Command window: where you give commands, do arithmetic calculation, see output. Workspace keep track of all the variables that are being used and their corresponding values. Variable is a data container.

'whos' – shows all the variables that are currently being manipulated by MATLAB

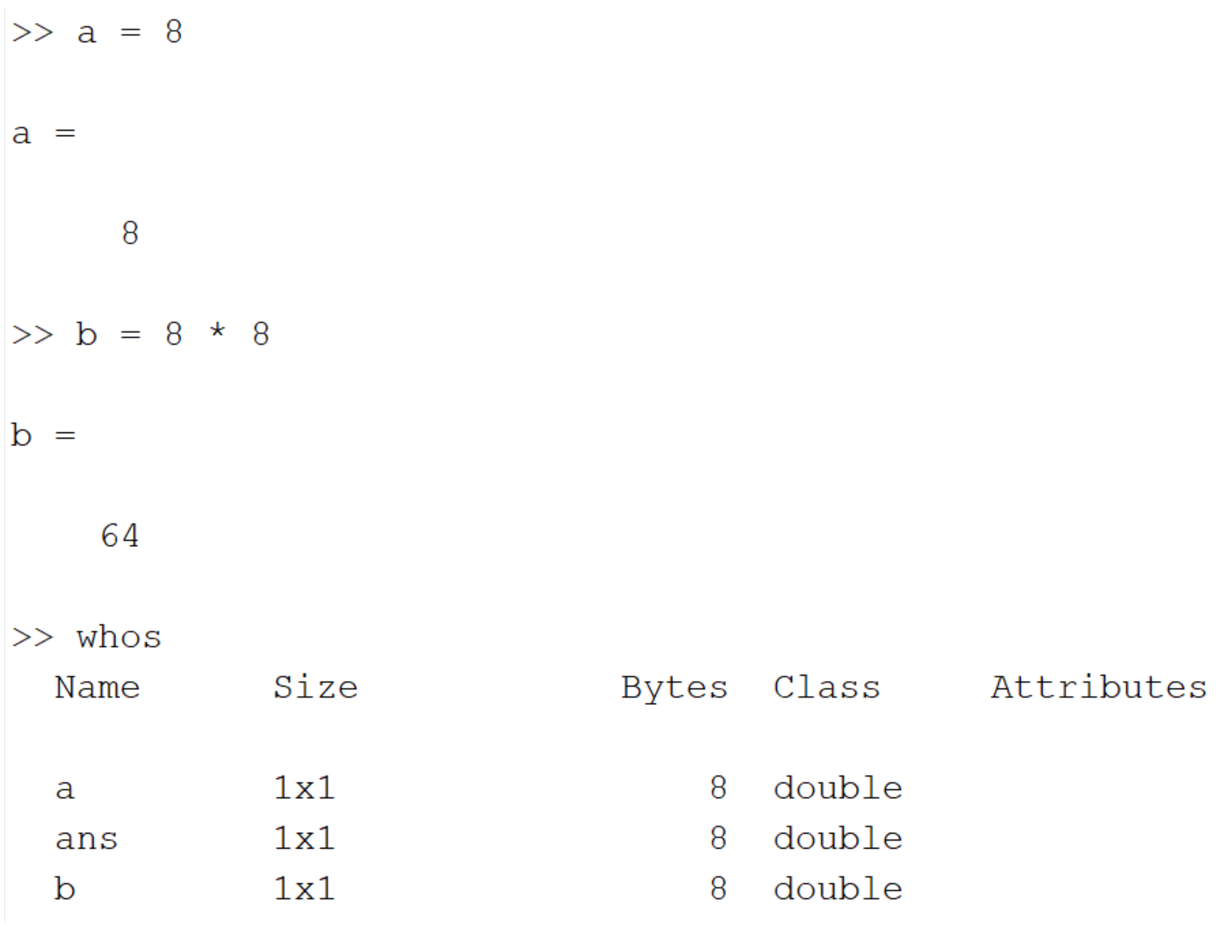
clear a – clear variable a

clear all – clear all variable

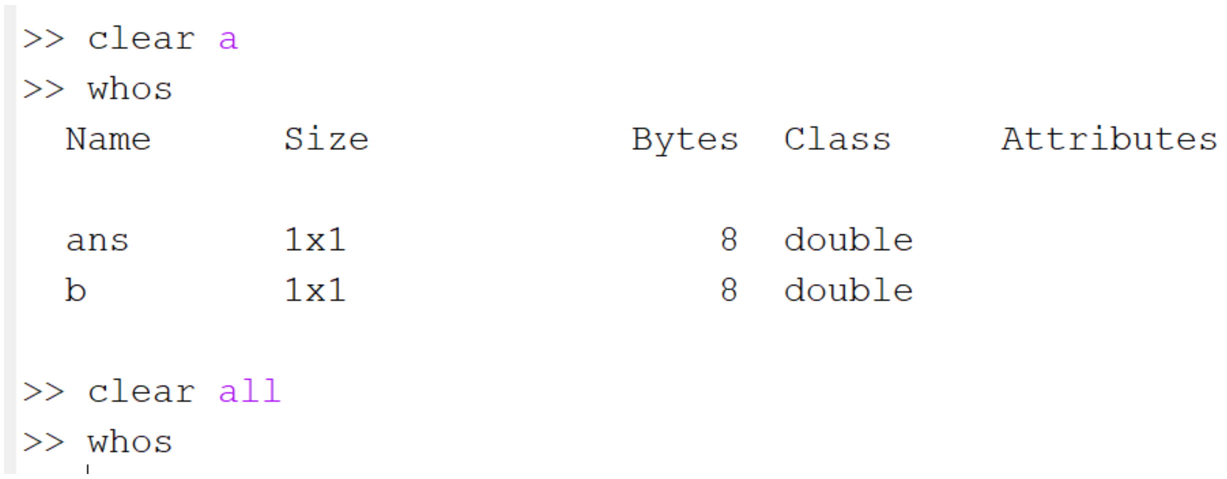
up arrow key – shows all the commands we used so far with time stamps

clc – clear the command window

```
>> 8 * 8

ans =

    64

>> 8 + 8

ans =

    16

>> 8 / 8

ans =

     1

>> 8 - 8

ans =

     0
```



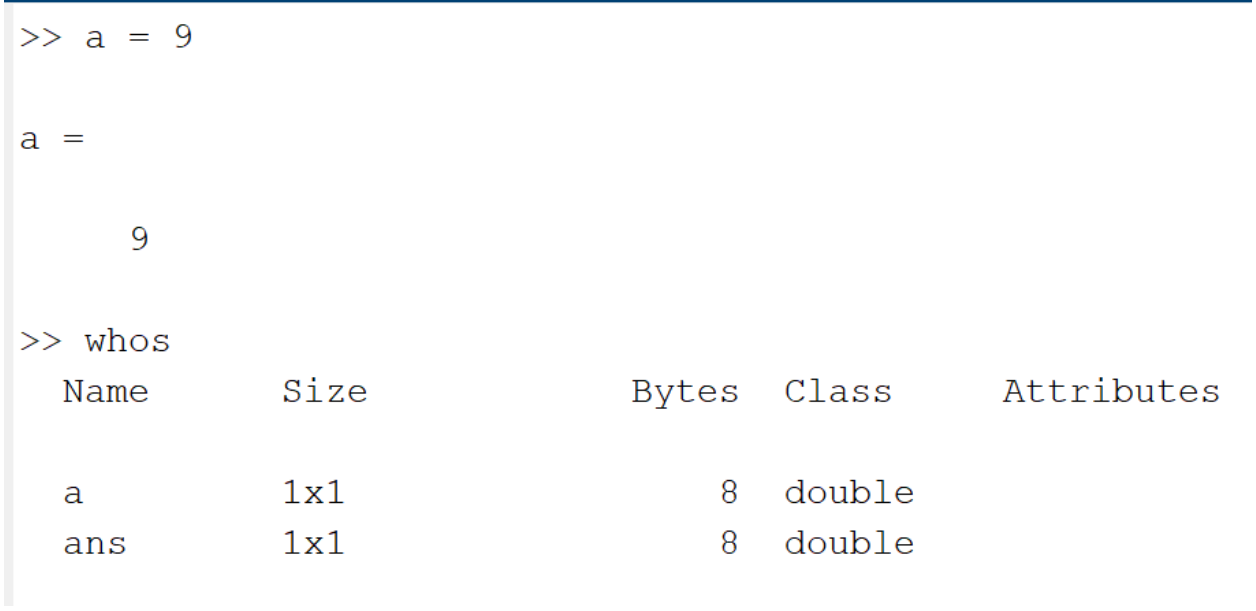
```
>> a = 8

a =

     8

>> b = 8 * 8

b =

    64

>> whos
  Name      Size            Bytes  Class     Attributes

  a         1x1                 8  double
  ans       1x1                 8  double
  b         1x1                 8  double
```



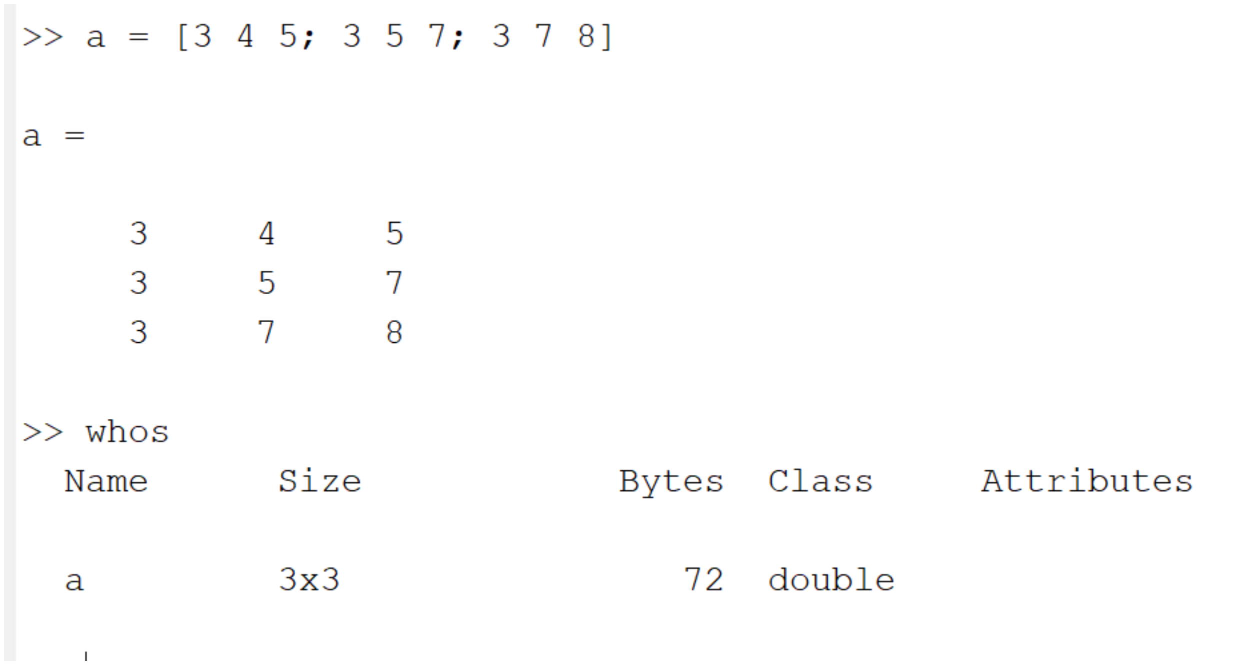
```
>> clear a
>> whos
  Name      Size            Bytes  Class     Attributes

  ans       1x1                 8  double
  b         1x1                 8  double

>> clear all
>> whos
```

MATLAB – Matrices Laboratory

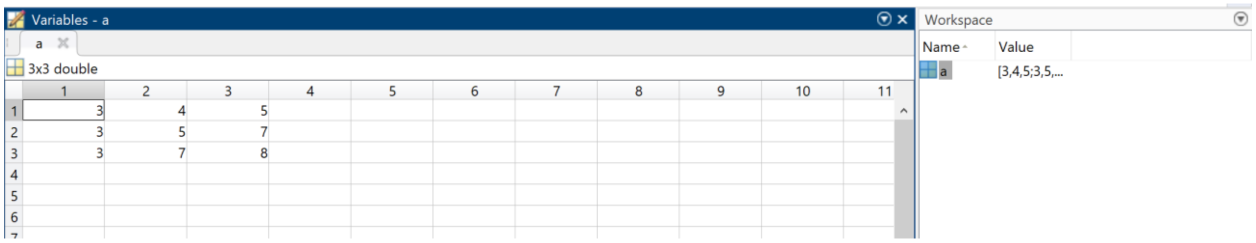
Whatever variable you define on the MATLAB is actually a Matrix.



```
>> a = 9

a =

     9

>> whos
  Name        Size            Bytes  Class     Attributes

  a           1x1                 8  double
  ans         1x1                 8  double
```



```
>> a = [3 4 5; 3 5 7; 3 7 8]

a =

     3     4     5
     3     5     7
     3     7     8

>> whos
  Name        Size            Bytes  Class     Attributes

  a           3x3                72  double
```



Variables - a

a

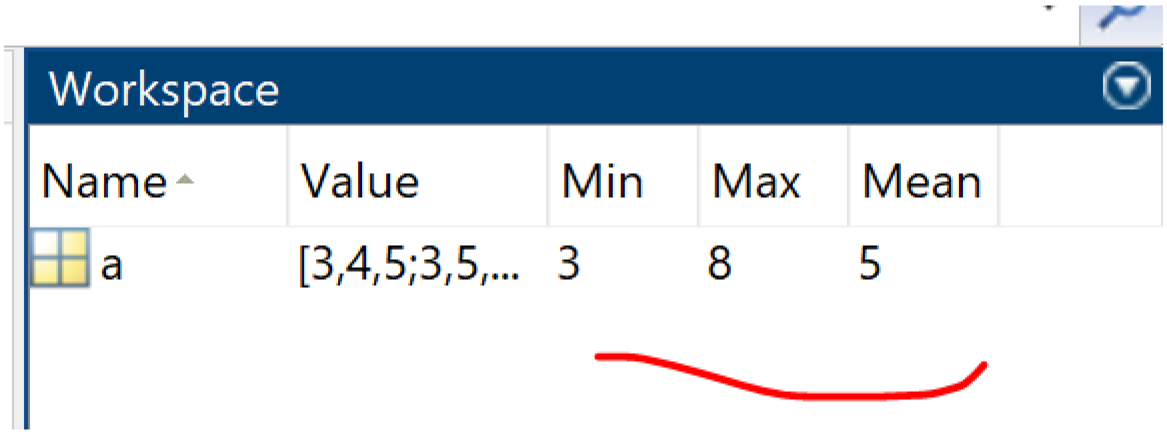
3x3 double

	1	2	3	4	5	6	7	8	9	10	11
1	3	4	5								
2	3	5	7								
3	3	7	8								
4											
5											
6											

Workspace

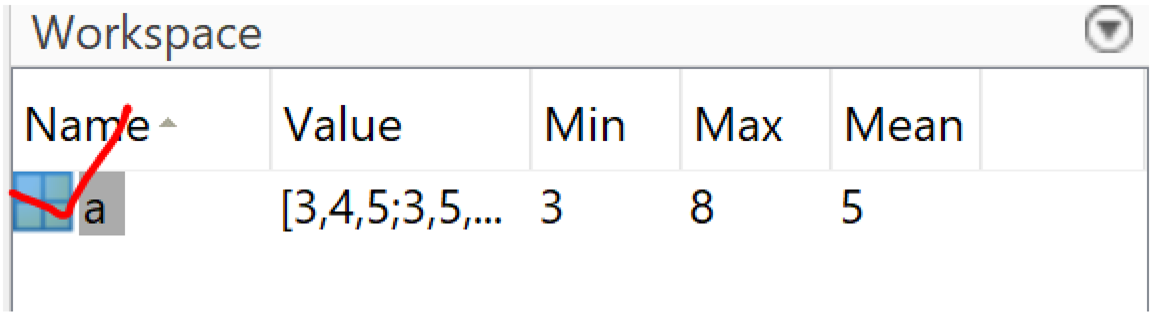
Name	Value
a	[3,4,5;3,5,...

