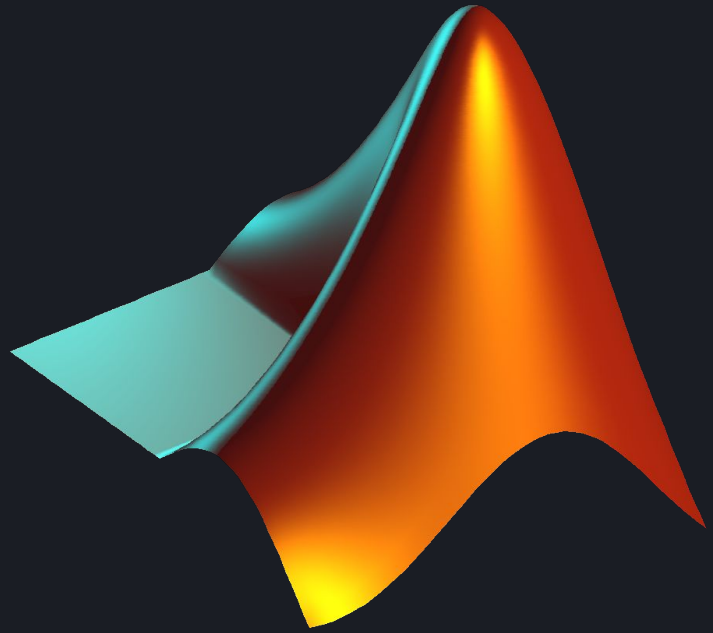
Matlab Basics Module

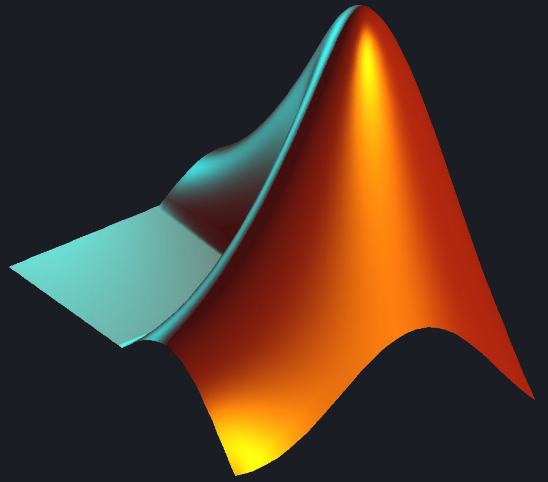


Model-Based Development Program

Matlab Basics Module

Objectives

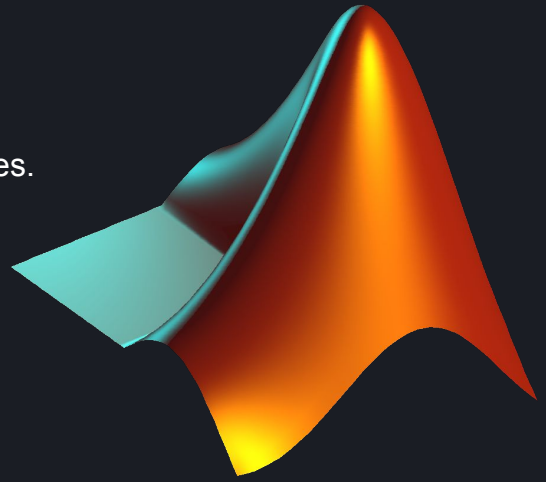
- This module aims to equip participants with foundational MATLAB skills.
- Participants will gain practical knowledge for effective use.



Matlab Basics Module

Module Content

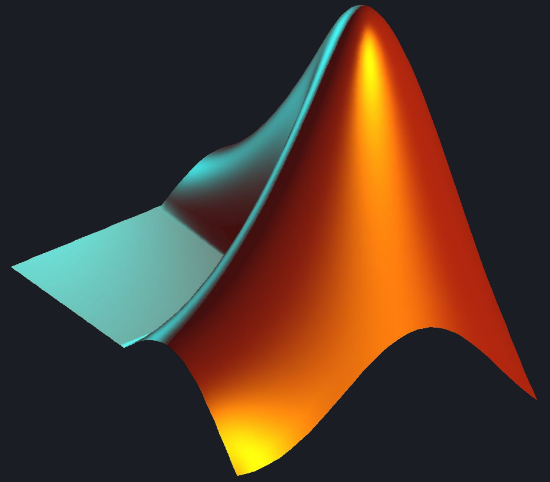
- Introduction to MATLAB
- Develop programming skills and proficiency in working with functions.
- Explore data types, plotting, fitting, logical indexing, and preallocation strategies.



Introduction to MATLAB

Session Content

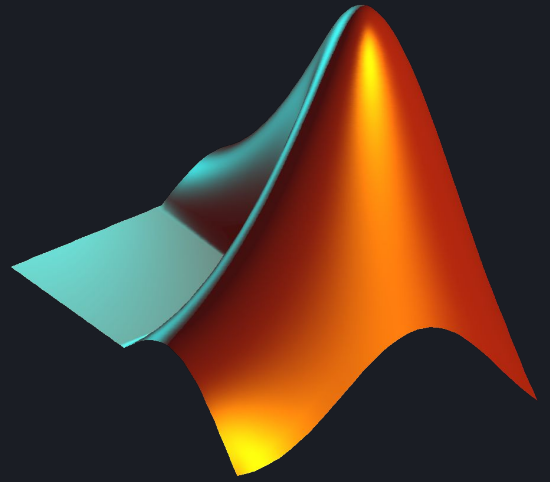
- what is a MATLAB?
- Matrix Creation & Generation Data
- Matrix Indexing
- Mathematical Operation
- Concatenation
- Repating



Introduction To MATLAB

What is a MATLAB?

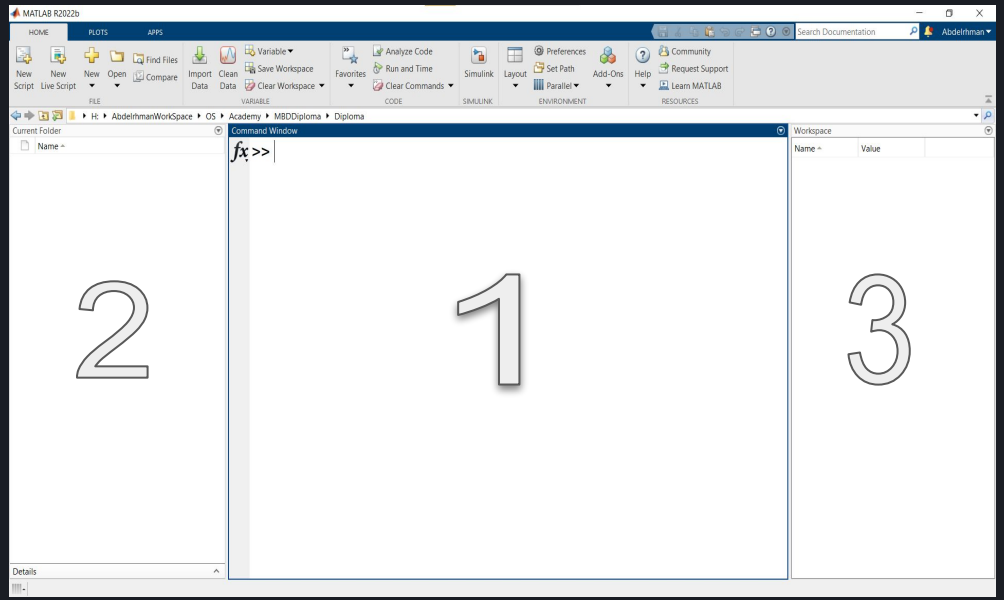
- **MATrix LABoratory**
- **is a powerful software tool for**
  - Performing **mathematical computations** and signal processing
  - Algorithm Development
  - **Modeling** and **Simulation** for physical systems and phenomena
  - **Analyzing** and **visualizing** data



Introduction To MATLAB

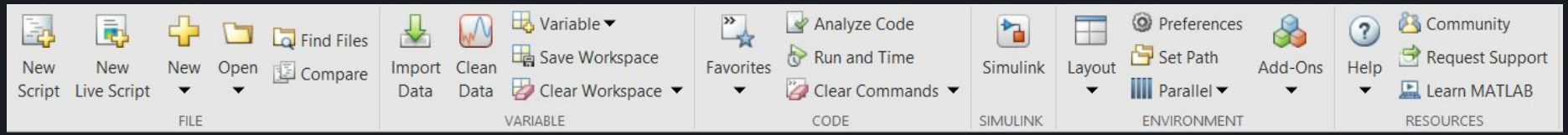
The MATLAB Environment Window

1. **Command Window**
   The Matlab command window serves as the interface through which the user interacts with Matlab, allowing for the input of command lines, instructions, and communication with the software.
2. **Current Folder**
   This window displays the presently open file within the software
3. **Workspace**
   This window displays variable names, formats, and allows workspace saving.



