 -3
```

```
>>  mat_add=mat_1+mat_2

mat_add =

9 12 15
15 14 13


>> mat_add=mat_1-mat_2

mat_add =

3 4 5
25 22 19
```

Matrix operations: Scalar multiplication

```
>> mat_3=[1:2:5; 5:-2:1]
>> scalar_3=5
>> scalar_mult=mat_3*scalar_3
```



```
>> mat_3=[1:2:5; 5:-2:1]
scalar_3=5
scalar_mult=mat_3*scalar_3

mat_3 =

     1     3     5
     5     3     1


scalar_3 =

     5


scalar_mult =

     5    15    25
    25    15     5
```

Matrix operations: Matrix (or vector) multiplication

Keep in mind the dimension of the two matrices for vector multiplication

$$[A]_{mxn} \text{ X } [B]_{nxm} = [C]_{mxm}$$

For example:

```
>> mat_4=[3:5; 5:7]
>> mat_5=[2:3; 4:5; 6:7]
>> vector_mult=mat_4*mat_5
```

```
>> mat_4 =      >> mat_5 =      >> vector_mult =

3   4   5       2    3          52    64
5   6   7       4    5          76    94
                6    7
```

Matrix operations: Element by element multiplication

Both the matrices should be of the same size.

For example:

```
>> mat_6=[3:5; 5:7]
>> mat_7=[2:4; 4:6]
>> element_by_element_mult=mat_6.*mat_7
```

```
>> mat_6 =              >> mat_7 =              >> element_by_element_mult =

3   4   5               2   3   4               6    12    20
5   6   7               4   5   6               20   30    42
```

Matrix operations: Transpose of a matrix

Flips the dimensions of the matrix

$$[A]_{mxn} \rightarrow [B]_{nxm}$$

```
>> mat_8=[3:5; -7:-5]
>> mat_9=mat_8'
```

```
>> mat_8 =

 3    4    5
-7   -6   -5
```

```
>> mat_9 =

3    -7
4    -6
5    -5
```

Splitting matrices

- Sometimes you might want to split the matrix and work on a subset of the matrix (for e.g. just a row or column)

```
>> original_mat=[1:7; 21:27; 51:57]
>> split_row=original_mat(2,:)
>> split_column=original_mat(:,4)
```

```
>> split_row =

21  22  23  24  25  26  27
```

```
>> split_column =

4
24
54
```

Concatenating matrices: Horizontal

- Combine two matrices horizontally
- If you have 2 matrices A and B then to horizontally concatenate them:
- C=[A B] or C=horzcat(A, B)

```
>> mat_10=ones(3,2)*4
>> mat_11=randi(7,3,2)
>> horz_cat_1=[mat_10 mat_11]
>> horz_cat_2=horzcat(mat_10,mat_11)
```

```
>> mat_10 =

4  4
4  4
4  4
```

```
>> mat_11 =

6  7
7  5
1  1
```

```
>> horz_cat_1 =

4  4  6  7
4  4  7  5
4  4  1  1
```

```
>> horz_cat_2 =

4  4  6  7
4  4  7  5
4  4  1  1
```

Concatenating matrices: Vertical

- Combine two matrices vertically
- If you have 2 matrices A and B then to horizontally concatenate them:
- C=[A; B] or C=vertcat(A, B)

```
>> mat_12=ones(2,3)*4
>> mat_13=randi(7,2,3)
>> vert_cat_1=[mat_12; mat_13]
>> vert_cat_2=vertcat(mat_12,mat_13)
```

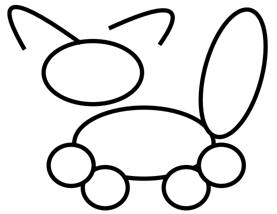
```
>> mat_12 =

4   4   4
4   4   4
```

```
>> mat_13 =

7   6   3
4   1   7
```

```
>>vert_cat_1 =

4   4   4
4   4   4
7   6   3
4   1   7
```

```
>>vert_cat_2 =

4   4   4
4   4   4
7   6   3
4   1   7
```

cat(DIM, A,B)



mean of an array

```
>> new_matrix=[2:2:16; 10:-2:-5; 3:3:24];
```

```
>> new_matrix =

2     4   6   8   10  12   14   16
10    8   6   4   2    0    -2   -4
3     6   9   12 15  18   21    24
```

By default, MATLAB takes mean across each column

```
| 2 | | 4 | | 6 | | 8 | | 10 | | 12 | | 14 | | 16 |
| 10| | 8 | | 6 | | 4 | | 2  | | 0  | | -2 | | -4 |
| 3 | | 6 | | 9 | | 12| | 15 | | 18 | | 21 | | 24 |
```

```
mean(new_matrix)

ans =

5 6 7 8 9 10 11 12
```

What if you want to find the mean across each row?

```
| 2     4   6   8   10  12   14   16 |
| 10    8   6   4   2    0    -2   -4 |
| 3     6   9   12 15  18   21    24 |
```

```
mean(new_matrix,2)

ans =

9
3
13.5
```

2=across each row

1=across each column

sort

- To sort an array, you can use the inbuilt MATLAB function sort

- mat_14=[10 5 2 3 6 7 0 -1 -12 7 6]
- sort(mat_14)
-

```
ans =

-12   -1   0   2   3   5   6   6   7   7   10
```

unique

- Sometimes your goal is to find the unique elements in an array
- For e.g. in the previous example 6 and 7 appeared twice.
- To find the unique elements, use the function ‘unique’.

- mat_14=[10 5 2 3 6 7 0 -1 -12 7 6]
- unique(mat_14)
- ans =

```
-12    -1   0    2    3    5    6    7    10
```

Note that the ‘unique’ function also sorts the output

reshape

- Using reshape function, we can reshape a matrix to another size

- to_reshape_array=[1:1:10]
- reshape(to_reshape_array, [2,5])

- ans =

```
1    6
2    7
3    8
4    9
5   10
```