In Workspace right click on Name: Find Mean, Max, Min etc.


Workspace

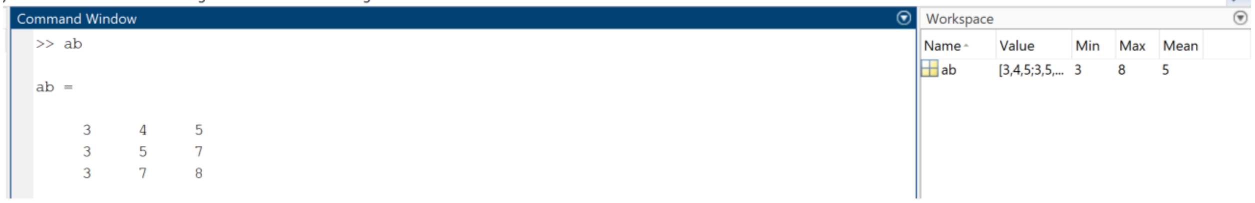
Name	Value	Min	Max	Mean
a	[3,4,5;3,5,...	3	8	5

In MATLAB, each and every variable is by default a matrix


Workspace

Name	Value	Min	Max	Mean
a	[3,4,5;3,5,...	3	8	5

Right click on the box: option for rename, save the variable etc.


Command Window

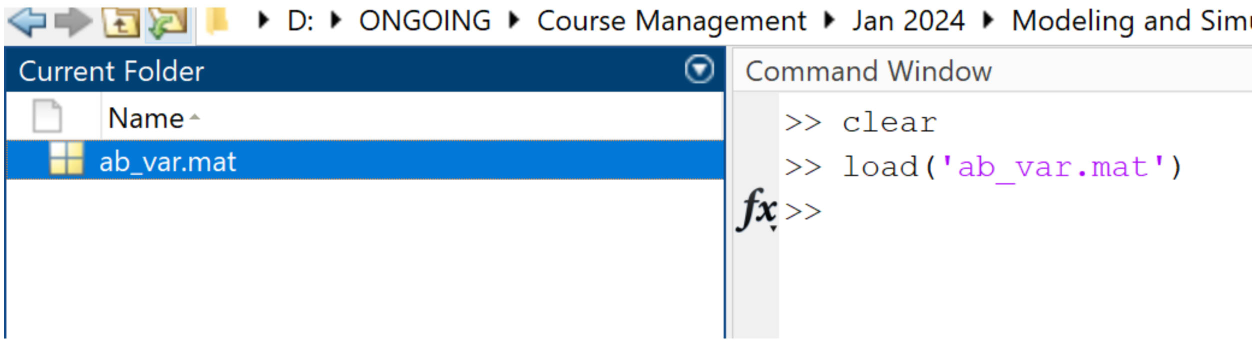
```
>> ab

ab =

     3     4     5
     3     5     7
     3     7     8
```

Workspace

Name	Value	Min	Max	Mean
ab	[3,4,5;3,5,...	3	8	5

Renamed the a as ab. And now, see ab.

If you save the variable, when you double click on it later, it will load the variable in the workspace


D: ▸ ONGOING ▸ Course Management ▸ Jan 2024 ▸ Modeling and Simu

Current Folder

Name

ab_var.mat

Command Window

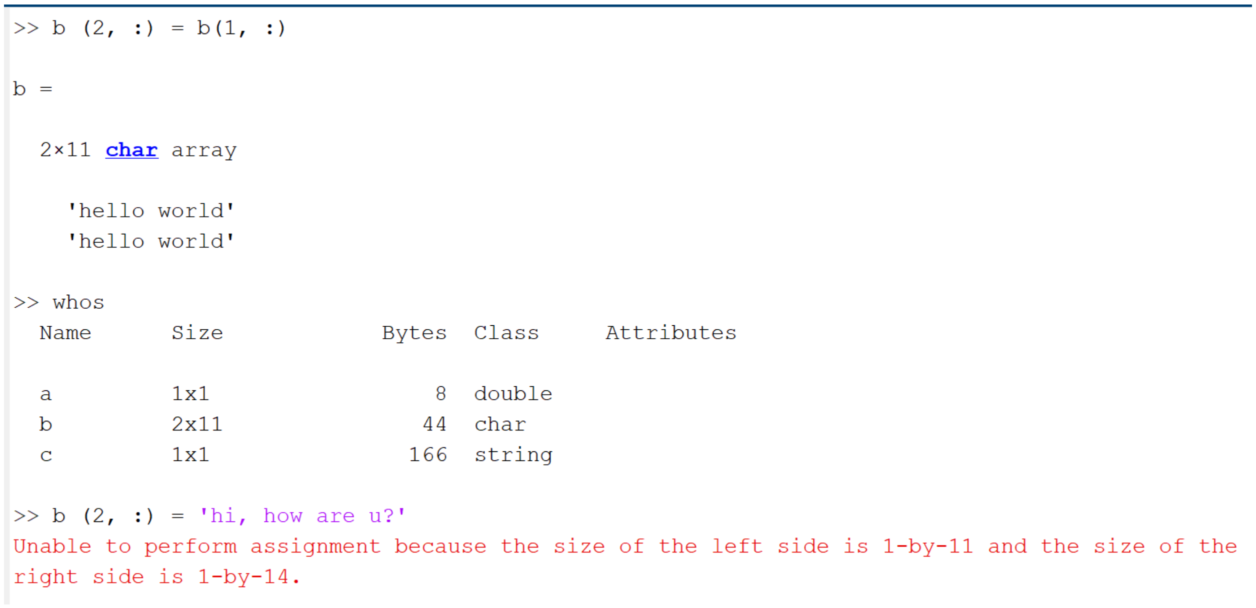
```
>> clear
>> load('ab_var.mat')
>>
```

Different Types of Variables: Numeric, Character, Strings, Logical

```
>> a = 9

a =

     9

>> b = 'hello world'

b =

    'hello world'

>> whos
  Name          Size                Bytes  Class      Attributes

  a             1x1                     8  double
  b             1x11                   22  char

>> c = "hello world"

c =

    "hello world"

>> whos
  Name          Size                Bytes  Class      Attributes

  a             1x1                     8  double
  b             1x11                   22  char
  c             1x1                   166  string
```

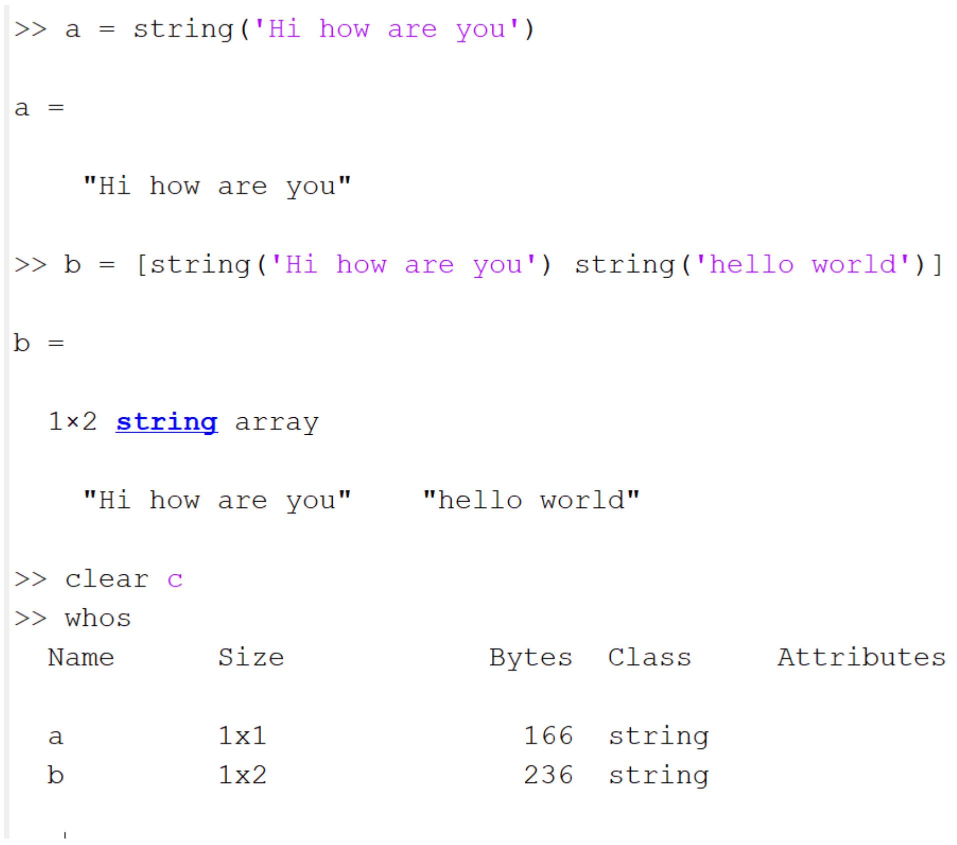
See the difference here for ‘ ’ and “ ”. One is char, another is string.

Say, we want to add one more row to this b matrix, and the second row will be the copy of the first row.

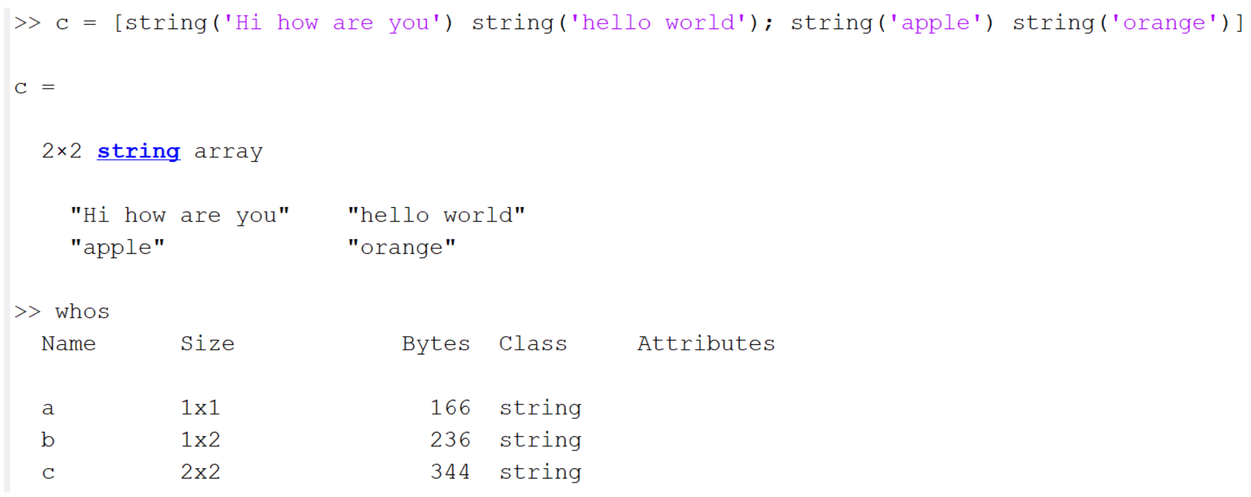


```
>> b (2, :) = b(1, :)

b =

  2×11 char array

    'hello world'
    'hello world'

>> whos
  Name      Size            Bytes  Class     Attributes

  a         1x1                 8  double
  b         2x11               44  char
  c         1x1               166  string

>> b (2, :) = 'hi, how are u?'
Unable to perform assignment because the size of the left side is 1-by-11 and the size of the
right side is 1-by-14.
```

The Colon operations here means all. So, b (row, column). So, b (2, :) means row: 2 and column: all. The error here because it does not contain the same number of columns as the previous rows. Now, say we want to store characters of various sizes. We will have to use string.

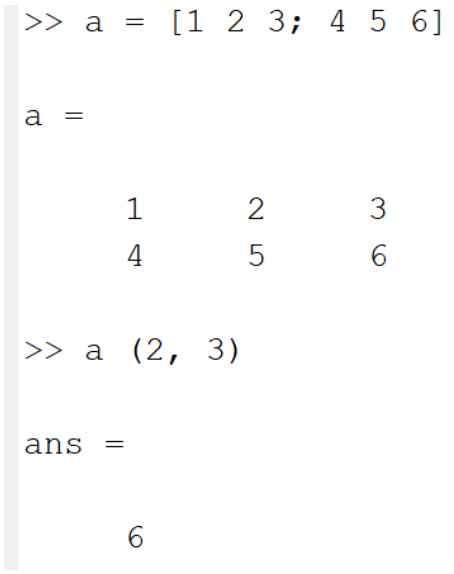


```
>> a = string('Hi how are you')

a =

    "Hi how are you"

>> b = [string('Hi how are you') string('hello world')]

b =

  1×2 string array

    "Hi how are you"    "hello world"

>> clear c
>> whos
  Name      Size            Bytes  Class     Attributes

  a         1x1               166  string
  b         1x2               236  string
```



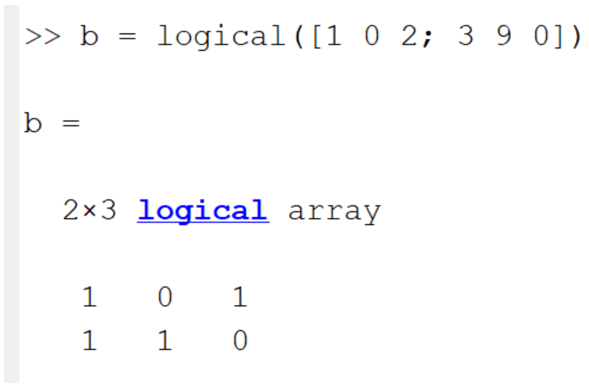
```
>> c = [string('Hi how are you') string('hello world'); string('apple') string('orange')]

c =

  2×2 string array

    "Hi how are you"    "hello world"
    "apple"             "orange"

>> whos
  Name      Size            Bytes  Class     Attributes

  a         1x1               166  string
  b         1x2               236  string
  c         2x2               344  string
```

Now, how we can access the individual elements of the matrix.