Introduction To MATLAB

The Matlab Command Bar

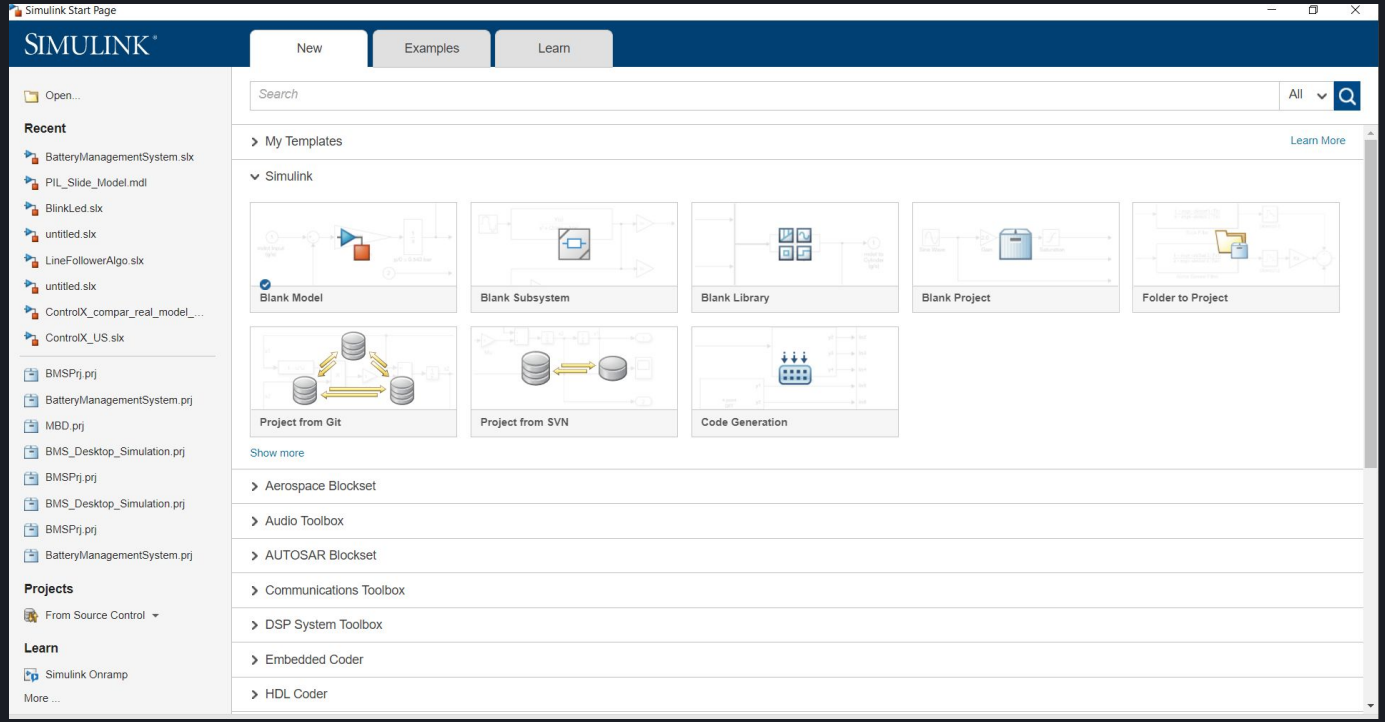


- To Lunch Simulink Click on

- To Lunch Simulink write "**simulink**" in Command window

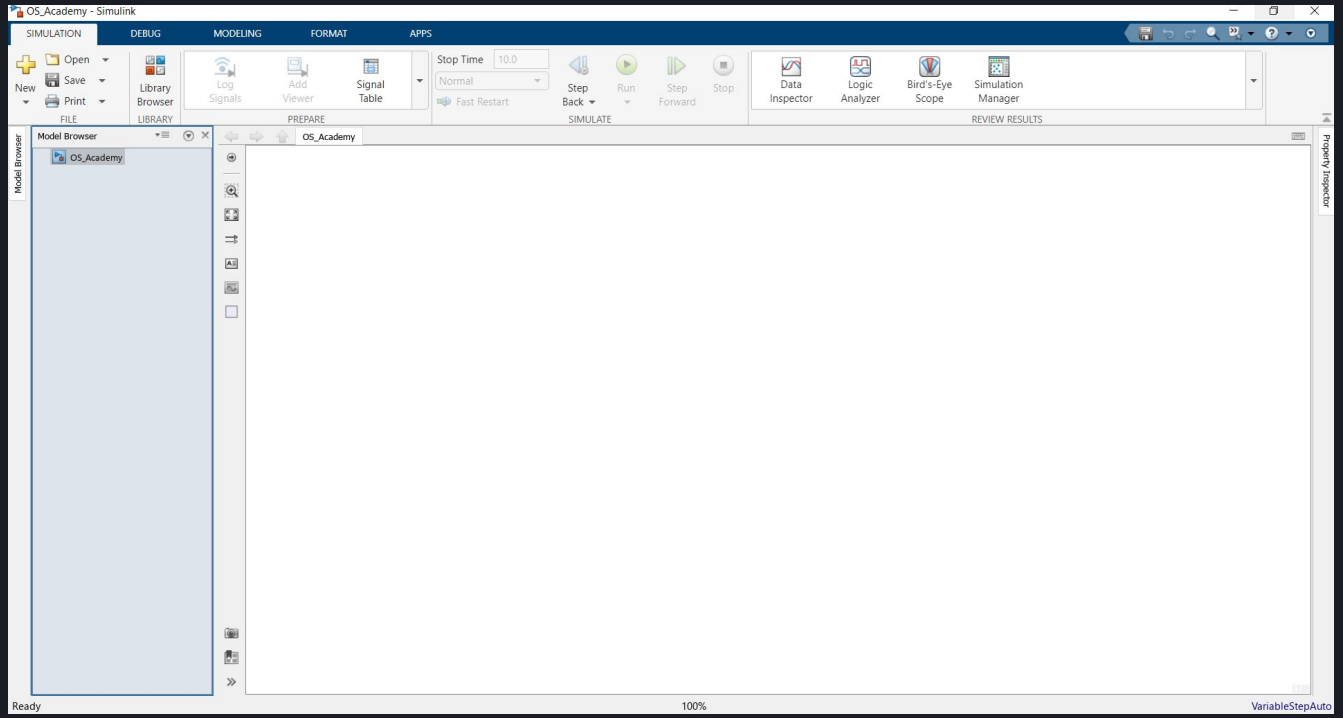
Introduction To MATLAB

The Simulink Environment Window



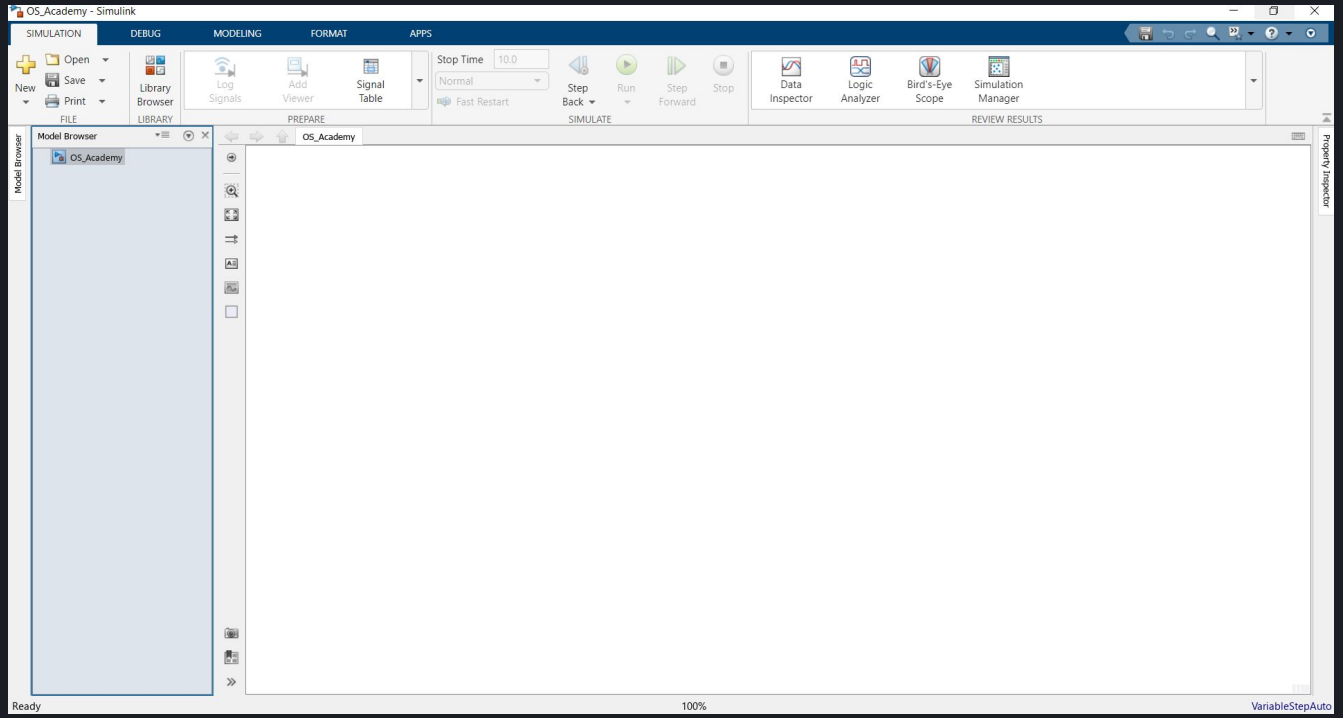
Introduction To MATLAB

The Simulink Environment Window



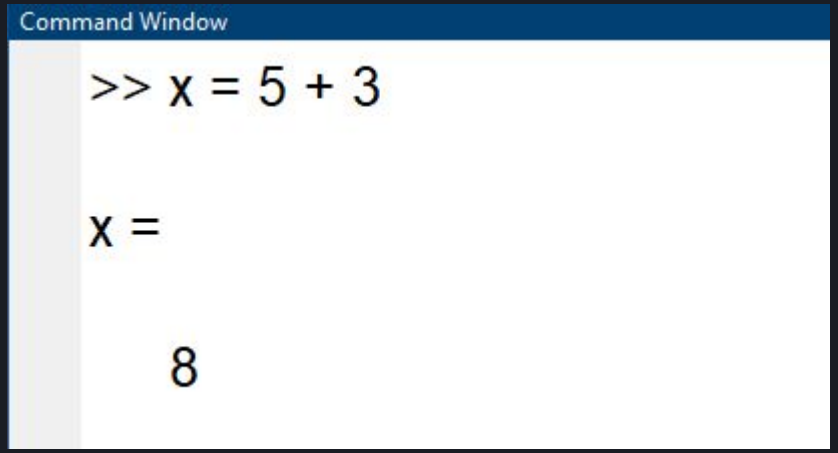
Introduction To MATLAB

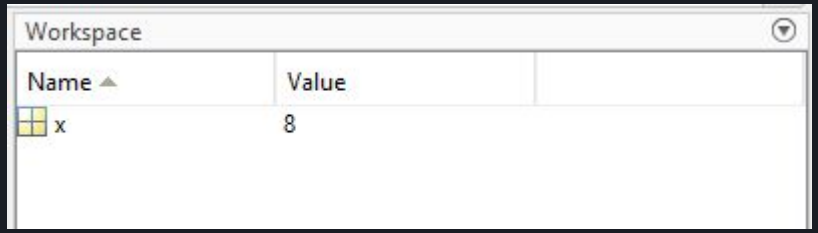
The Simulink Environment Window

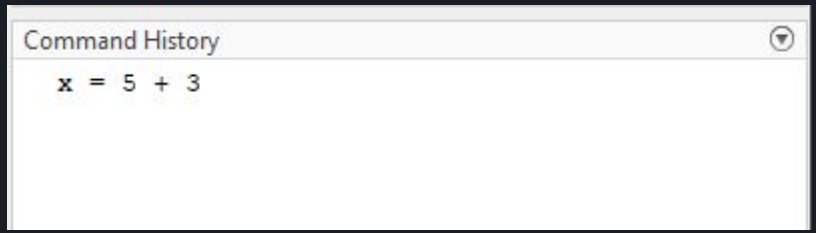


Introduction To MATLAB

- **First Command**
  - x = 5 + 3







Introduction To MATLAB

- **clear**

In MATLAB, the `clear` function is used to remove one or more variables or all variables from the workspace. The workspace is the memory space where MATLAB stores the variables and functions you create during a session

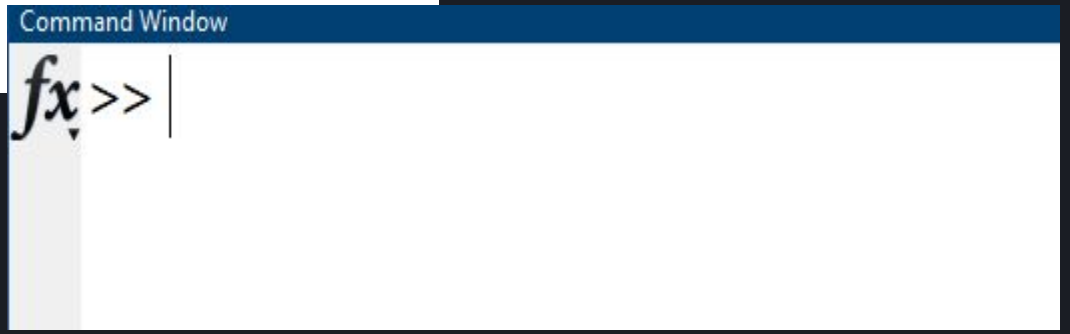
```
1   clear % Clear All Variables
2   clear variableName % Clear a Specific Variable
3   clear variable1 variable2 %Clear Multiple Variables
```

Introduction To MATLAB

- **clc**

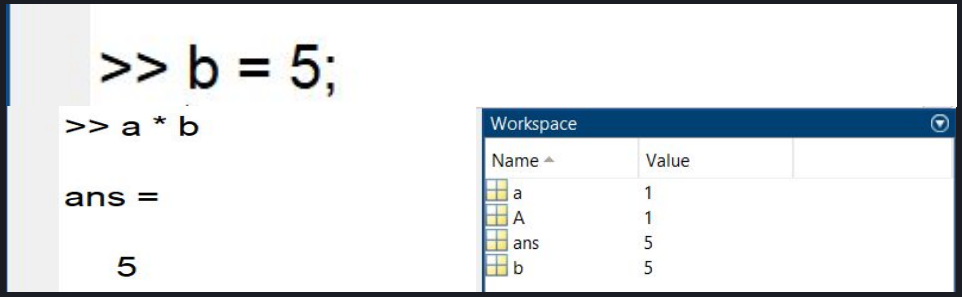
In MATLAB, the `clc` command stands for "clear command window." When you execute `clc`, it clears the command window, which is the area where you enter and execute MATLAB commands.

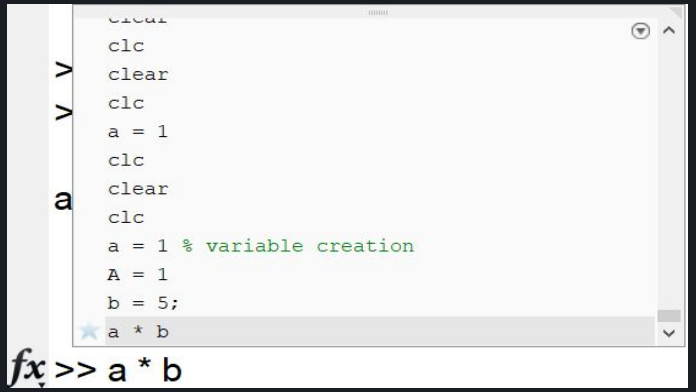
```
>> clear % Clear All Variables
clear variableName % Clear a Specific Variable
clear variable1 variable2 %Clear Multiple Variables

>> clc
```

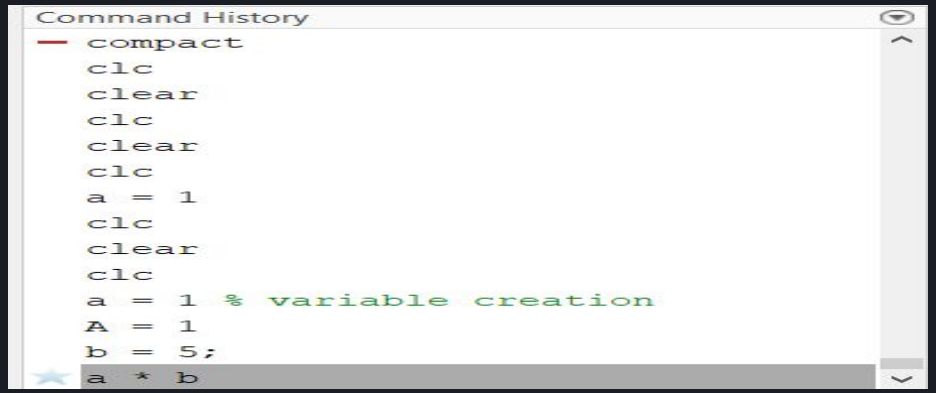


Introduction To MATLAB

- **Semicolon** at the end of a command line suppresses the response.
- Variable "b" appears in the workspace.
- Operations between variables, like multiplication, are straightforward.
- Command history can be accessed using the **up and down arrow** keys.



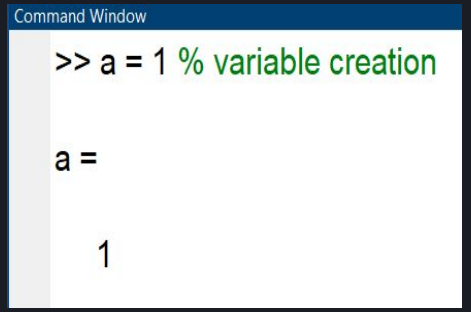




Introduction To MATLAB

- MATLAB distinguishes between capital and lowercase letters.
- "a" and "A" are treated as two separate variables.
- MATLAB creates variable "a" when used.
- Variable "a" appears in the Workspace window.
- The command entered is recorded in the command history window.

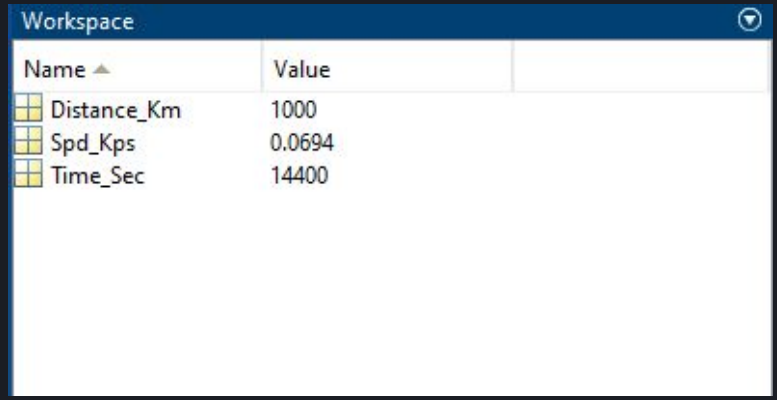

```
>> A = 1

A =

     1
```



Introduction To MATLAB

- Calculate Car Velocity

```
>> Distance_Km = 1e3;
>> Time_Sec = 4 * 60 * 60;
>> Spd_Kps = Distance_Km / Time_Sec;
>> Spd_Kps = Distance_Km / Time_Sec

Spd_Kps =

    0.0694
```