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

a =

     1     2     3
     4     5     6

>> a (2, 3)

ans =

     6
```

a (2, 3) means the value in the row = 2 and column = 3 position in the matrix.

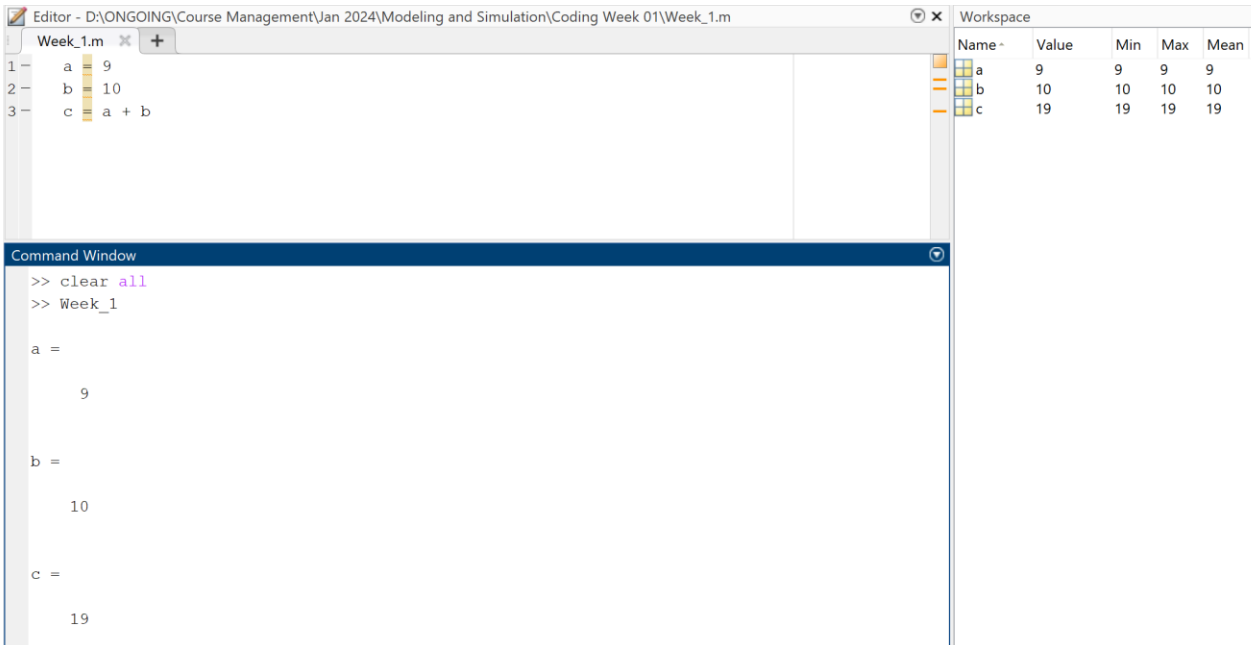


```
>> b = logical([1 0 2; 3 9 0])

b =

  2×3 logical array

   1   0   1
   1   1   0
```

Logical here is a data type which can only contain value 1 or 0.

Writing scripts, Commenting and Semicolon effect

Actually codes are written in the script, because you want to edit time to time and save the program.



Editor - D:\ONGOING\Course Management\Jan 2024\Modeling and Simulation\Coding Week 01\Week_1.m

Week_1.m

```
a = 9
b = 10
c = a + b
```

Workspace

Name	Value	Min	Max	Mean
a	9	9	9	9
b	10	10	10	10
c	19	19	19	19

Command Window

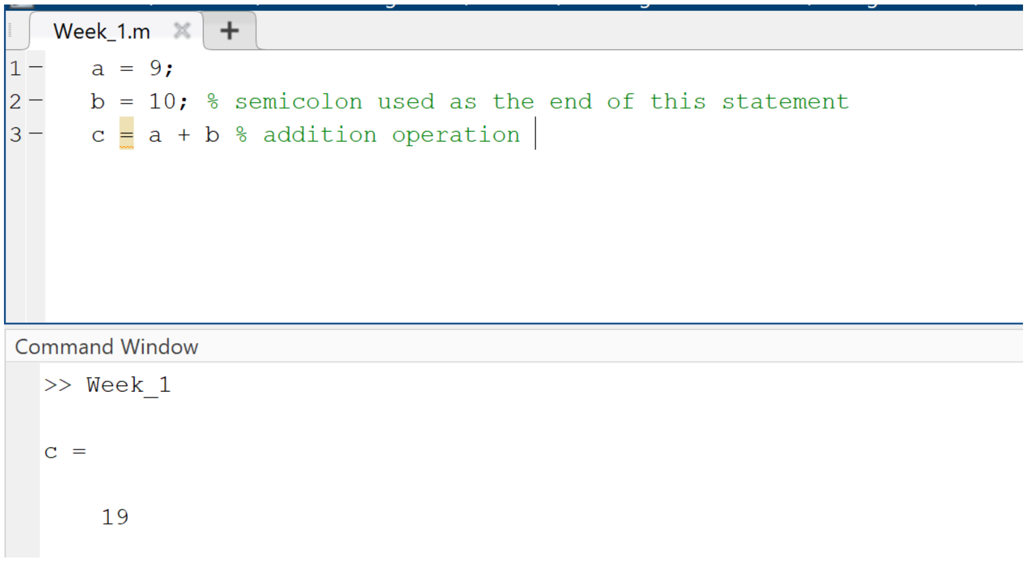
```
>> clear all
>> Week_1

a =

     9


b =

    10


c =

    19
```

Write the code in the script. Save it. Extension is .m

Run it. Output is shown in the command window.

Use semicolon at the end of those statement for which you don't want to see output.

See with semicolon:



Week_1.m

```
a = 9;
b = 10; % semicolon used as the end of this statement
c = a + b % addition operation
```

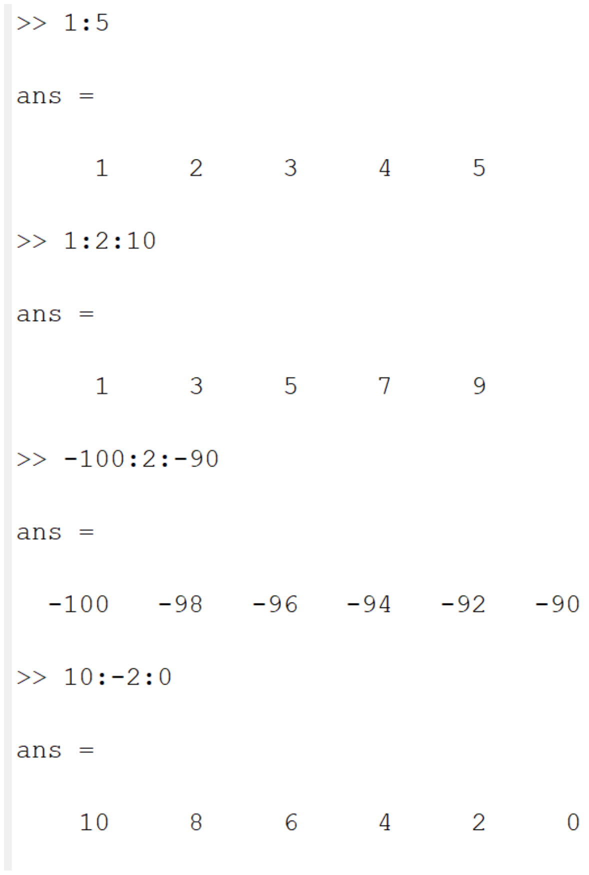
Command Window

```
>> Week_1

c =

    19
```

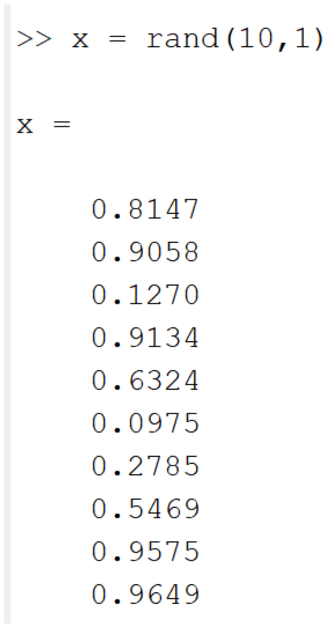
Also, see, % is used for commenting.

Colon operator:

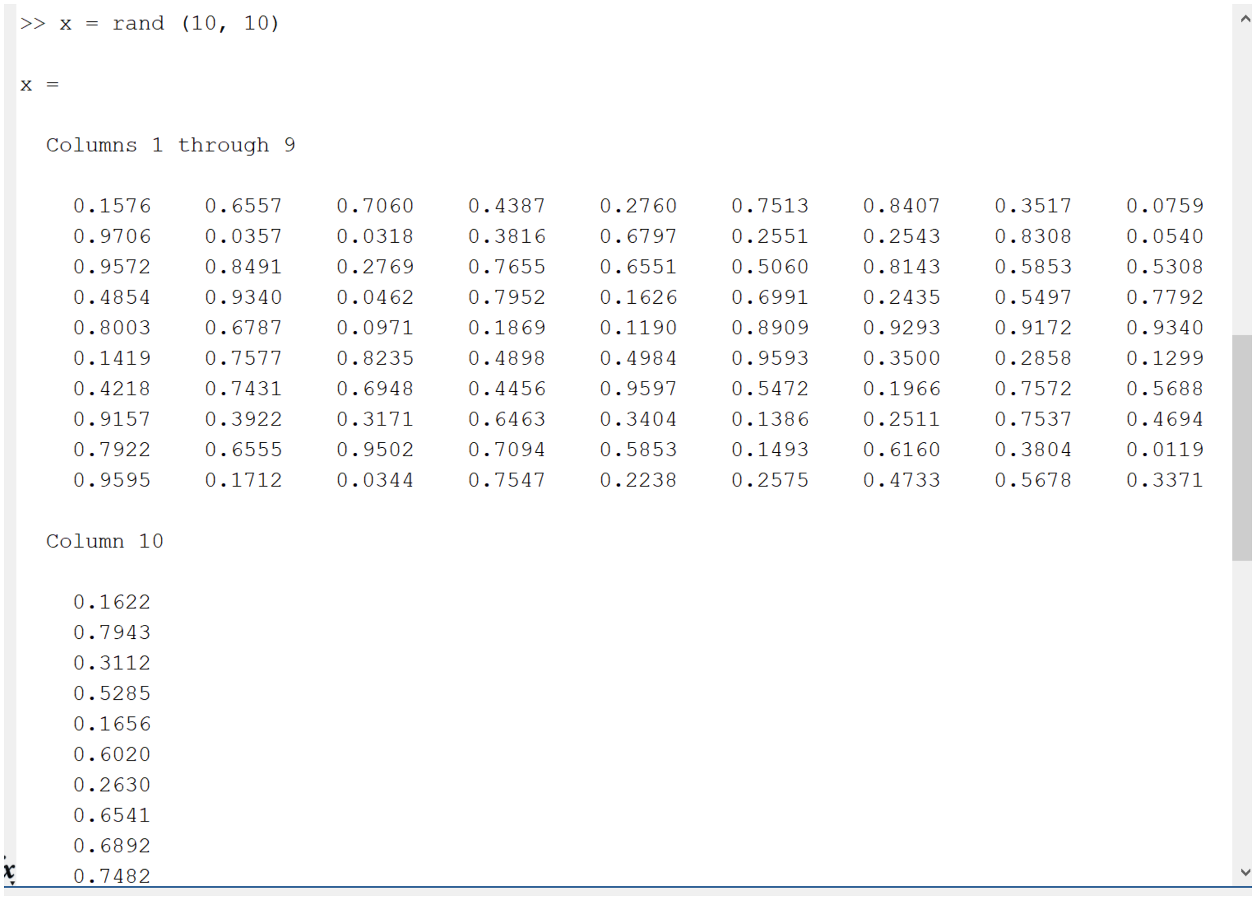
Colon can be used to select different portion or subset of the data. Very helpful to manipulate data.



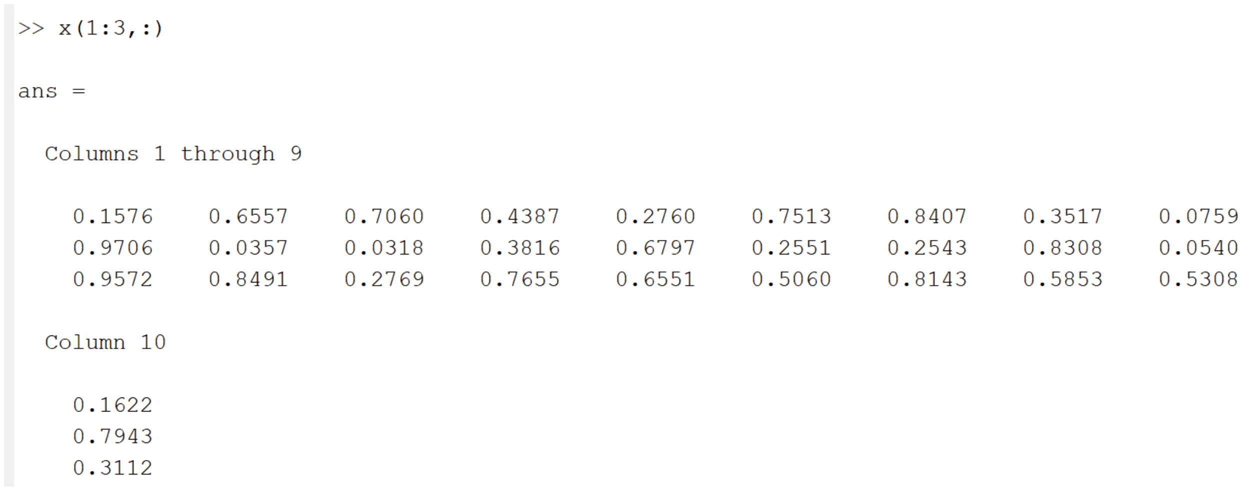
```
>> 1:5

ans =

     1     2     3     4     5

>> 1:2:10

ans =

     1     3     5     7     9

>> -100:2:-90

ans =

  -100   -98   -96   -94   -92   -90

>> 10:-2:0

ans =

    10     8     6     4     2     0
```



```
>> x = rand(10,1)

x =

    0.8147
    0.9058
    0.1270
    0.9134
    0.6324
    0.0975
    0.2785
    0.5469
    0.9575
    0.9649
```



```
>> x = rand (10, 10)

x =

  Columns 1 through 9

    0.1576    0.6557    0.7060    0.4387    0.2760    0.7513    0.8407    0.3517    0.0759
    0.9706    0.0357    0.0318    0.3816    0.6797    0.2551    0.2543    0.8308    0.0540
    0.9572    0.8491    0.2769    0.7655    0.6551    0.5060    0.8143    0.5853    0.5308
    0.4854    0.9340    0.0462    0.7952    0.1626    0.6991    0.2435    0.5497    0.7792
    0.8003    0.6787    0.0971    0.1869    0.1190    0.8909    0.9293    0.9172    0.9340
    0.1419    0.7577    0.8235    0.4898    0.4984    0.9593    0.3500    0.2858    0.1299
    0.4218    0.7431    0.6948    0.4456    0.9597    0.5472    0.1966    0.7572    0.5688
    0.9157    0.3922    0.3171    0.6463    0.3404    0.1386    0.2511    0.7537    0.4694
    0.7922    0.6555    0.9502    0.7094    0.5853    0.1493    0.6160    0.3804    0.0119
    0.9595    0.1712    0.0344    0.7547    0.2238    0.2575    0.4733    0.5678    0.3371

  Column 10

    0.1622
    0.7943
    0.3112
    0.5285
    0.1656
    0.6020
    0.2630
    0.6541
    0.6892
    0.7482
```



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

ans =

  Columns 1 through 9

    0.1576    0.6557    0.7060    0.4387    0.2760    0.7513    0.8407    0.3517    0.0759
    0.9706    0.0357    0.0318    0.3816    0.6797    0.2551    0.2543    0.8308    0.0540
    0.9572    0.8491    0.2769    0.7655    0.6551    0.5060    0.8143    0.5853    0.5308

  Column 10

    0.1622
    0.7943
    0.3112
```

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

ans =

  Columns 1 through 9

    0.1576    0.6557    0.7060    0.4387    0.2760    0.7513    0.8407    0.3517    0.0759
    0.9572    0.8491    0.2769    0.7655    0.6551    0.5060    0.8143    0.5853    0.5308
    0.8003    0.6787    0.0971    0.1869    0.1190    0.8909    0.9293    0.9172    0.9340
    0.4218    0.7431    0.6948    0.4456    0.9597    0.5472    0.1966    0.7572    0.5688
    0.7922    0.6555    0.9502    0.7094    0.5853    0.1493    0.6160    0.3804    0.0119

  Column 10

    0.1622
    0.3112
    0.1656
    0.2630
    0.6892
```

Basic Math Operations

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

a =

     1     2     3
     4     5     6

>> b = [3 4 5; 7 8 9]

b =

     3     4     5
     7     8     9

>> a - b

ans =

    -2    -2    -2
    -3    -3    -3

>> a * b
Error using  * 
Incorrect dimensions for matrix multiplication. Check that the number of columns in the first
matrix matches the number of rows in the second matrix. To perform elementwise
multiplication, use '.*'.

Related documentation
```

For multiplication, the column of the first matrix should match the row of the 2nd matrix

Let's take transpose of matrix b

```
>> b = b'

b =

     3     7
     4     8
     5     9
```

Now you can multiply a and b.