Introduction To MATLAB

- **Variable Naming Rules**
  - Variable names must begin with a letter, followed by letters, digits, or underscores.
  - MATLAB is case-sensitive, so uppercase and lowercase letters are distinct.
  - Avoid using MATLAB reserved words as variable names. For example, words like `if`, `else`, `for`, and `while` have specific meanings in MATLAB and should not be used as variable names.
  - While variable names can include numbers, they cannot start with a number.
  - MATLAB allows the use of underscores (_) However, other special characters are not allowed.
  - While MATLAB allows a variety of characters in variable names, it's good practice to avoid uncommon symbols to ensure code readability and compatibility.

```
myVariable
temperature_1
counter
sum_values
x
y
```

```
1 1stVariable
2 if
3 variable-name
4 temperature#
```

Introduction To MATLAB

- **Variable Naming Rules**

```
1   1stVariable      % Starts with a number
2   if               % Reserved word
3   variable-name    % Contains an invalid character (-)
4   temperature#     % Contains an invalid character (#)
```

- Note: if no variable written, the data will be kept in a variable called ans

```
>> 50 * 10

ans =

   500
```

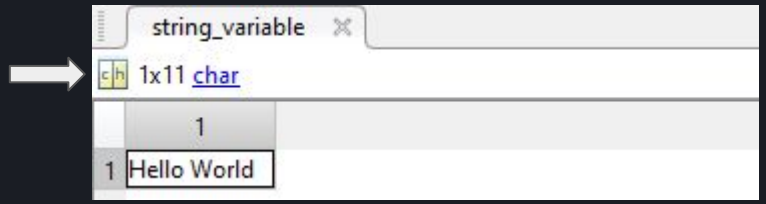
Name	Value
ans	500
Distance_Km	1000
Spd_Kps	0.0694
Time_Sec	14400

Introduction To MATLAB

- **Basic Data in MATLAB**

The fundamental data structure in MATLAB is the array, representing any set of numbers organized in a rectangular pattern. In MATLAB, a one-dimensional array is referred to as a "vector," while a two-dimensional array is called a "matrix." These arrays can encompass various data elements, including numbers, characters, Boolean values (true or false), structures, or cells. MATLAB also extends its support to data structures with more than two dimensions, accepting arrays with N-dimensions.

```
>> string_variable = 'Hello World';
```



Introduction To MATLAB

- **Matrix Creation & Generation Data**

  - Start by using an opening square bracket [to signify the beginning of a matrix. Differentiate elements within a row by using commas or spaces. If you have multiple rows, separate them with a semicolon (;), and conclude the matrix with a closing square bracket]

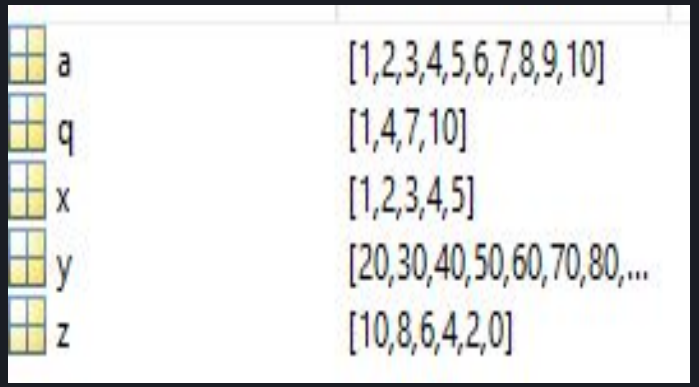
$$A = \begin{bmatrix} 1 & 3 & 7 \\ -5 & 4 & 2 \\ 8 & 9 & 0 \end{bmatrix}$$

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

y =

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

Introduction To MATLAB

- **Matrix Creation & Generation Data**
  - Colon Operator

```
x = 1:5;
y = 20:10:100;
z = 10:-2:0;
a = colon(1,10);
q = colon(1,3,10);
```



Note: Operator is a function that is invoked by a symbol

Introduction To MATLAB

- **Matrix Creation & Generation Data**

A row vector in MATLAB is a one-dimensional array that consists of elements arranged in a single row. Row vectors are useful for representing data when you need to work with a sequence of values or perform operations that require data to be organized in a horizontal fashion. Here's how you can create and work with row vectors in MATLAB:

Using the Colon Operator (`:`): The colon operator allows you to create row vectors with regularly spaced values. You specify a starting value, an increment, and an ending value. For example:

OSAcademy.m

```
% Create a row vector from 1 to 5 with an increment of 1
rowVector = 1:5;

```

Using Horizontal Concatenation: You can create a row vector by horizontally concatenating individual elements or other row vectors using square brackets `[]`. For example:

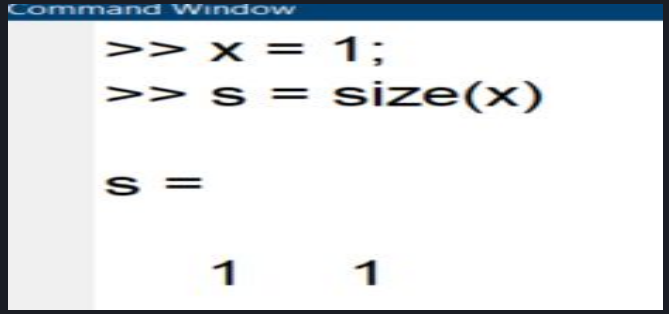
OSAcademy.m

```
% Create a row vector by concatenating individual elements
rowVector = [10 20 30 40 50]; % Separate elements by a space.
rowvector = [10, 20, 30, 40, 50]; % Separate elements by a colon.
```

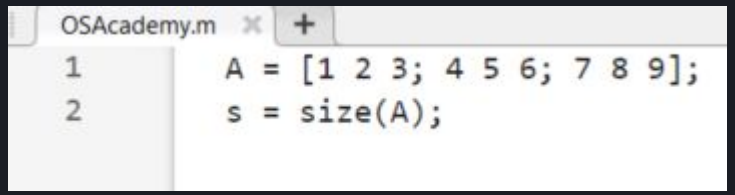
Introduction To MATLAB

- **Matrix Creation & Generation Data**

  - size
    The size function in MATLAB is used to determine the dimensions (size) of an array or matrix. It returns a row vector containing the number of rows and columns in the input array. The general syntax is:



In this **example**, `s` will be `[3 3]` because the matrix `A` has 3 rows and 3 columns.



Where:

- `x` is the input array or matrix.
- `s` is the output row vector containing the size information.

Introduction To MATLAB

- **Matrix Creation & Generation Data**

In this **example**, `s` will be `[3 3]` because the matrix `A` has 3 rows and 3 columns.

The **`size`** function can be used with N-dimensional arrays as well. It returns the size of each dimension as elements in the output vector.

Here, `s` will be `[2 3 4]`, indicating that the array `C` has dimensions of 2x3x4.

In this **case**, `s` will be `[1 5]` because the vector `B` is a 1x5 matrix (1 row and 5 columns).

OSAcademy.m

```
1   A = [1 2 3; 4 5 6; 7 8 9];
2   s = size(A);
```

OSAcademy.m

```
1   C = rand(2, 3, 4);  % A 3-dimensional array
2   s = size(C);
```

```
>> OSAcademy
s =
     2     3     4
```

OSAcademy.m

```
1   B = [10 20 30 40 50];
2   s = size(B);
```

Introduction To MATLAB

- **Matrix Creation & Generation Data**

  You can also specify a particular dimension using an additional argument in the `size` function. For example:

OSAcademy.m

```
1  A = [1 2 3; 4 5 6; 7 8 9];
2  numRows = size(A, 1);  % Get the number of rows (1st dimension)
3  numCols = size(A, 2);  % Get the number of columns (2nd dimension)
4
```

In this case, `numRows` will be `3`, and `numCols` will be `3`.

Command Window

```
>> OSAcademy
numRows =
     3
numCols =
     3
```

Introduction To MATLAB

- **Matrix Creation & Generation Data**

You can use `size` to check if an array is empty. An empty array has dimensions `[0 0]`.

OSAcademy.m

```
1  D = [];  % An empty array
2  isEmpty = (size(D) == [0 0]);  % Check if it's empty
3
```

`isEmpty` will be `true` because the size of the empty array `D` is `[0 0]`.

The `size` function is valuable for dynamically determining the dimensions of arrays, especially when working with data of varying sizes or dimensions in MATLAB. It allows you to adapt your code to the size of the input data, making it more versatile and robust.

Introduction To MATLAB

- **Matrix Creation & Generation Data**

In MATLAB, a matrix is a fundamental data structure that represents a two-dimensional, rectangular array of elements. Each element in a matrix can hold numerical values, and the matrix's dimensions are defined by its number of rows and columns. Matrices play a crucial role in various mathematical and computational tasks, making them a fundamental concept in MATLAB.

You can create a matrix in MATLAB using square brackets `[]` to enclose its elements. Elements within each row are separated by spaces or commas, and semicolons `;` or newline characters separate rows.

OSAcademy.m

```
% Creating a 2x3 matrix
matrix = [1 2 3; 4 5 6];
```

In this example, we've created a 2x3 matrix called `matrix`.

You can determine the dimensions of a matrix using the `size` function, which returns the number of rows and columns in the matrix.

OSAcademy.m

```
% Get the dimensions of the matrix
[rowCount, colCount] = size(matrix);
disp(['Rows: ' num2str(rowCount) ', Columns: ' num2str(colCount)]);
```

Introduction To MATLAB

TRY THIS!