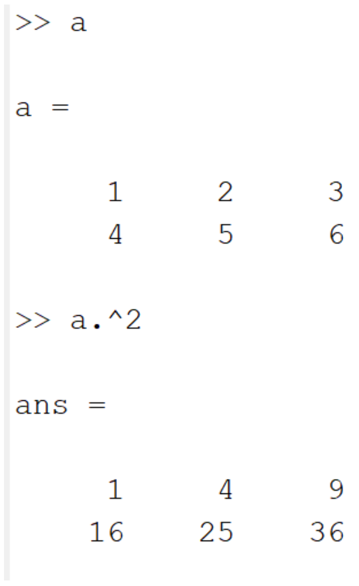
```
>> a * b

ans =

    26    50
    62   122
```

This was matrix multiplication. What if I want to do element by element multiplication? For this the two matrices must have same number of row and column.

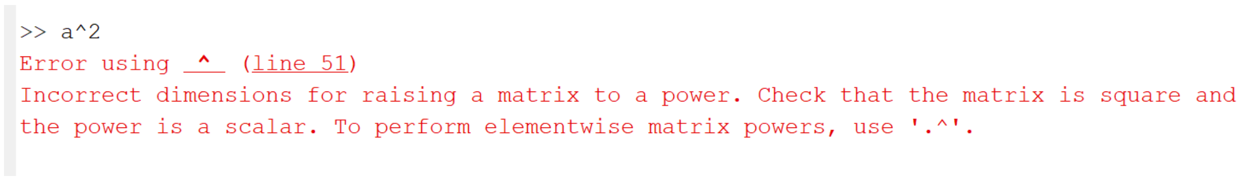
```
>> a.*a

ans =

     1     4     9
    16    25    36
```

For addition too the matrices have to be in same size.

```
a =

     1     2     3
     4     5     6

>> b

b =

     3     4     5
     7     8     9

>> a + b

ans =

     4     6     8
    11    13    15
```



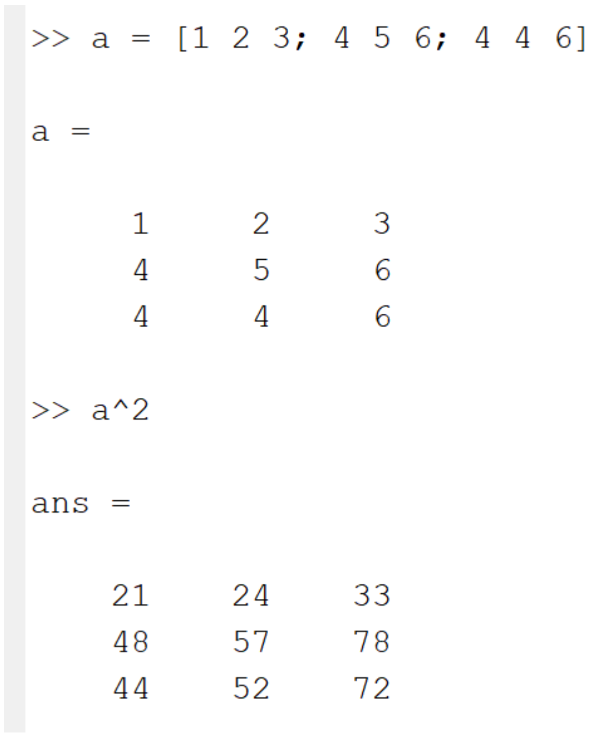
```
>> a

a =

     1     2     3
     4     5     6

>> a.^2

ans =

     1     4     9
    16    25    36
```



```
>> a^2
Error using  ^  (line 51)
Incorrect dimensions for raising a matrix to a power. Check that the matrix is square and
the power is a scalar. To perform elementwise matrix powers, use '.^'.
```

The reason for this error: here actually a*a is happening, but that's not possible because number of column do not match number of row.

See, here it's possible:



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

a =

     1     2     3
     4     5     6
     4     4     6

>> a^2

ans =

    21    24    33
    48    57    78
    44    52    72
```

Here are some MATH functions:

The least common multiple (LCM) of 4 and 6 is 12 because it is the smallest number that both 4 and 6 divide into evenly (4 × 3 = 12, 6 × 2 = 12). The greatest common divisor (GCD) of 8 and 12 is 4 because it is the largest number that divides both 8 and 12 without leaving a remainder (8 ÷ 4 = 2, 12 ÷ 4 = 3).

gcd – Greatest Common Divisor

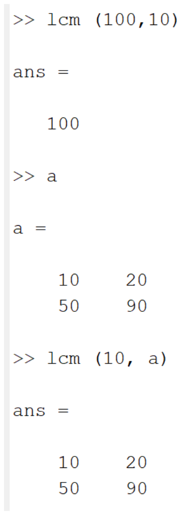
```
>> gcd (10, 20)

ans =

    10

>> a = [10 20; 50 90]

a =

    10    20
    50    90

>> b = [10 30; 100 180]

b =

    10    30
   100   180

>> gcd(a,b)

ans =

    10    10
    50    90
```

For gcd function, the input matrices have to be of same sizes.

```
>> a

a =

    10    20
    50    90

>> gcd (10, a)

ans =

    10    10
    10    10
```

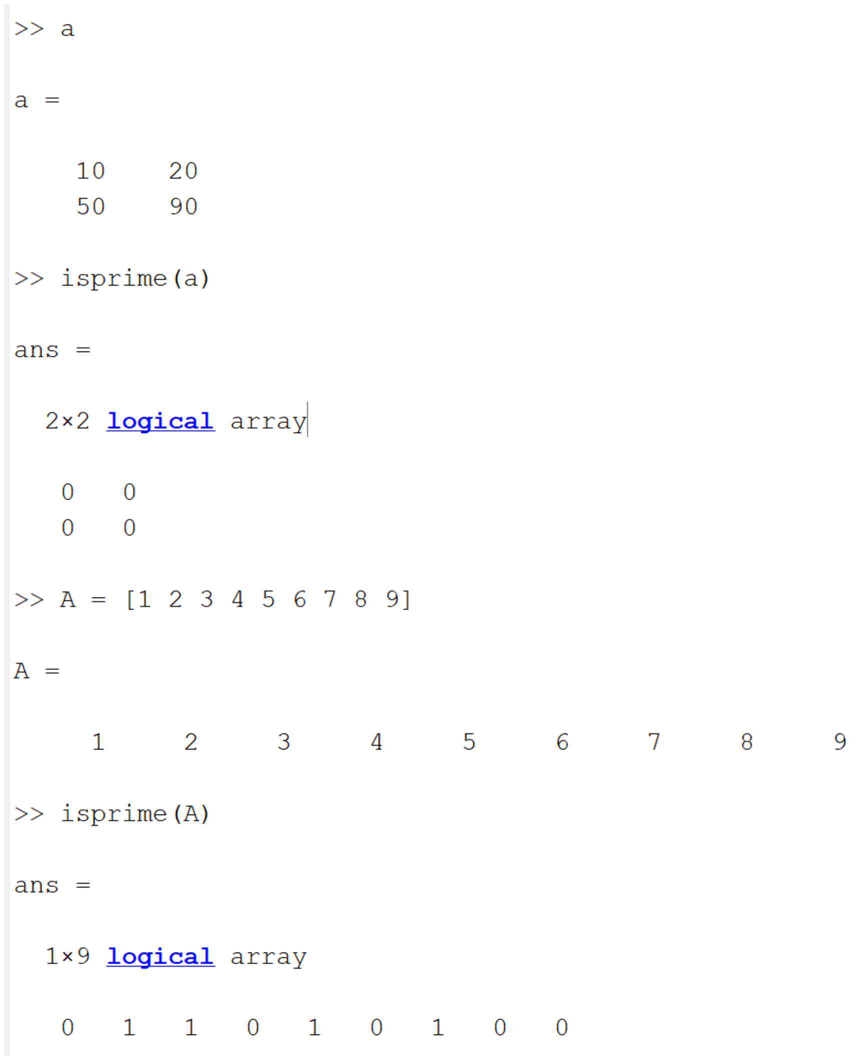
gcd of 10 and each element of the matrix a.

Now for least common multiple, lcm is used.



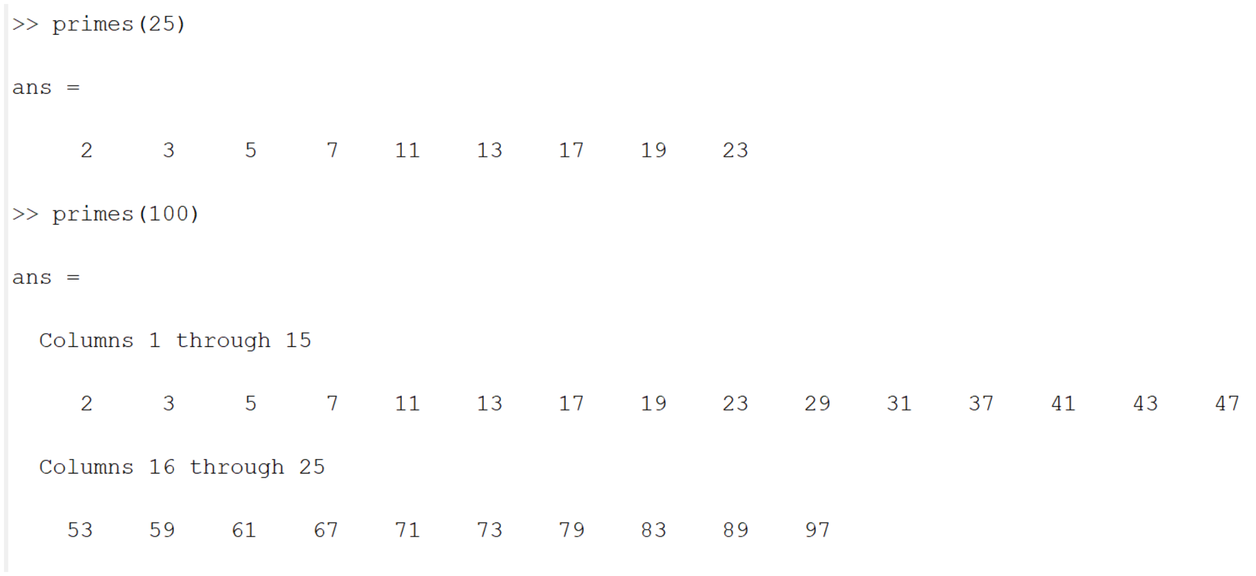
```
>> lcm (100,10)

ans =

   100

>> a

a =

    10    20
    50    90

>> lcm (10, a)

ans =

    10    20
    50    90
```

Now, isprime function is used to check whether a number is prime or not.



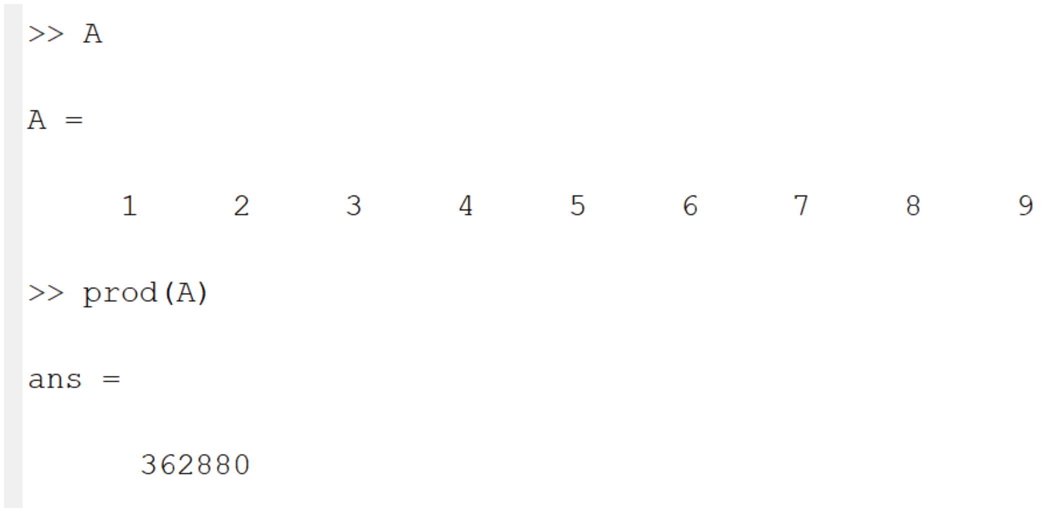
```
>> a

a =

    10    20
    50    90

>> isprime(a)

ans =

  2×2 logical array

   0   0
   0   0

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

A =

     1     2     3     4     5     6     7     8     9

>> isprime(A)

ans =

  1×9 logical array

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

primes(25) will generate all the prime numbers that are less than or equal to 25.