- **Matrix Creation & Generation Data**

Exercise 1: Finding the Size of a Matrix:

OSAcademy.m *

```
% 1.Create a matrix A with the following elements:
%
% 1  2  3  4
% 5  6  7  8
% 9 10 11 12
% 2.Use the size function to find the dimensions of the matrix A. Store the result in a variable A_size.
%
% 3.Display the value of A_size.
```

Introduction To MATLAB

- **Matrix Creation & Generation Data**

Exercise 2: Size of Vectors

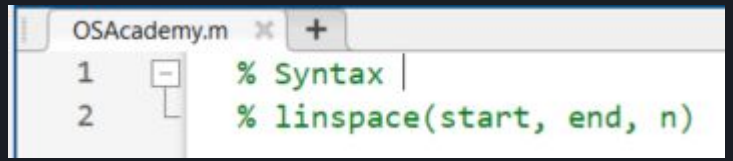
OSAcademy.m *

```
1  % Create a row vector B with the numbers from 1 to 10 using the colon operator (:).
2  %
3  % Use the size function to determine the size of vector B. Store the result in a variable B_size.
4  %
5  % Display the value of B_size.
```

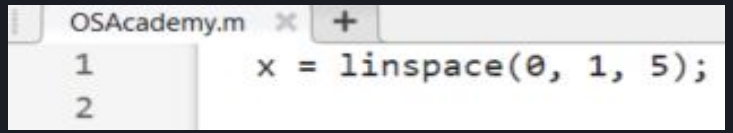
Introduction To MATLAB

- **Matrix Creation & Generation Data**

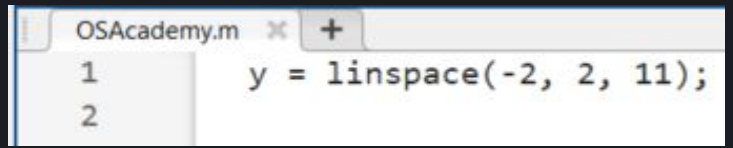
The `linspace` function in MATLAB is used to create an evenly spaced vector of numbers between two specified values. It is particularly useful when you need a set of values at regular intervals within a specified range. The syntax for `linspace` is as follows:



- `start`: The starting value of the sequence.
- `end`: The ending value of the sequence.
- `n`: The number of equally spaced points you want to generate.



In this example, `linspace` generates a vector `x` containing 5 equally spaced values between 0 and 1. The resulting `x` vector would be `[0, 0.25, 0.5, 0.75, 1]`.



In this case, `linspace` creates a vector `y` with 11 equally spaced values between -2 and 2. The resulting `y` vector would be `[-2, -1.6, -1.2, -0.8, -0.4, 0, 0.4, 0.8, 1.2, 1.6, 2]`.

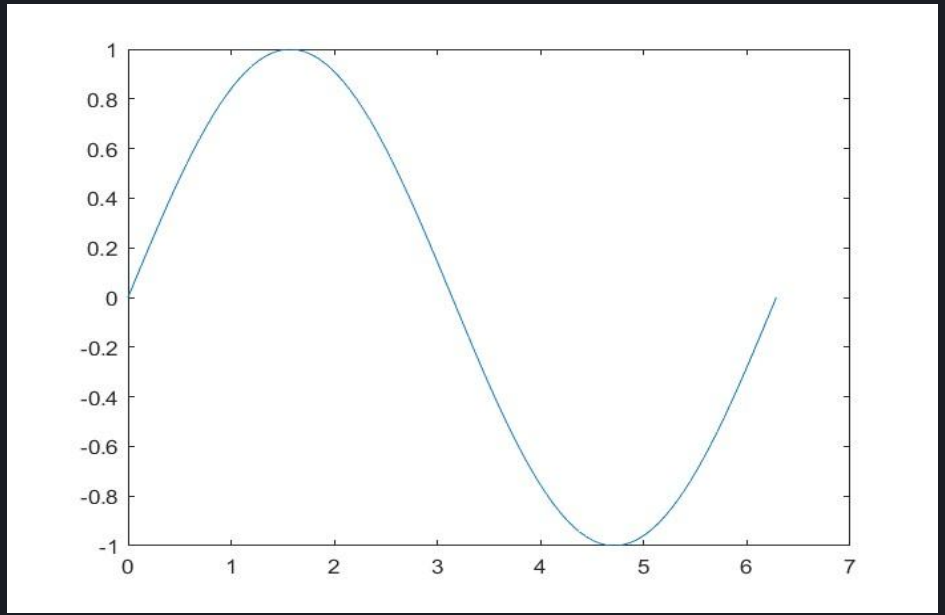
Introduction To MATLAB

- **Matrix Creation & Generation Data**

The `linspace` function in MATLAB is used to create an evenly spaced vector of numbers between two specified values. It is particularly useful when you need a set of values at regular intervals within a specified range. The syntax for `linspace` is as follows:

OSAcademy.m

```
1   x = linspace(0, 2*pi, 100);
2   y = sin(x);
3
4   plot(x, y);
5
```



In this example, `linspace` is used to generate 100 equally spaced points between 0 and `2*pi` (a full circle), and then the `sin` function is applied to calculate the corresponding `y` values. This is commonly used to create smooth curves for plotting.

Introduction To MATLAB

- **Matrix Creation & Generation Data**

The `logspace` function in MATLAB is used to create a vector of values that are logarithmically spaced between two specified exponents. It is particularly useful when you need values that cover a wide range in a logarithmic scale. The syntax for `logspace` is as follows:

OSAcademy.m *

```
% Syntax
% logspace(startExponent, endExponent, n)
```

- `startExponent`: The starting exponent of the sequence.
- `endExponent`: The ending exponent of the sequence.
- `n`: The number of values you want to generate.

OSAcademy.m

```
x = logspace(0, 2, 5);
```

In this example, `logspace` generates a vector `x` containing 5 values that are logarithmically spaced between 10^0 and 10^2. The resulting `x` vector would be `[1, 10, 100, 1000, 10000]`.

OSAcademy.m

```
y = logspace(-2, 2, 11);
```

Here, `logspace` creates a vector `y` with 11 values that are logarithmically spaced between 10^-2 and 10^2. The resulting `y` vector would be `[0.01, 0.0258, 0.066, 0.171, 0.439, 1.13, 2.92, 7.54, 19.5, 50.1, 129]`.

Introduction To MATLAB

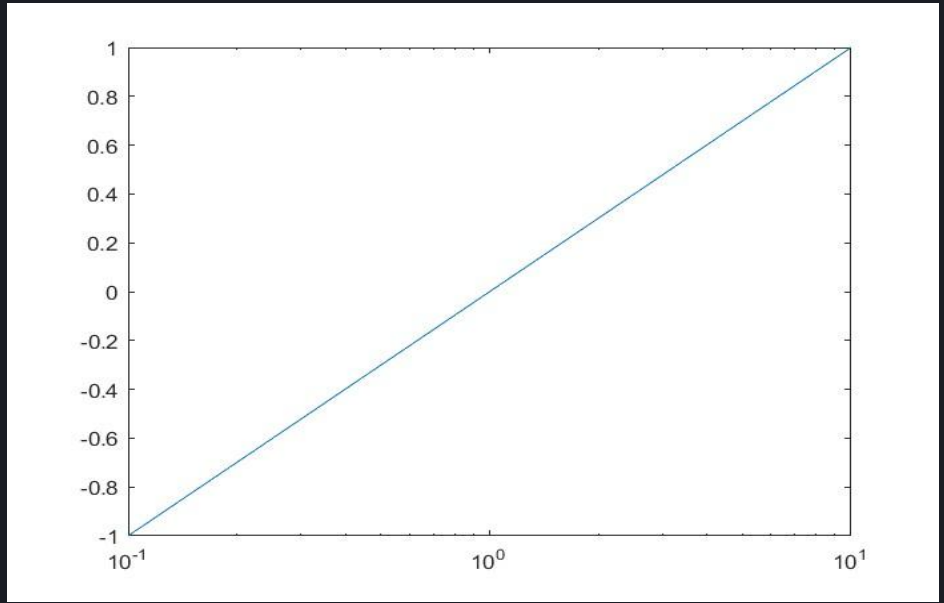
- **Matrix Creation & Generation Data**

The `logspace` function in MATLAB is used to create a vector of values that are logarithmically spaced between two specified exponents. It is particularly useful when you need values that cover a wide range in a logarithmic scale. The syntax for `logspace` is as follows:

OSAcademy.m

```
1  x = logspace(-1, 1, 100);
2  y = log10(x);
3
4  semilogx(x, y);
5
```

In this example, `logspace` is used to generate 100 values that are logarithmically spaced between 10^-1 and 10^1. Then, the `log10` function is applied to calculate the logarithm base 10 of these values. The `semilogx` function is used for plotting, which displays the x-axis in logarithmic scale.



Introduction To MATLAB

- **Matrix Creation & Generation Data**

In MATLAB, the `zeros` function is used to create matrices or arrays filled with zeros. It allows you to specify the dimensions of the resulting matrix, and it generates a matrix of the specified size with all elements set to 0. The `zeros` function is commonly used when you need to initialize a matrix with zeros for various mathematical or computational tasks.

You can create a matrix of zeros by specifying the number of rows and columns as arguments to the `zeros` function. For example, to create a 2x3 matrix of zeros:

OSAcademy.m

```
1  % Creating a 2x3 matrix of zeros
2  matrix = zeros(2, 3);
3
```

The `zeros` function can also create one-dimensional arrays filled with zeros. For example, to create a row vector with four zeros:

OSAcademy.m

```
1  % Creating a row vector with four zeros
2  rowVector = zeros(1, 4);
```

Introduction To MATLAB

- **Matrix Creation & Generation Data**

In MATLAB, the `zeros` function is used to create matrices or arrays filled with zeros. It allows you to specify the dimensions of the resulting matrix, and it generates a matrix of the specified size with all elements set to 0. The `zeros` function is commonly used when you need to initialize a matrix with zeros for various mathematical or computational tasks.

You can specify the data type of the zeros using additional arguments. For instance, if you want to create a matrix of single-precision zeros:

OSAcademy.m

```
% Creating a 3x3 matrix of single-precision zeros
matrix = zeros(3, 3, 'single');
```

Similar to the `ones` function, you can combine the `zeros` function with scalar multiplication to generate matrices or arrays with values other than 0. For example, to create a 4x4 matrix with all elements set to 2:

OSAcademy.m

```
% Creating a 4x4 matrix with all elements as 2
matrix = 2 * zeros(4, 4);

```

The `zeros` function is valuable when you need to initialize arrays or matrices with specific values (usually zeros) for further calculations or data manipulation in MATLAB.

Introduction To MATLAB

- **Matrix Creation & Generation Data**

In MATLAB, the `ones` function is used to create matrices or arrays filled with ones. You can specify the dimensions of the resulting matrix, and it will generate a matrix of the specified size with all elements set to 1. The `ones` function is handy when you need to initialize a matrix with ones for various mathematical or computational tasks.

You can create a matrix of ones by specifying the number of rows and columns as arguments to the `ones` function. For example, to create a 3x2 matrix of ones:

OSAcademy.m

```
1  % Creating a 3x2 matrix of ones
2  matrix = ones(3, 2);
3
```

The `ones` function can also create one-dimensional arrays filled with ones. For example, to create a row vector with five ones:

OSAcademy.m

```
1  % Creating a row vector with five ones
2  rowVector = ones(1, 5);
3
```

Introduction To MATLAB

- **Matrix Creation & Generation Data**

In MATLAB, the `ones` function is used to create matrices or arrays filled with ones. You can specify the dimensions of the resulting matrix, and it will generate a matrix of the specified size with all elements set to 1. The `ones` function is handy when you need to initialize a matrix with ones for various mathematical or computational tasks.

You can specify the data type of the ones using additional arguments. For instance, if you want to create a matrix of double-precision ones:

OSAcademy.m

```
1  % Creating a 2x3 matrix of double-precision ones
2  matrix = ones(2, 3, 'double');
3
```

You can also combine the `ones` function with scalar multiplication to generate matrices or arrays with values other than 1. For example, to create a 4x4 matrix with all elements set to 5:

OSAcademy.m

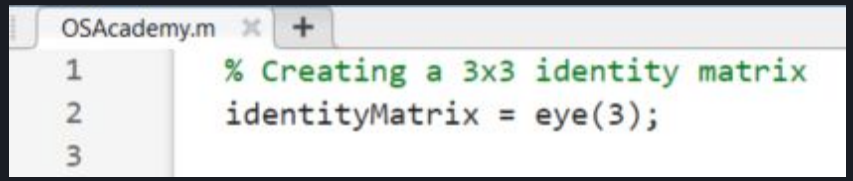
```
1  % Creating a 4x4 matrix with all elements as 5
2  matrix = 5 * ones(4, 4);
3
```

Introduction To MATLAB

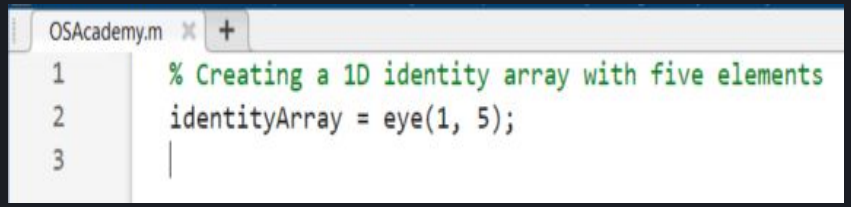
- **Matrix Creation & Generation Data**

In MATLAB, the `eye` function is used to create identity matrices or arrays. An identity matrix is a square matrix in which all the elements of the main diagonal are set to 1, and all other elements are set to 0. Identity matrices are commonly used in linear algebra and various mathematical operations.

You can create an identity matrix by specifying the number of rows and columns as an argument to the `eye` function. For example, to create a 3x3 identity matrix:



The `eye` function can also create identity arrays. For instance, to create a 1D array with five elements:



Introduction To MATLAB

- **Matrix Creation & Generation Data**

In MATLAB, the `eye` function is used to create identity matrices or arrays. An identity matrix is a square matrix in which all the elements of the main diagonal are set to 1, and all other elements are set to 0. Identity matrices are commonly used in linear algebra and various mathematical operations.

similar to other MATLAB functions, you can specify the data type of the identity matrix or array using additional arguments. For example, to create a single-precision identity matrix:

OSAcademy.m

```
% Creating a 4x4 single-precision identity matrix
singleIdentityMatrix = eye(4, 'single');
```

You can combine the `eye` function with scalar multiplication to generate identity matrices or arrays with values other than 1. For example, to create a 2x2 matrix with all elements set to 2:

OSAcademy.m

```
% Creating a 2x2 matrix with all elements as 2
customIdentityMatrix = 2 * eye(2);
```

Introduction To MATLAB

- **Matrix Creation & Generation Data**

In MATLAB, the `rand` function is used to generate random numbers from a uniform distribution within the range [0, 1]. This function is commonly used when you need to create arrays or matrices with random values for various applications, such as simulations, statistical analysis, and testing.

You can use the `rand` function to generate random scalar values. For example, to generate a single random value between 0 and 1:

```
% Generating a random scalar value between 0 and 1
randomValue = rand;
```

`randomValue` will contain a random decimal value between 0 (inclusive) and 1 (exclusive).

The `rand` function is often used to create random arrays or matrices of specified dimensions. For example, to create a 2x3 matrix of random values:

```
% Creating a 2x3 matrix of random values
randomMatrix = rand(2, 3);
```

Introduction To MATLAB

- **Matrix Creation & Generation Data**

In MATLAB, the `rand` function is used to generate random numbers from a uniform distribution within the range [0, 1]. This function is commonly used when you need to create arrays or matrices with random values for various applications, such as simulations, statistical analysis, and testing.

You can scale and shift the random values generated by the `rand` function to fit a different range or distribution. For example, to generate random values between 5 and 10:

OSAcademy.m

```
% Generating random values between 5 and 10
scaledRandomValues = 5 + rand(1, 5) * 5;

```

The `rand` function is often used to create random arrays or matrices of specified dimensions. For example, to create a 2x3 matrix of random values:

OSAcademy.m

```
% Creating a 2x3 matrix of random values
randomMatrix = rand(2, 3);
```

Introduction To MATLAB

Quiz 1: Click Here To Start

Quiz 2: Click Here To Start

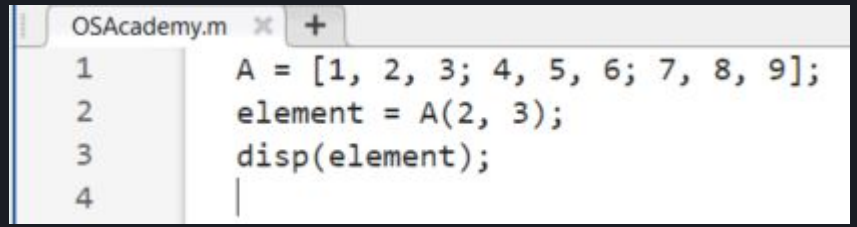
Introduction To MATLAB

- **Matrix Indexing**

In MATLAB, you can access elements in a matrix by specifying the row and column indices. Here are examples of how to access a matrix element and some MATLAB code to illustrate it:

Accessing a Single Element:

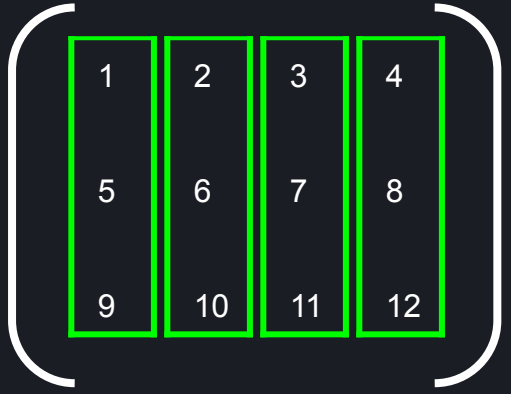
- To access a specific element in a matrix, you use the notation `matrix_name(row_index, column_index)`.
- Replace `matrix_name` with the name of your matrix.
- Replace `row_index` with the index of the row where the element is located.
- Replace `column_index` with the index of the column where the element is located.