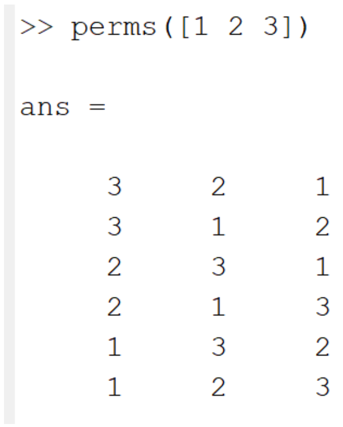
```
>> primes(25)

ans =

     2     3     5     7    11    13    17    19    23

>> primes(100)

ans =

  Columns 1 through 15

     2     3     5     7    11    13    17    19    23    29    31    37    41    43    47

  Columns 16 through 25

    53    59    61    67    71    73    79    83    89    97
```

Now prod() function will return multiplication of all the elements of the given matrix.



```
>> A

A =

     1     2     3     4     5     6     7     8     9

>> prod(A)

ans =

      362880
```

perms() function compute permutation

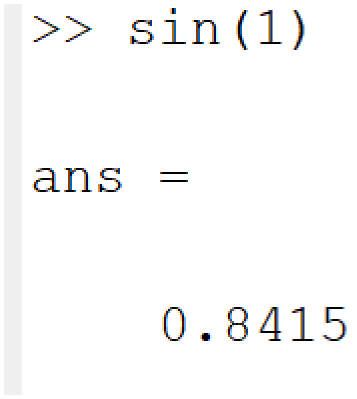


```
>> perms([1 2 3])

ans =

     3     2     1
     3     1     2
     2     3     1
     2     1     3
     1     3     2
     1     2     3
```

```
>> perms([1 2; 3 4])

ans =

     4     2     3     1
     4     2     1     3
     4     3     2     1
     4     3     1     2
     4     1     2     3
     4     1     3     2
     2     4     3     1
     2     4     1     3
     2     3     4     1
     2     3     1     4
     2     1     4     3
     2     1     3     4
     3     4     2     1
     3     4     1     2
     3     2     4     1
     3     2     1     4
     3     1     4     2
     3     1     2     4
     1     4     2     3
     1     4     3     2
     1     2     4     3
     1     2     3     4
     1     3     4     2
     1     3     2     4
```

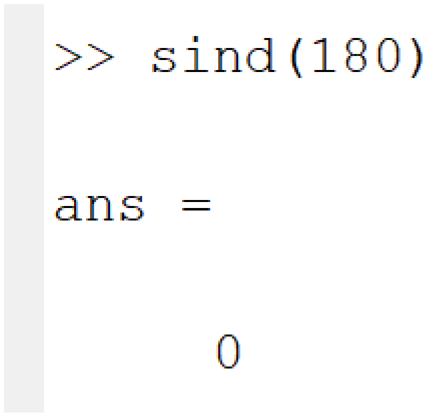
See, perms functions does not care about row and column. It takes all the numbers and do permutation.

Trigonometric Math Functions



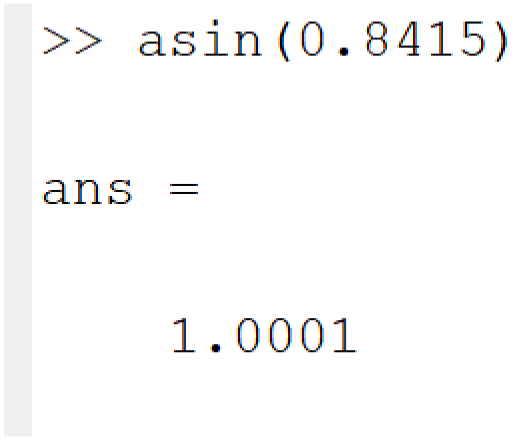
```
>> sin(1)

ans =

    0.8415
```

You get output in radian. If you want it in degree use:



```
>> sind(180)

ans =

     0
```

Now to use inverse sin



```
>> asin(0.8415)

ans =

    1.0001
```

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

a =

     1     2     3
     4     5     6

>> sin(a)

ans =

    0.8415    0.9093    0.1411
   -0.7568   -0.9589   -0.2794
```

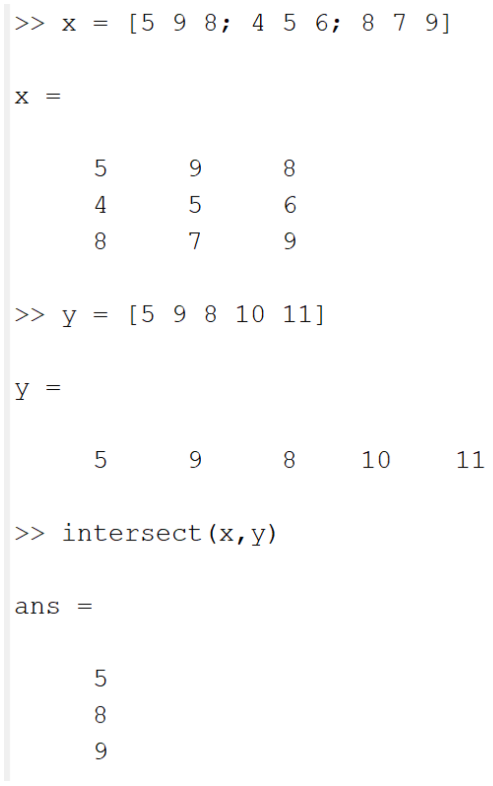
Similarly, cos, cosd, acos, tan, tand, atan, sec, secd, asec, csc, cscd, acsc

```
>> rad2deg(3.1416)

ans =

  180.0004

>> deg2rad(180.0004)

ans =

    3.1416
```

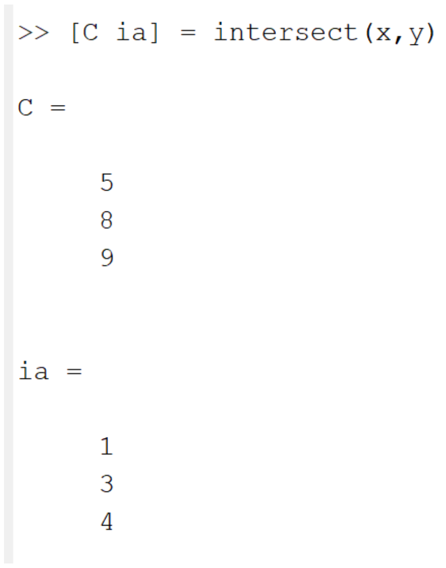
Find more:

Help → Documentation →MATLAB →Mathematics →Elementary Math → Trigonometry

Set Operations:



```
>> x = [5 9 8; 4 5 6; 8 7 9]

x =

     5     9     8
     4     5     6
     8     7     9

>> y = [5 9 8 10 11]

y =

     5     9     8    10    11

>> intersect(x,y)

ans =

     5
     8
     9
```



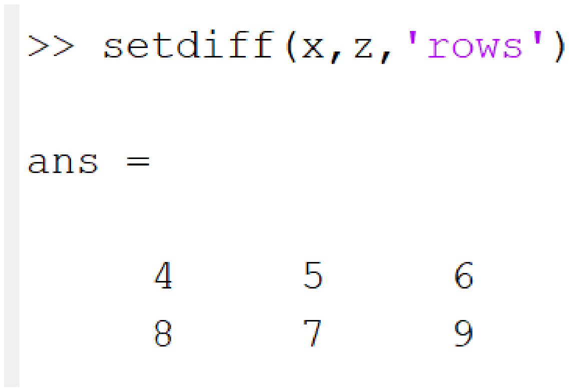
```
>> [C ia] = intersect(x,y)

C =

     5
     8
     9


ia =

     1
     3
     4
```

```
>> x

x =

     5     9     8
     4     5     6
     8     7     9

>> z = [5 9 8]

z =

     5     9     8

>> [a b] = intersect(x, z, 'rows')

a =

     5     9     8


b =

     1
```

Now in the above, it is comparing the rows. Before it compare considering all the elements.

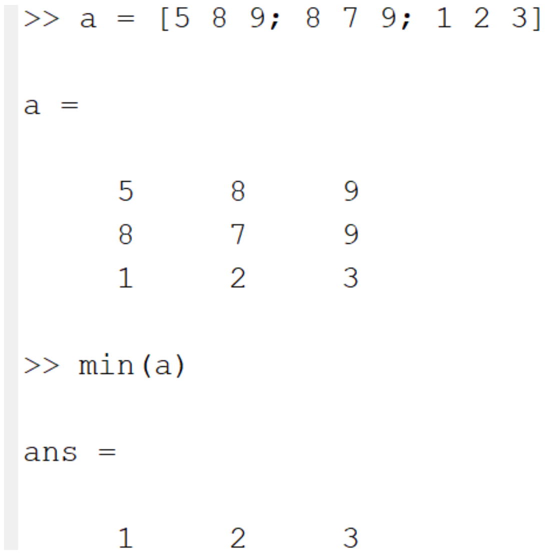
```
>> x

x =

     5     9     8
     4     5     6
     8     7     9

>> y

y =

     5     9     8    10    11

>> union(x,y)

ans =

     4
     5
     6
     7
     8
     9
    10
    11
```

```
>> x

x =

     5     9     8
     4     5     6
     8     7     9

>> z

z =

     5     9     8

>> union(x,z,'rows')

ans =

     4     5     6
     5     9     8
     8     7     9
```



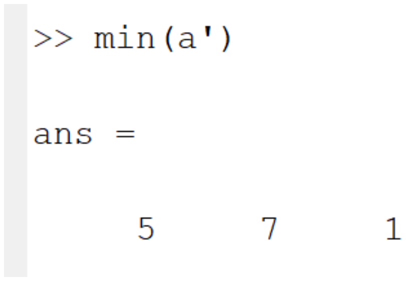
```
>> setdiff(x,z,'rows')

ans =

     4     5     6
     8     7     9
```

Statistical Functions:



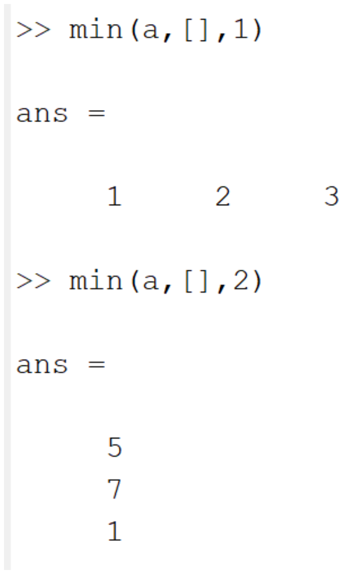
```
>> a = [5 8 9; 8 7 9; 1 2 3]

a =

     5     8     9
     8     7     9
     1     2     3

>> min(a)

ans =

     1     2     3
```

Here we are getting column wise minimum value. What if I want row wise.

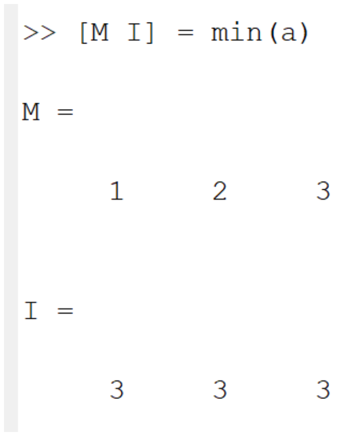


```
>> min(a')

ans =

     5     7     1
```

Another way:

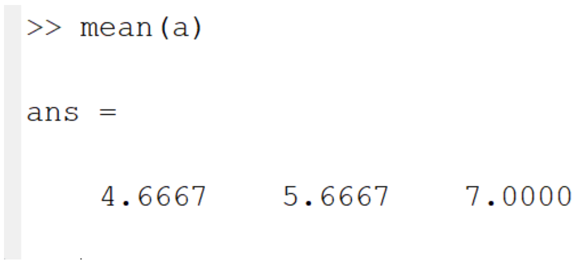


```
>> min(a,[],1)

ans =

     1     2     3

>> min(a,[],2)

ans =

     5
     7
     1
```



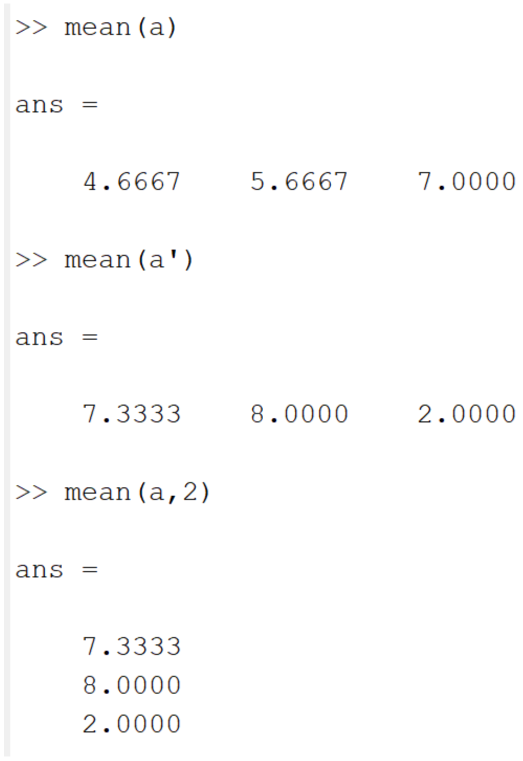
```
>> [M I] = min(a)

M =

     1     2     3


I =

     3     3     3
```

```
>> a

a =

     5     8     9
     8     7     9
     1     2     3

>> max(a)

ans =

     8     8     9

>> max(a,[],2)

ans =

     9
     9
     3

>> [M I] = max(a)

M =

     8     8     9


I =

     2     1     1
```



```
>> mean(a)

ans =

    4.6667    5.6667    7.0000
```

Mean function also give value as per the column. But if we want as a row, we either do transpose, or we can add additional arguments.

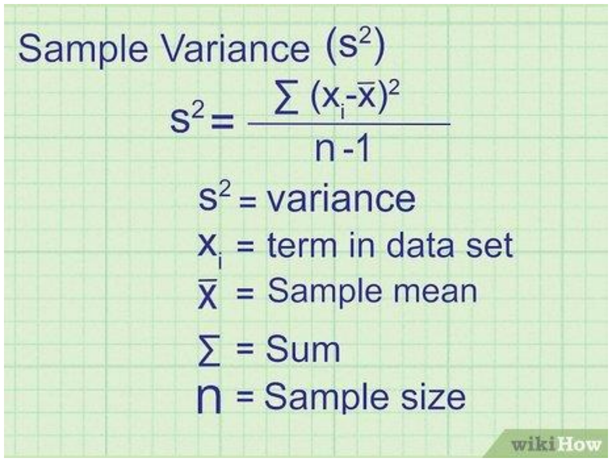


```
>> mean(a)

ans =

    4.6667    5.6667    7.0000

>> mean(a')

ans =

    7.3333    8.0000    2.0000

>> mean(a,2)

ans =

    7.3333
    8.0000
    2.0000
```

```
>> var(a)

ans =

   12.3333   10.3333   12.0000

>> var(a')

ans =

    4.3333    1.0000    1.0000

>> var(a,0,2)

ans =

    4.3333
    1.0000
    1.0000

>> var(a,1,2)

ans =

    2.8889
    0.6667
    0.6667
```

var(a, 1, 2) applies the N-1 normalization, often used in sample variance calculations, while var(a, 0, 2) does not apply this normalization and gives the variance without division by N-1.

The term "N-1 normalization" refers to dividing by (N-1) rather than N when computing the variance, where N is the number of data points. This adjustment is often used when calculating the sample variance. The formula for the sample variance (s^2) is given by:



```
>> std(a)

ans =

    3.5119    3.2146    3.4641

>> std(a')

ans =

    2.0817    1.0000    1.0000

>> sum(a)

ans =

    14    17    21

>> sum(a')

ans =

    22    24     6

>> median(a)

ans =

     5     7     9

>> median(a')

ans =

     8     8     2
```

Percentile:

Suppose you have the following dataset of test scores: 60, 65, 70, 75, 80, 85, 90, 95.

Now, let's find the 25th percentile (P25), which is the value below which 25% of the data falls.

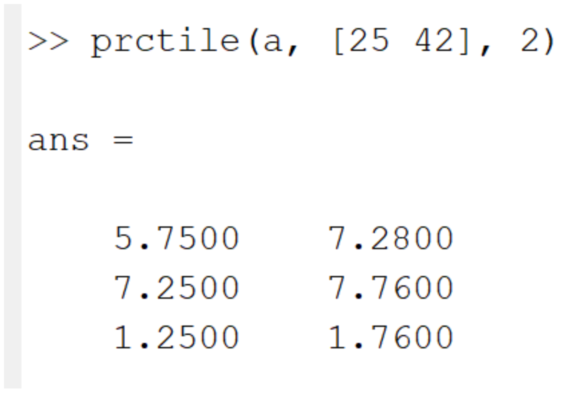
1. **Sort the data:**
   Sort the scores in ascending order: 60, 65, 70, 75, 80, 85, 90, 95.
2. **Calculate the position:**
   Use the formula: $\text{Position} = \frac{P \times (N+1)}{100}$, where P is the percentile (25 in this case), and N is the number of data points (8 in our example).
   $\text{Position} = \frac{25 \times (8+1)}{100} = \frac{225}{100} = 2.25.$
   The position is between the 2nd and 3rd data points.
3. **Find the value:**
   Interpolate between the values at positions 2 and 3: $70 + 0.25 \times (75 - 70) = 71.25.$

So, the 25th percentile (P25) in this dataset is 71.25. This means that 25% of the test scores are below 71.25.

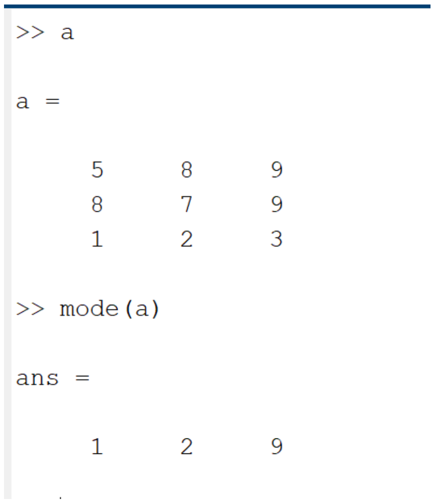
```
>> a

a =

     5     8     9
     8     7     9
     1     2     3

>> prctile(a,25)

ans =

    2.0000    3.2500    4.5000

>> prctile(a, [25 42])

ans =

    2.0000    3.2500    4.5000
    4.0400    5.8000    7.5600
```

This also operated on the columns by default.

For rows,



```
>> prctile(a, [25 42], 2)

ans =

    5.7500    7.2800
    7.2500    7.7600
    1.2500    1.7600
```



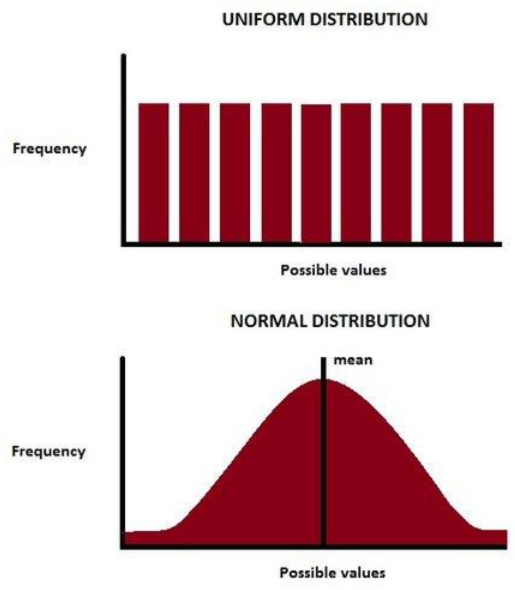
```
>> a

a =

     5     8     9
     8     7     9
     1     2     3

>> mode(a)

ans =

     1     2     9
```

The mode is the value that appears most often in a set of data values. Check column wise. For column 3, nine is the value appeared most.

Handling random numbers:

```
>> rand(5,1)

ans =

    0.4505
    0.0838
    0.2290
    0.9133
    0.1524

>> randn(5)

ans =

    0.9610   -1.2078   -0.4686   -2.0518    0.2820
    0.1240    2.9080   -0.2725   -0.3538    0.0335
    1.4367    0.8252    1.0984   -0.8236   -1.3337
   -1.9609    1.3790   -0.2779   -1.5771    1.1275
   -0.1977   -1.0582    0.7015    0.5080    0.3502
```

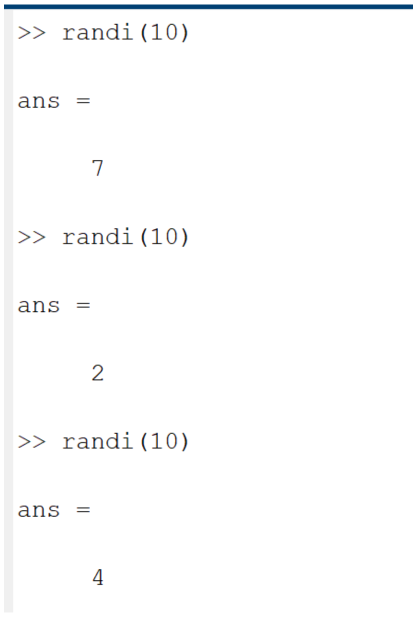
rand provides uniformly distributed random numbers between 0 and 1, while randn provides random numbers from a standard normal distribution (bell-shaped curve with mean 0 and standard deviation 1).



```
>> randperm(9)

ans =

     4     6     7     5     8     1     3     9     2

>> randperm(9)

ans =

     1     9     6     8     5     4     7     2     3

>> randperm(9,3)

ans =

     6     4     5

>> randperm(9,3)

ans =

     7     8     5
```

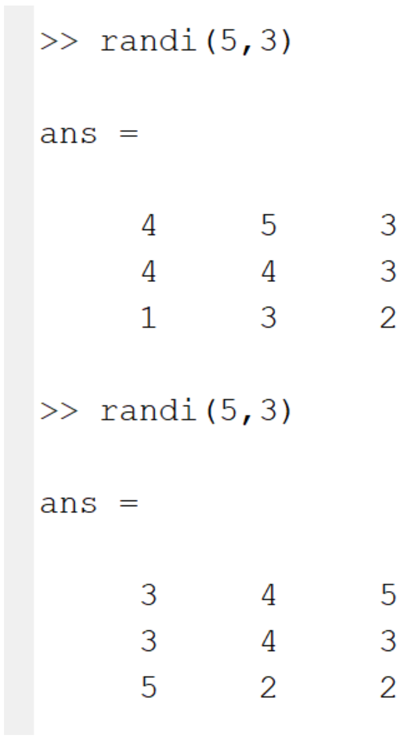
Here random numbers from 3 to 7

```
>> 2+randperm(5,3)

ans =

     5     4     7

>> 2+randperm(5,3)

ans =

     7     4     5
```



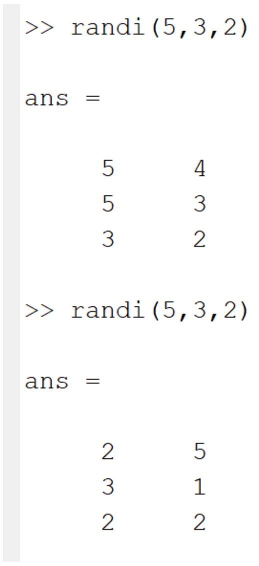
```
>> randi(10)

ans =

     7

>> randi(10)

ans =

     2

>> randi(10)

ans =

     4
```

Random 3/3 matrix whose values are in between 1 to 5



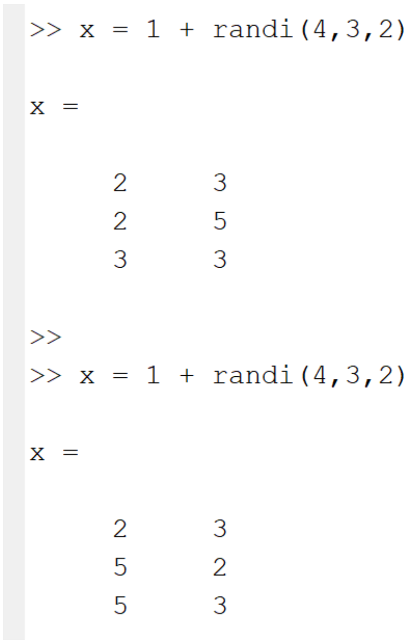
```
>> randi(5,3)

ans =

     4     5     3
     4     4     3
     1     3     2

>> randi(5,3)

ans =

     3     4     5
     3     4     3
     5     2     2
```

Random 3/2 matrix whose values are in between 1 to 5



```
>> randi(5,3,2)

ans =

     5     4
     5     3
     3     2

>> randi(5,3,2)

ans =

     2     5
     3     1
     2     2
```

Same, but getting values between 2 and 5

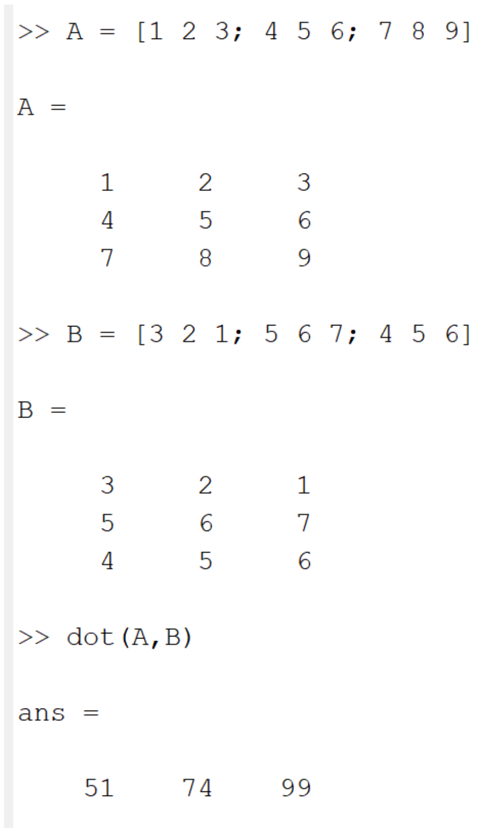


```
>> x = 1 + randi(4,3,2)

x =

     2     3
     2     5
     3     3

>>
>> x = 1 + randi(4,3,2)

x =

     2     3
     5     2
     5     3
```

Cross Product and Dot Product:

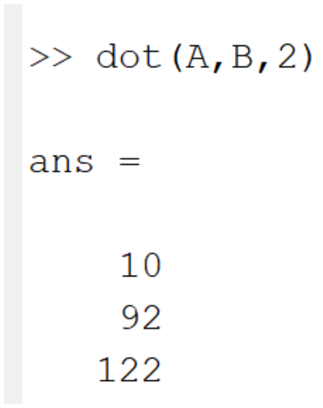
A Vector in MATLAB is represented by matrix on one row or one column.

```
>> c = [4 5 6]

c =

     4     5     6

>> d = [5 5 5]

d =

     5     5     5

>> dot(c,d)

ans =

    75

>> c.*d

ans =

    20    25    30

>> sum(c.*d)

ans =

    75
```



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

A =

     1     2     3
     4     5     6
     7     8     9

>> B = [3 2 1; 5 6 7; 4 5 6]

B =

     3     2     1
     5     6     7
     4     5     6

>> dot(A,B)

ans =

    51    74    99
```

Here column wise dot product is being done.



```
>> dot(A,B,2)

ans =

    10
    92
   122
```

Now for cross product, we can only do that when the length of the vectors are equal to 3.

In MATLAB, the cross product is a mathematical operation defined for 3D vectors. The vectors must have exactly three elements along the specified dimension for the cross product to be valid. The reason behind this requirement is that the cross product is specifically defined in three-dimensional space.

The dot product works in any number of dimensions, but the cross product only works in 3D. The dot product measures how much two vectors point in the same direction, but the cross product measures how much two vectors point in different directions.

```
>> A

A =

     1     2     3
     4     5     6
     7     8     9

>> B

B =

     3     2     1
     5     6     7
     4     5     6

>> cross(A,B)

ans =

   -19   -23   -27
    17     6    -9
    -7     2    15

>> cross(A,B,2)

ans =

    -4     8    -4
    -1     2    -1
     3    -6     3
```

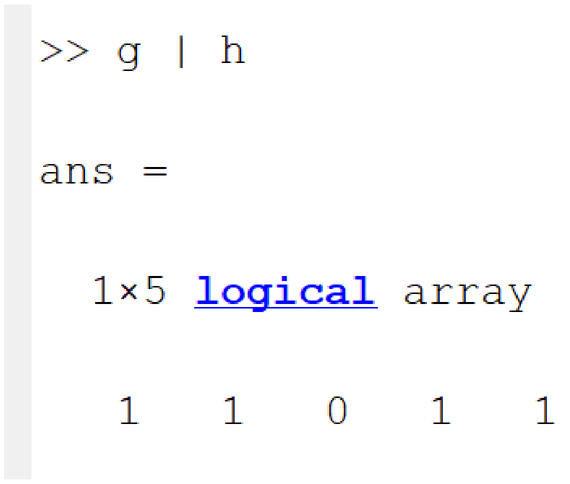
Basic Logical Operation:

```
g =

     1     5     8     7     0

>> h = [1 1 0 2 3]

h =

     1     1     0     2     3

>> and(g,h)

ans =

  1×5 logical array

   1   1   0   1   0

>> g & h

ans =

  1×5 logical array

   1   1   0   1   0
```

Here in the below operation is done with only 0 value or 1 value

```
>> and(g,0)

ans =

  1×5 logical array

   0   0   0   0   0

>> and(g,1)

ans =

  1×5 logical array

   1   1   1   1   0
```

```
>> g

g =

     1     5     8     7     0

>> h

h =

     1     1     0     2     3

>> or(g,h)

ans =

  1×5 logical array

   1   1   1   1   1

>> g(3) = 0

g =

     1     5     0     7     0

>> or(g,h)

ans =

  1×5 logical array

   1   1   0   1   1
```



```
>> g | h

ans =

  1×5 logical array

   1   1   0   1   1
```