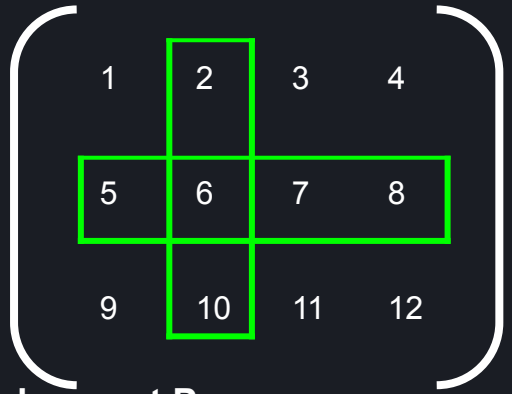
Introduction To MATLAB

- **Matrix Indexing**

```
>> X =[1:4;5:8;9:12];
```



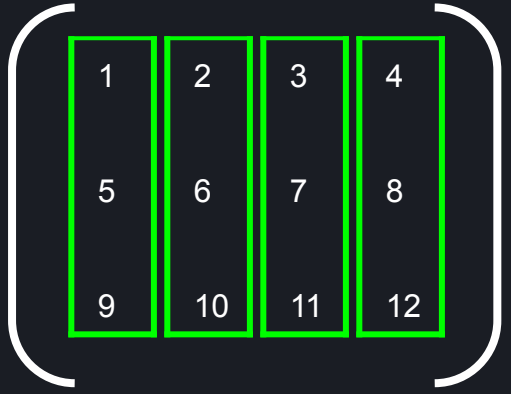

```
>> Element = X(2,2);
```



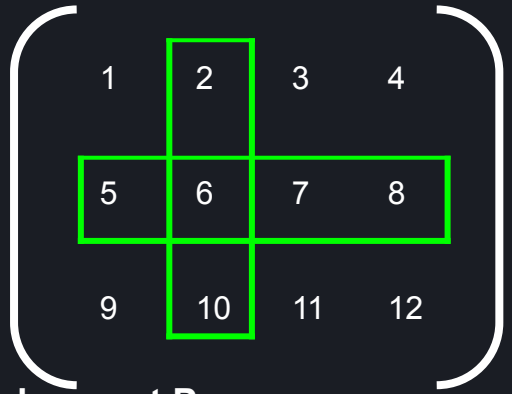
Introduction To MATLAB

- **Matrix Indexing**

```
>> X =[1:4;5:8;9:12];
```




```
>> Element = X(2,2);
```

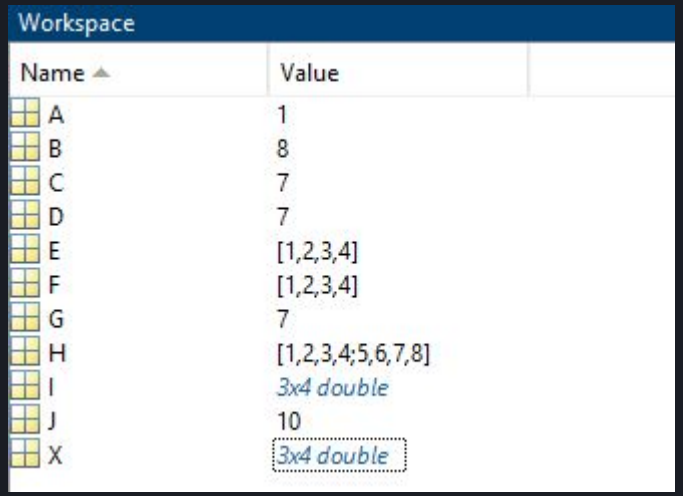


Introduction To MATLAB

- **Matrix Indexing**

```
>> X =[1:4;5:8;9:12];
>> A = X(1,1);
>> B = X(2,4);
>> C = X(8);
>> E = X(1,[1,2,3,4]);
>> F = X(1,:);
>> G = X(2,3);
>> H = X([1,2],:);
>> I = X(:,:);
>> J = X(end,2);
```

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



Introduction To MATLAB

- **Mathematical Operation**
  - Array Operations
    a. Array operations in MATLAB work on elements that correspond in arrays of equal dimensions.
    b. Each element in the first input array is matched with the corresponding element in the second input array.
    c. This mechanism is suitable for vectors, matrices, and multidimensional arrays.
    d. If the input arrays are of different sizes, MATLAB cannot perform one-to-one matching of elements.

  - Matrix Operation
    a. Matrix operations adhere to the principles of linear algebra and do not support compatibility with multidimensional arrays.
    b. The necessary size and arrangement of the inputs, concerning each other, depend on the specific operation.

Introduction To MATLAB

- **Mathematical Operation**
  - Array Operations (Addition)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} + \begin{pmatrix} 2 & 3 \\ 4 & 5 \\ 7 & 6 \end{pmatrix} = \begin{pmatrix} 1+2 & 2+3 \\ 3+4 & 4+5 \\ 5+7 & 6+6 \end{pmatrix} = \begin{pmatrix} 3 & 5 \\ 7 & 9 \\ 12 & 12 \end{pmatrix}$$

```
>> x = [1 2; 3 4; 5 6];
>> y = [2 3;4 5; 7 6];
>> z = plus(x,y);
```

Introduction To MATLAB

- **Mathematical Operation**
  - Array Operations (Subtraction)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} - \begin{pmatrix} 2 & 3 \\ 4 & 5 \\ 7 & 6 \end{pmatrix} = \begin{pmatrix} 1-2 & 2-3 \\ 3-4 & 4-5 \\ 5-7 & 6-6 \end{pmatrix} = \begin{pmatrix} -1 & -1 \\ -1 & -1 \\ -2 & 0 \end{pmatrix}$$

Command Window

```
>> x = [1 2; 3 4; 5 6];
>> y = [2 3;4 5; 7 6];
>> z = minus(x,y);
```

Introduction To MATLAB

- **Mathematical Operation**
  - Array Operations (Multiplication)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} .* \begin{pmatrix} 2 & 3 \\ 4 & 5 \\ 7 & 6 \end{pmatrix} = \begin{pmatrix} 1*2 & 2*3 \\ 3*4 & 4*5 \\ 5*7 & 6*6 \end{pmatrix} = \begin{pmatrix} 2 & 6 \\ 12 & 20 \\ 35 & 36 \end{pmatrix}$$

```
>> x = [1 2; 3 4; 5 6];
>> y = [2 3;4 5; 7 6];
>> z = times(x,y);
>> z = x.*y;
```

Introduction To MATLAB

- **Mathematical Operation**
  - Array Operations (Multiplication)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} .^{\wedge} \begin{pmatrix} 2 & 3 \\ 4 & 5 \\ 7 & 6 \end{pmatrix} = \begin{pmatrix} 1^2 & 2^3 \\ 3^4 & 4^5 \\ 5^7 & 6^6 \end{pmatrix} = \begin{pmatrix} 1 & 8 \\ 81 & 1024 \\ 7815 & 46656 \end{pmatrix}$$

```
>> x = [1 2; 3 4; 5 6];
>> y = [2 3;4 5; 7 6];
>> z = x.^y;
>> z = power(x,y);
```

Introduction To MATLAB

- **Mathematical Operation**
  - Array Operations (Division)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} ./ \begin{pmatrix} 2 & 3 \\ 4 & 5 \\ 7 & 6 \end{pmatrix} = \begin{pmatrix} 1/2 & 2/3 \\ 3/4 & 4/5 \\ 5/7 & 6/6 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.6667 \\ 0.7500 & 0.8000 \\ 0.7143 & 1 \end{pmatrix}$$

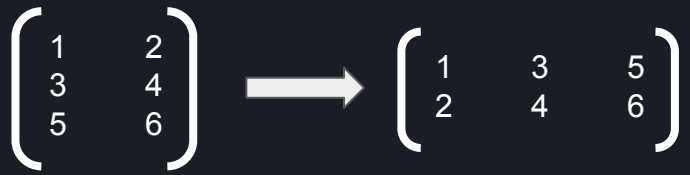
$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} .\backslash \begin{pmatrix} 2 & 3 \\ 4 & 5 \\ 7 & 6 \end{pmatrix} = \begin{pmatrix} 1/2 & 2/3 \\ 3/4 & 4/5 \\ 5/7 & 6/6 \end{pmatrix} = \begin{pmatrix} 2 & 1.5 \\ 1.3333 & 1.2500 \\ 1.4000 & 1 \end{pmatrix}$$

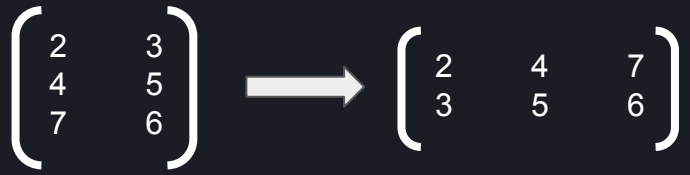
```
>> x = [1 2; 3 4; 5 6];
>> y = [2 3;4 5; 7 6];
>> z_rd = x./y;
>> z_rd = rdivide(x,y);
```

```
>> x = [1 2; 3 4; 5 6];
>> y = [2 3;4 5; 7 6];
>> z_ld = x.\y;
>> z_ld = ldivide(x,y);
```

Introduction To MATLAB

- **Mathematical Operation**
  - Array Operations (Transpose)



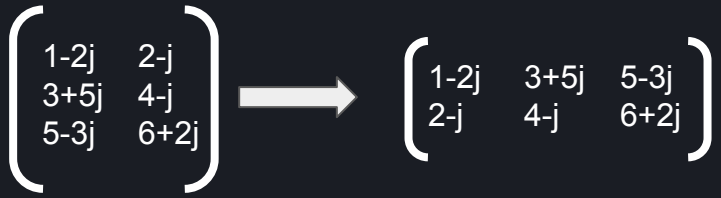


```
>> x = [1 2; 3 4; 5 6];
>> z = x.';
```

```
>> y = [2 3;4 5; 7 6];
>> z = y.';
```

Introduction To MATLAB

- **Mathematical Operation**
  - Array Operations (Transpose)




```
>> z = [1-2*j 2-j 3+5*j; 4-j 5-3*j 6+2*j];
>> z.'

ans =

   1.0000 - 2.0000i   4.0000 - 1.0000i
   2.0000 - 1.0000i   5.0000 - 3.0000i
   3.0000 + 5.0000i   6.0000 + 2.0000i
```

Introduction To MATLAB

- **Mathematical Operation**
  - Array Operations (Unary plus & minus)

$$x = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

$$z = +x = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

$$z = -x = \begin{pmatrix} -1 & -2 \\ -3 & -4 \\ -5 & -6 \end{pmatrix}$$

```
>> x = [1 2; 3 4; 5 6];
>> z = +x;
```

```
>> x = [1 2; 3 4; 5 6];
>> z = uplus(x);
>> z = uminus(x);
```

```
>> x = [1 2; 3 4; 5 6];
>> z = -x;
```

Introduction To MATLAB

- **Concatenation**

Matrix concatenation in MATLAB refers to the process of combining two or more matrices or arrays to create a larger matrix. This operation can be performed either horizontally (side by side) or vertically (top and bottom). MATLAB provides several ways to concatenate matrices, allowing you to manipulate and organize data effectively.