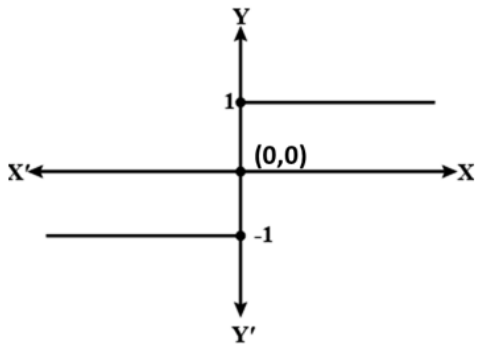
```
>> g

g =

     1     5     0     7     0

>> not(g)

ans =

  1×5 logical array

   0   0   1   0   1
```

```
>> -g

ans =

    -1    -5     0    -7     0
```

Sign and absolute functions:

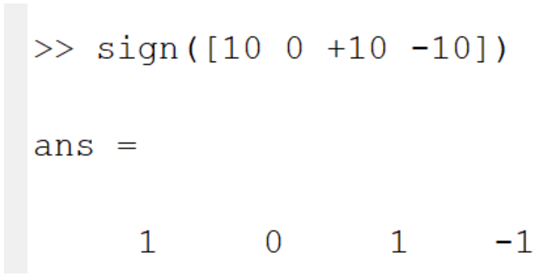
$$Sgn(x) = \begin{cases} -1\,, & if\ x < 0 \\ 0, & if, x = 0 \\ 1, & if, x > 0 \end{cases}$$

Graph of Signum Function

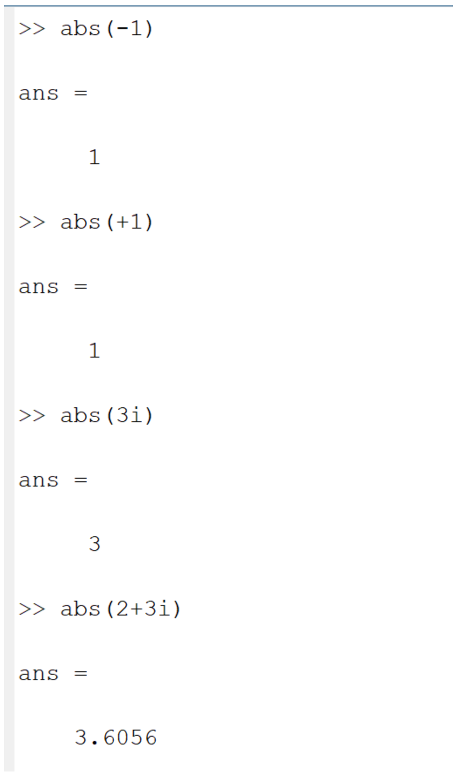


```
>> sign(+6)

ans =

     1

>> sign(0)

ans =

     0

>> sign(-1)

ans =

    -1

>> sign(-6)

ans =

    -1
```

If the input is greater than 0, you get 1, if it is lesser than 0 you get -1, if it is 0, you get 0.



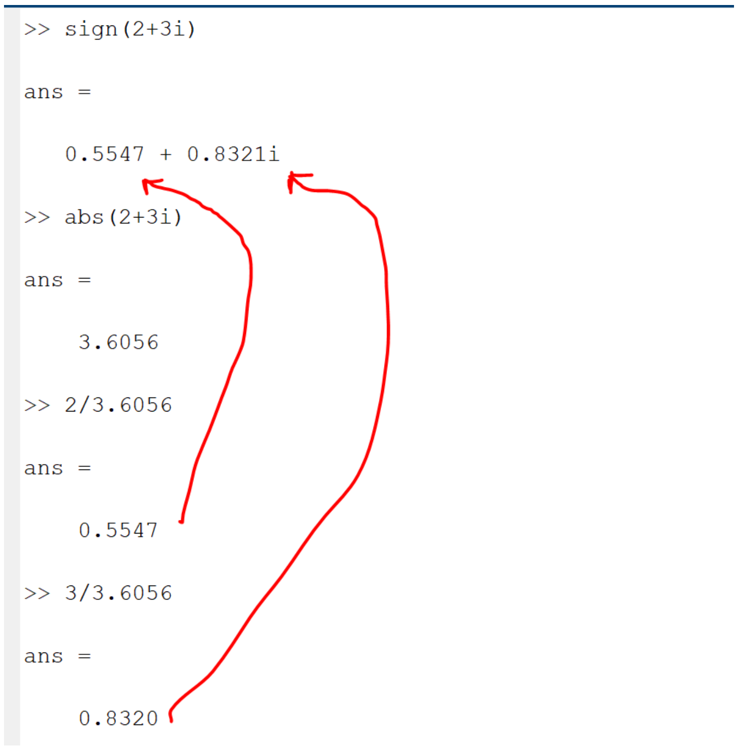
```
>> sign([10 0 +10 -10])

ans =

     1     0     1    -1
```



```
>> abs(-1)

ans =

     1

>> abs(+1)

ans =

     1

>> abs(3i)

ans =

     3

>> abs(2+3i)

ans =

    3.6056
```

Whatever the sign, it gives you an absolute value. For complex number it will give the magnitude.

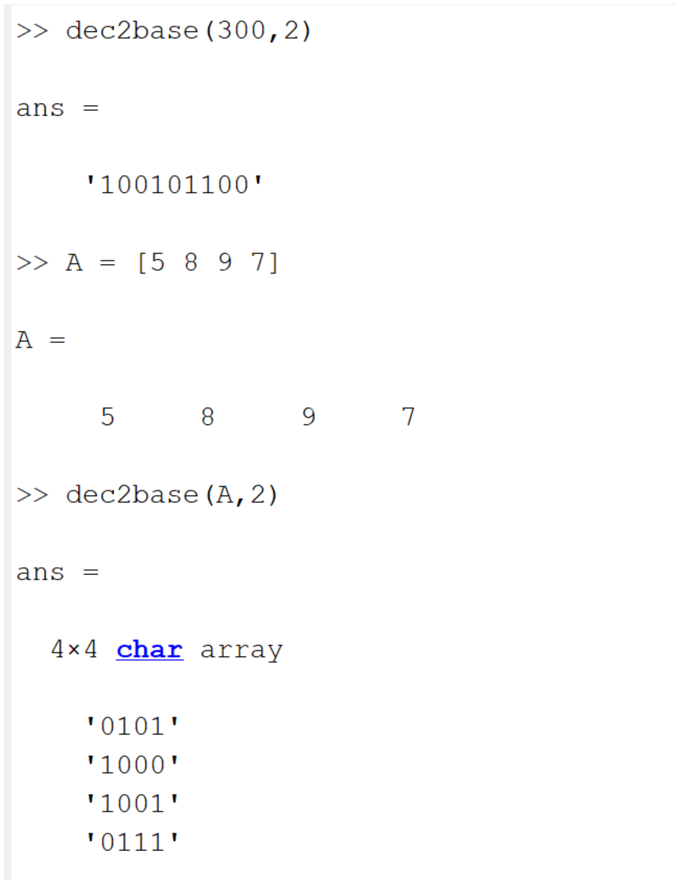
See below sign function of a complex number issue:



```
>> sign(2+3i)

ans =

   0.5547 + 0.8321i

>> abs(2+3i)

ans =

    3.6056

>> 2/3.6056

ans =

    0.5547

>> 3/3.6056

ans =

    0.8320
```

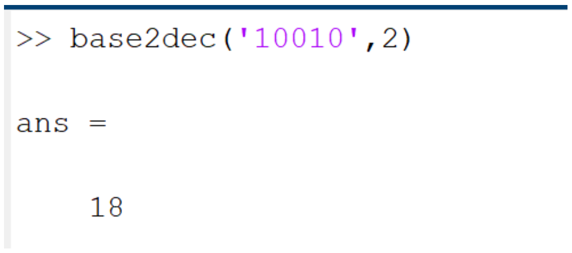
Converting numbers between different bases:

Decimal Number to Binary Number: (2 here is for binary)



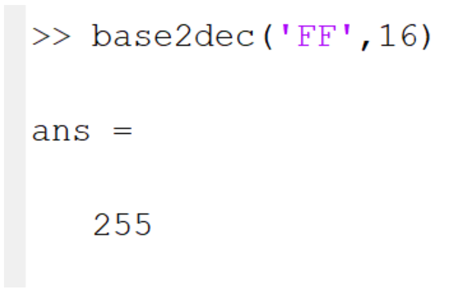
```
>> dec2base(300,2)

ans =

    '100101100'

>> A = [5 8 9 7]

A =

     5     8     9     7

>> dec2base(A,2)

ans =

  4×4 char array

    '0101'
    '1000'
    '1001'
    '0111'
```

Now, Binary to Decimal. You have input the binary number as character. Here by 2, we mean that the input is binary.



```
>> base2dec('10010',2)

ans =

    18
```

Now, Hexadecimal to decimal. Here by 16, we mean that the input is hexadecimal.



```
>> base2dec('FF',16)

ans =

   255
```

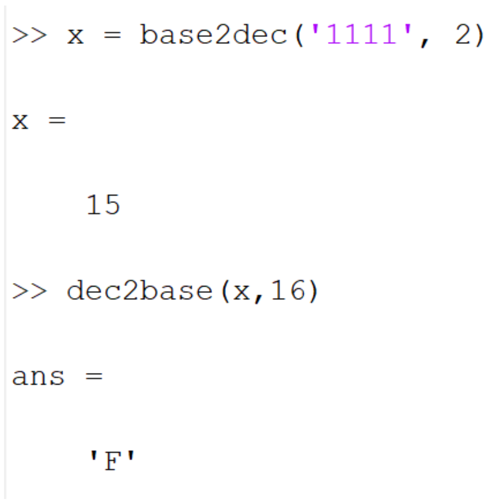
We just saw base 2 and base 16.

The allowable basis in MATLAB is base 2 to base 16.

base2dec() function actually helps with converting a number of any base to decimal number. In the argument of the function you define the base. Say, base2dec(‘10010’, 2), here 2 is meaning binary. So, this function will cover binary (base = 2) to decimal.

This function is very general.

Now, let’s convert a binary number to a hexa number. In this case you will have to first convert binary to decimal. Then next, decimal to hexa.

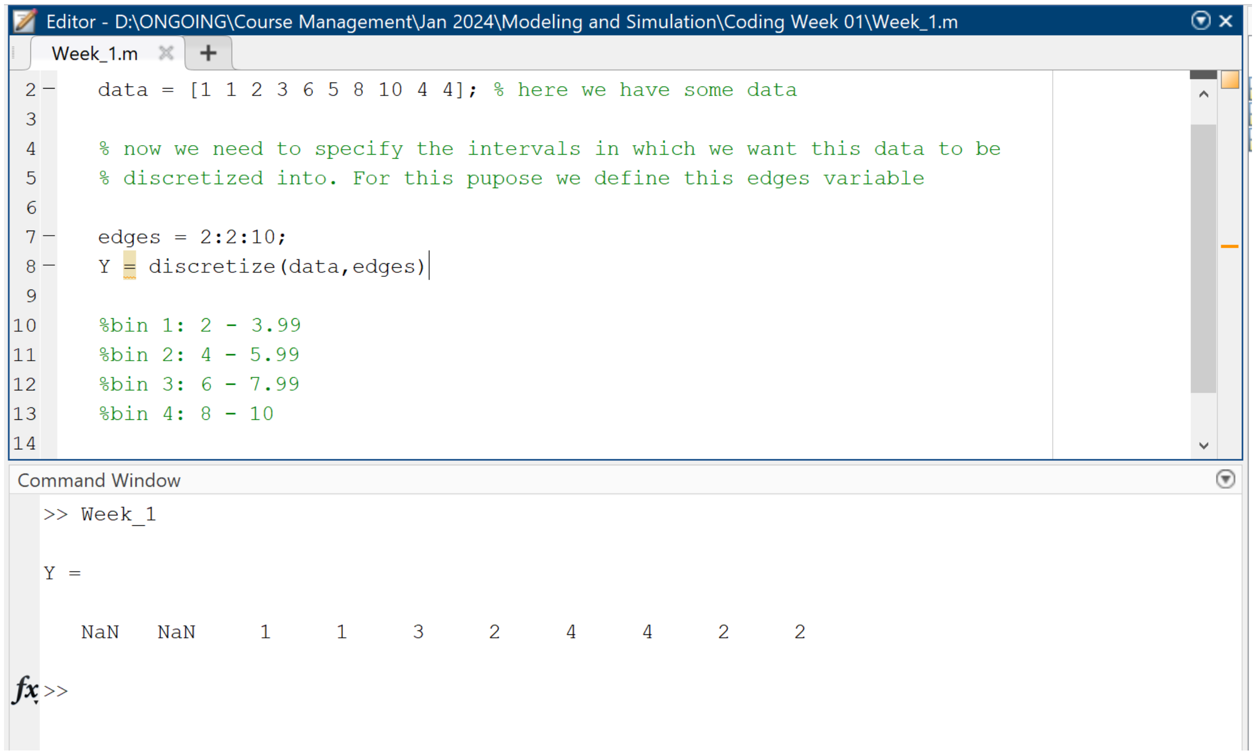


```
>> x = base2dec('1111', 2)

x =

    15

>> dec2base(x,16)

ans =

    'F'
```

Following this way, any base can be converted to different base.

Data Discretization:

Data discretization is a process used in data preprocessing and analysis to convert continuous data into discrete form. In other words, it involves dividing a range of continuous values into intervals or bins. This transformation is particularly useful when dealing with continuous numerical data in machine learning, statistics, and data analysis tasks


Editor - D:\ONGOING\Course Management\Jan 2024\Modeling and Simulation\Coding Week 01\Week_1.m
Week_1.m
Command Window

```
data = [1 1 2 3 6 5 8 10 4 4]; % here we have some data

% now we need to specify the intervals in which we want this data to be
% discretized into. For this pupose we define this edges variable

edges = 2:2:10;
Y = discretize(data,edges)

%bin 1: 2 - 3.99
%bin 2: 4 - 5.99
%bin 3: 6 - 7.99
%bin 4: 8 - 10
```

```
>> Week_1

Y =

   NaN   NaN     1     1     3     2     4     4     2     2

>>
```

We need to specify the number of bins or intervals, into which we want our data to be discretized into. The variable edge is going to provide that information to us.

edges = 2:2:10;

means, it will contain value starting from the value of 2, and then it will skip one value, then it will have value of 4, then 6, then 8 and finally value of 10.

The first two values in the edge, defines the values for bin number one

bin 1: 2 – 3.99

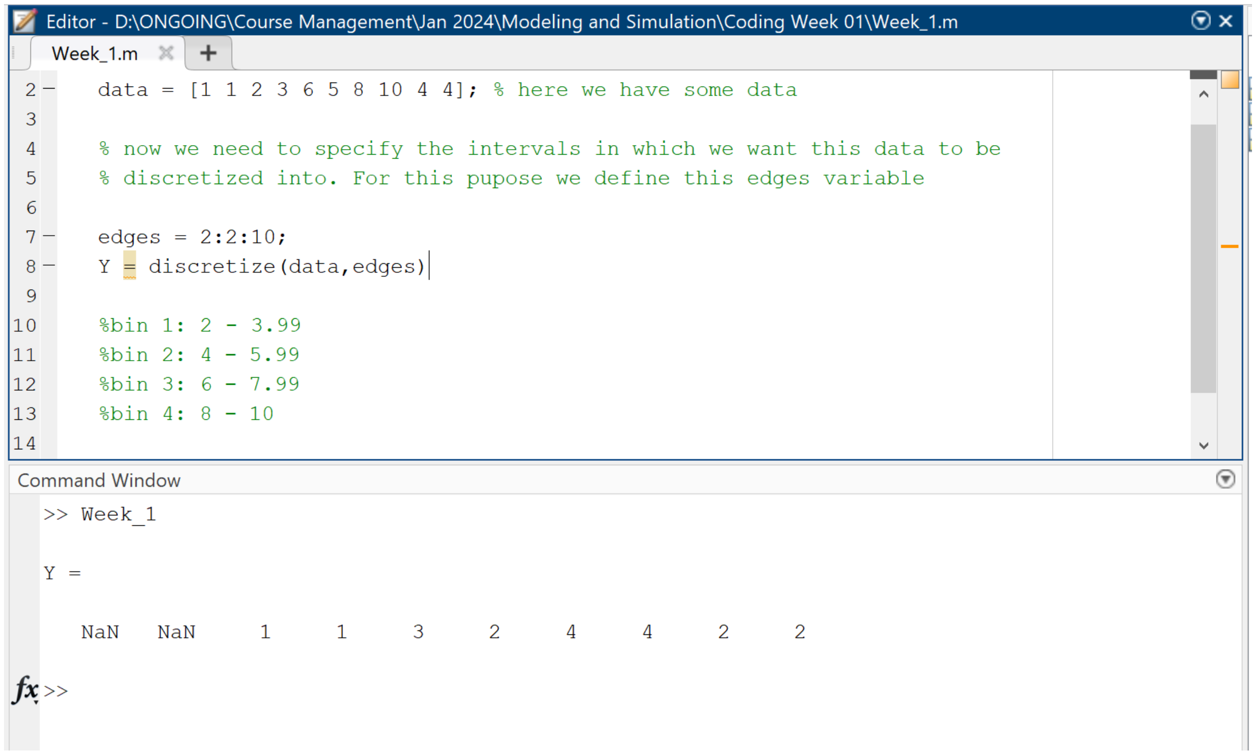
seems value 4 is not included.

bin 2: 4 – 5.99

bin 3: 6 – 7.99

bin 4: 8 – 10

here, except for the very last bin, we note that other bin does not include the right edge value. Using these rules the discretize function will discretize the data.


Editor - D:\ONGOING\Course Management\Jan 2024\Modeling and Simulation\Coding Week 01\Week_1.m
Week_1.m
Command Window

```
data = [1 1 2 3 6 5 8 10 4 4]; % here we have some data

% now we need to specify the intervals in which we want this data to be
% discretized into. For this pupose we define this edges variable

edges = 2:2:10;
Y = discretize(data,edges)

%bin 1: 2 - 3.99
%bin 2: 4 - 5.99
%bin 3: 6 - 7.99
%bin 4: 8 - 10

```

```
>> Week_1

Y =

   NaN   NaN     1     1     3     2     4     4     2     2

>>
```

Now, in the output, NaN (Not a Number) for 1 because it does not fall into any of the bin.

data = [1 1 2 3 6 5 8 10 4 4]

see,

1 does not fall into any bin - NaN

1 does not fall into any bin - NaN

2 fall into bin 1

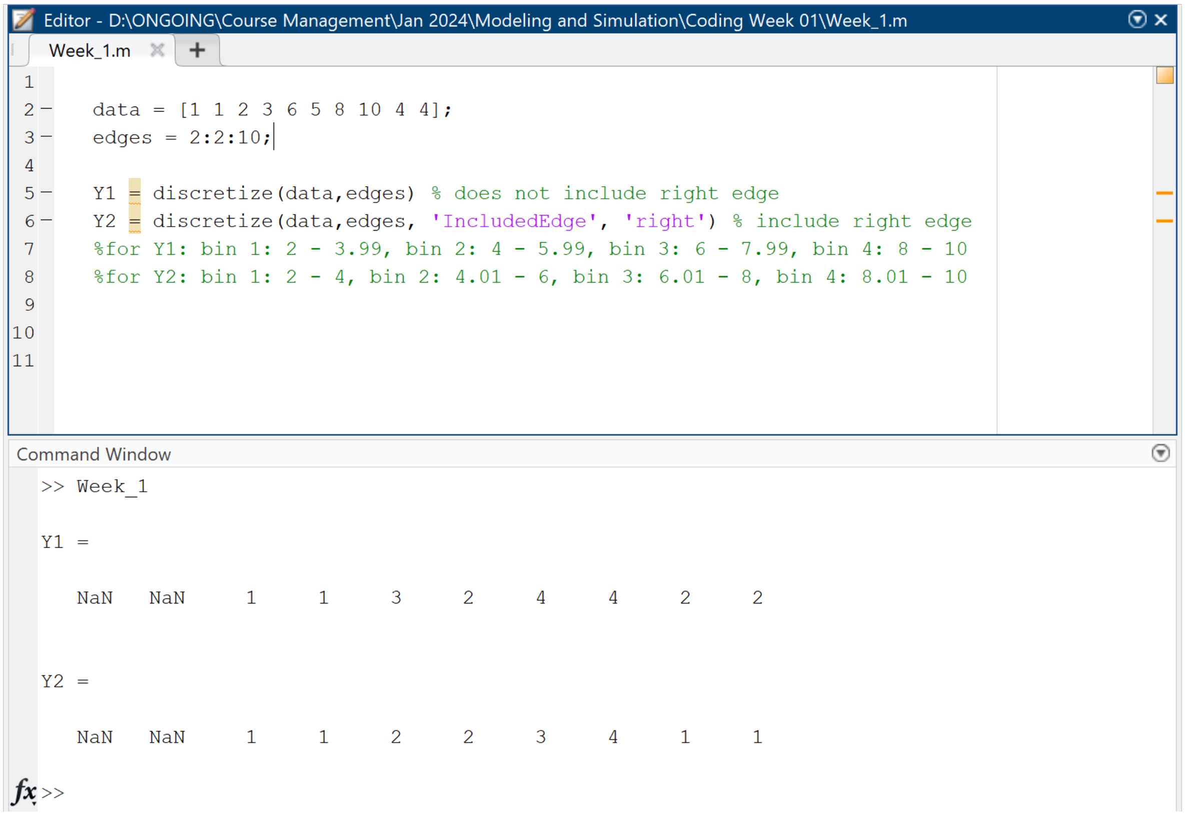
3 fall into bin 1

6 fall into bin 3

……

See, this is how the output Y is shown.

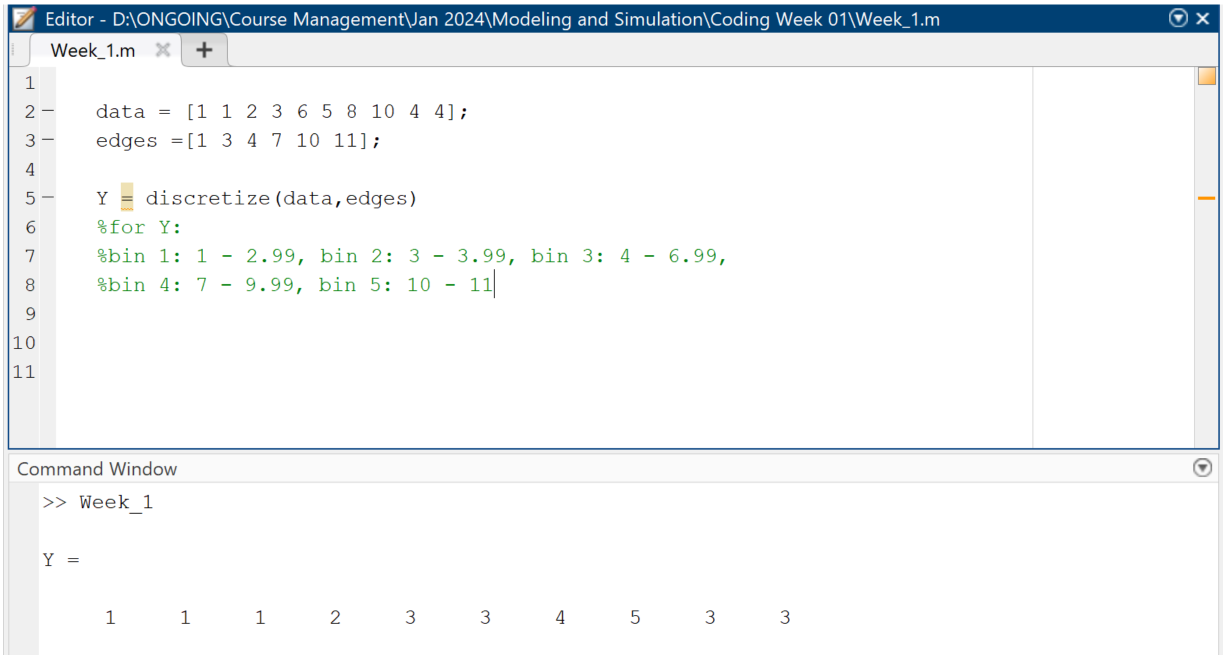
By default discretize function does not include the right edge. But what if I want to include the right edge. See below:


Editor - D:\ONGOING\Course Management\Jan 2024\Modeling and Simulation\Coding Week 01\Week_1.m
Week_1.m

```

data = [1 1 2 3 6 5 8 10 4 4];
edges = 2:2:10;

Y1 = discretize(data,edges) % does not include right edge
Y2 = discretize(data,edges, 'IncludedEdge', 'right') % include right edge
%for Y1: bin 1: 2 - 3.99, bin 2: 4 - 5.99, bin 3: 6 - 7.99, bin 4: 8 - 10
%for Y2: bin 1: 2 - 4, bin 2: 4.01 - 6, bin 3: 6.01 - 8, bin 4: 8.01 - 10
```

```
>> Week_1

Y1 =

   NaN   NaN     1     1     3     2     4     4     2     2


Y2 =

   NaN   NaN     1     1     2     2     3     4     1     1

>>
```

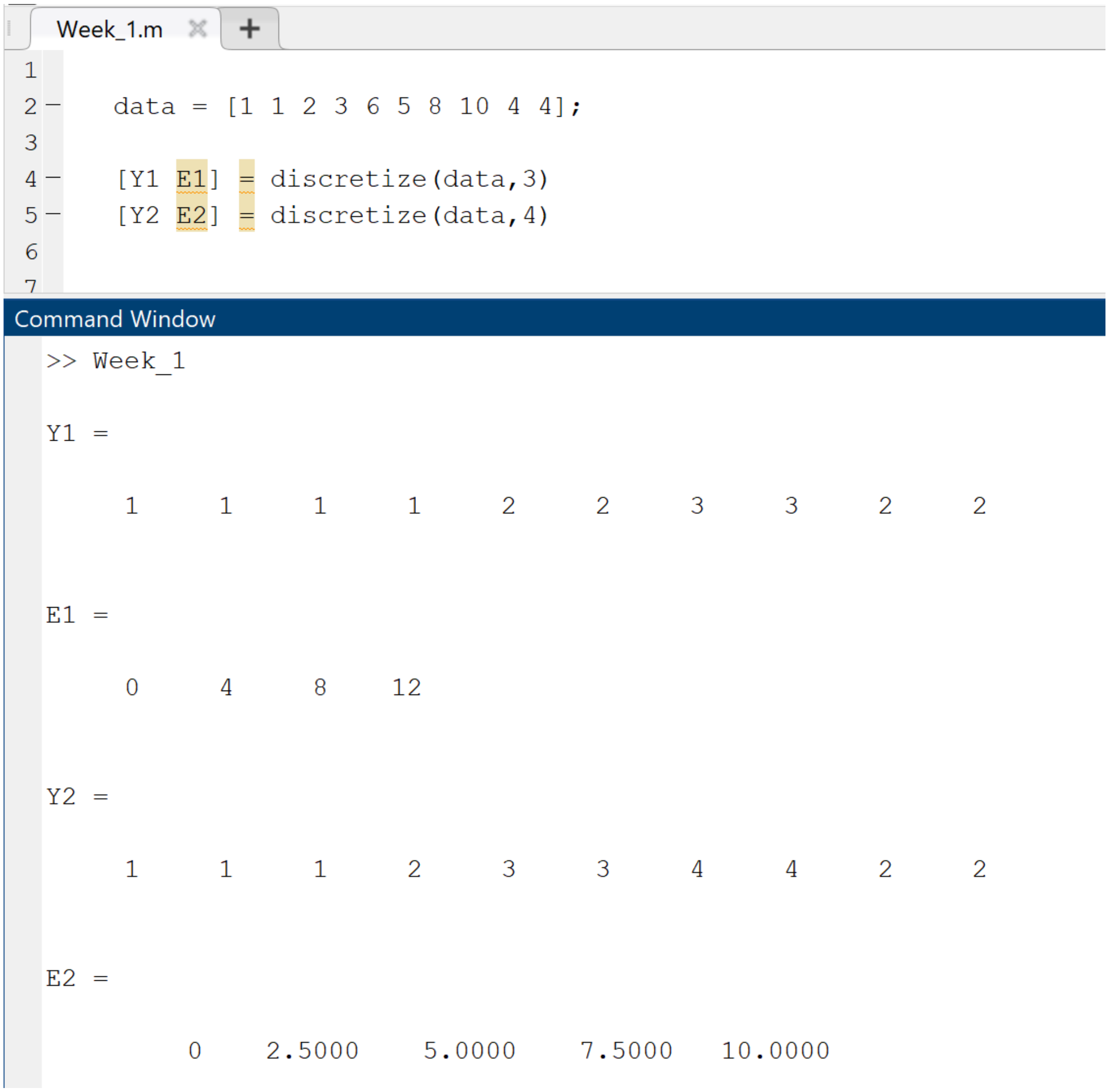
Now, we can also specify edges as a matrix of values. edges does not need to be equally space values.


Editor - D:\ONGOING\Course Management\Jan 2024\Modeling and Simulation\Coding Week 01\Week_1.m
Week_1.m

```

data = [1 1 2 3 6 5 8 10 4 4];
edges =[1 3 4 7 10 11];

Y = discretize(data,edges)
%for Y:
%bin 1: 1 - 2.99, bin 2: 3 - 3.99, bin 3: 4 - 6.99,
%bin 4: 7 - 9.99, bin 5: 10 - 11
```

```
>> Week_1

Y =

     1     1     1     2     3     3     4     5     3     3
```

Say, I want to predefine the number of bins. And MATLAB will figure out the rest.



```
data = [1 1 2 3 6 5 8 10 4 4];

[Y1 E1] = discretize(data,3)
[Y2 E2] = discretize(data,4)
```

```
>> Week_1

Y1 =

     1     1     1     1     2     2     3     3     2     2


E1 =

     0     4     8    12


Y2 =

     1     1     1     2     3     3     4     4     2     2


E2 =

         0    2.5000    5.0000    7.5000   10.0000
```

Same dataset is discretized here with 3 bins or 4 bins.

Y provide the info regarding which data is in which bin. And, E variable here return the information regarding how the bins are formulated. Here, MATLAB is defining the bins based on minimum and maximum value. By default, it will work with the left edge, but with argument we can make it right edge.