Horizontal concatenation combines two or more matrices or arrays by placing them next to each other in the same row. You can use square brackets `[]` to perform horizontal concatenation. For example:

OSAcademy.m

```
% Horizontal concatenation of two matrices
matrix1 = [1 2; 3 4];
matrix2 = [5 6; 7 8];
result = [matrix1, matrix2];

```

Vertical concatenation combines matrices by stacking them on top of each other in the same column. You can use semicolons `;` or the `vertcat` function for vertical concatenation. For example:

OSAcademy.m

```
% Vertical concatenation of two matrices
matrix1 = [1 2; 3 4];
matrix2 = [5 6; 7 8];
result = [matrix1; matrix2];

```

Introduction To MATLAB

- **Concatenation**

The `cat` function in MATLAB is used for concatenating arrays along a specified dimension. It allows you to combine arrays either horizontally (side by side) or vertically (top and bottom) based on the dimension you specify. Here's how the `cat` function works with examples:

- `dim`: Specifies the dimension along which concatenation will occur (1 for vertical concatenation, 2 for horizontal concatenation).
- `A`, `B`, ...: The arrays you want to concatenate.

OSAcademy.m *

```
% syntax
% C = cat(dim, A, B, ...)
```

Concatenate two arrays vertically to stack them on top of each other.

OSAcademy.m

```
% Vertical concatenation along dimension 1
A = [1 2; 3 4];
B = [5 6; 7 8];
C = cat(1, A, B);
```

Concatenate two arrays vertically to stack them on top of each other.

OSAcademy.m

```
% Horizontal concatenation along dimension 2
A = [1 2; 3 4];
B = [5 6; 7 8];
C = cat(2, A, B);
```

Introduction To MATLAB

- **Concatenation**

The `cat` function in MATLAB is used for concatenating arrays along a specified dimension. It allows you to combine arrays either horizontally (side by side) or vertically (top and bottom) based on the dimension you specify. Here's how the `cat` function works with examples:

You can concatenate more than two arrays at once.

OSAcademy.m

```
% Concatenating three arrays vertically
A = [1 2; 3 4];
B = [5 6; 7 8];
D = [9 10; 11 12];
E = cat(1, A, B, D);
```

The `cat` function is particularly useful when you need to concatenate arrays along a specific dimension, and it provides flexibility in managing data in various applications, such as data analysis, signal processing, and image manipulation.

Introduction To MATLAB

- **Concatenation**

The `horzcat` function in MATLAB is short for "horizontal concatenation." It is used to concatenate arrays horizontally, meaning it combines arrays side by side along the second dimension (columns). Here's how the `horzcat` function works with examples:

Combine two arrays horizontally by placing them side by side.

OSAcademy.m

```
1  % Horizontal concatenation
2  A = [1 2; 3 4];
3  B = [5 6; 7 8];
4  C = horzcat(A, B);
5
```

You can concatenate more than two arrays at once.

OSAcademy.m

```
1  % Horizontal concatenation of three arrays
2  A = [1 2; 3 4];
3  B = [5 6; 7 8];
4  D = [9 10; 11 12];
5  E = horzcat(A, B, D);
6
```

The `horzcat` function is helpful when you need to combine data from multiple arrays or matrices side by side to create a larger matrix. It simplifies the process of organizing and working with data in various applications, including numerical computations, data analysis, and more.

Introduction To MATLAB

- **Concatenation**

The `vertcat` function in MATLAB is short for "vertical concatenation." It is used to concatenate arrays vertically, meaning it combines arrays on top of each other along the first dimension (rows). Here's how the `vertcat` function works with examples:

Combine two arrays vertically by stacking them on top of each other.

OSAcademy.m

```
% Vertical concatenation
A = [1 2; 3 4];
B = [5 6; 7 8];
C = vertcat(A, B);
```

You can concatenate more than two arrays at once.

OSAcademy.m

```
% Vertical concatenation of three arrays
A = [1 2; 3 4];
B = [5 6; 7 8];
D = [9 10; 11 12];
E = vertcat(A, B, D);
```

The `vertcat` function is helpful when you need to combine data from multiple arrays or matrices on top of each other to create a larger matrix. It simplifies the process of organizing and working with data in various applications, including numerical computations, data analysis, and more.

Introduction To MATLAB

- **Repating**

The `repmat` function in MATLAB is short for "repeat matrix." It is used to create an array by repeating a given matrix or array in both row and column dimensions. Here's how the `repmat` function works with examples:

OSAcademy.m *

```
% Syntax
% B = repmat(A, m, n)
```

- `A`: The matrix or array you want to repeat.
- `m`: The number of times you want to repeat `A` in the row dimension (vertical).
- `n`: The number of times you want to repeat `A` in the column dimension (horizontal).

Create a larger matrix by repeating a smaller one.

OSAcademy.m

```
% Repeat a 2x2 matrix to create a 4x4 matrix
A = [1 2; 3 4];
B = repmat(A, 2, 2);
```

You can also repeat arrays.

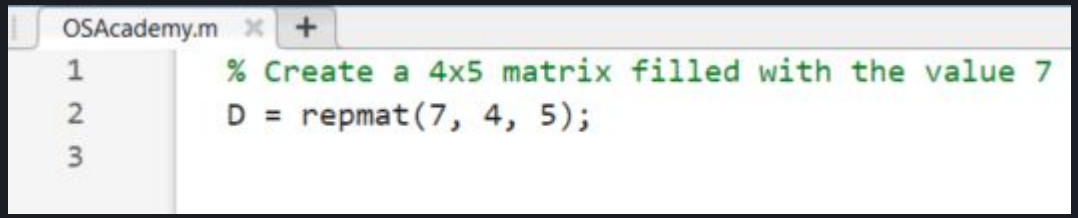
OSAcademy.m

```
% Repeat a 1x3 array to create a 3x6 array
A = [10 20 30];
C = repmat(A, 3, 2);
```

Introduction To MATLAB

- **Concatenation**

You can also use `repmat` with scalar values, and it will create a matrix of the specified size filled with that scalar value.

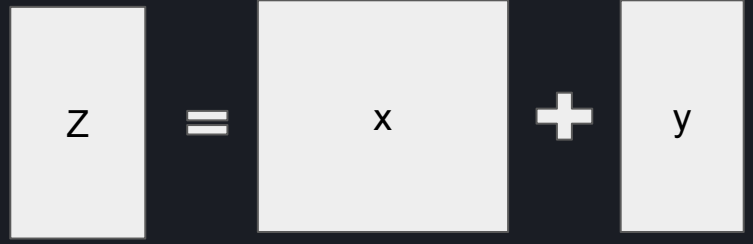


The `repmat` function is useful when you need to create larger arrays or matrices by repeating existing ones in both row and column dimensions. This can be handy for various applications, including creating grids, initializing matrices for calculations, and more.

Introduction To MATLAB

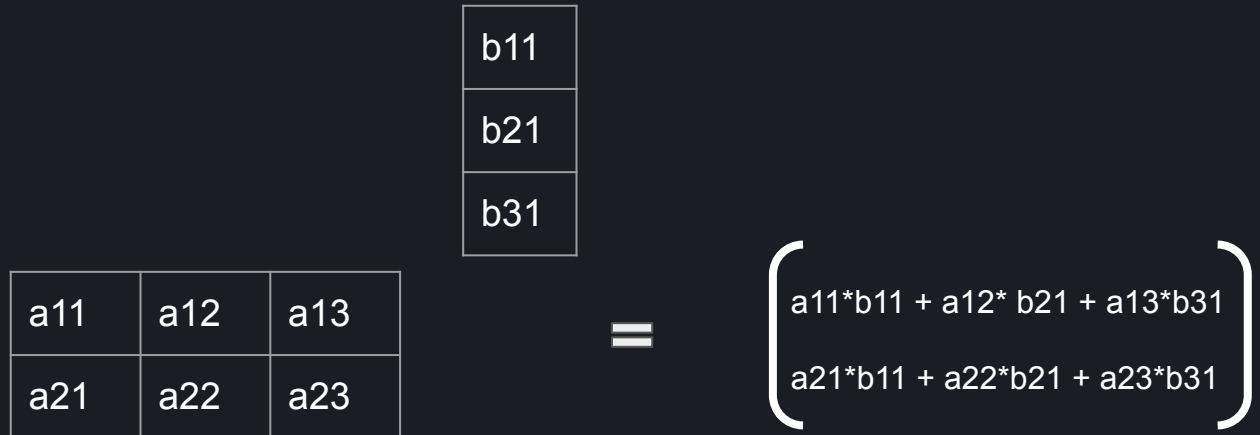
- **Mathematical Operation**
  - Matrix Operations (Multiplication)

$$x = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \qquad y = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$$



Introduction To MATLAB

- **Mathematical Operation**
  - Matrix Operations (Multiplication)



Introduction To MATLAB

- **Mathematical Operation**
  - Matrix Operations (Multiplication)

$$x = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \qquad y = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$Z = x * y = \begin{pmatrix} 1*1 + 2*2 + 3*3 \\ 4*1 + 5*2 + 6*3 \end{pmatrix} = \begin{pmatrix} 14 \\ 32 \end{pmatrix}$$

```
>> A = [1 2 3; 4 5 6];
>> B = [1; 2; 3];
>> C = A * B

C =

   14
   32

>> C = mtimes(A,B)

C =

   14
   32
```

Introduction To MATLAB

Quiz 1: Click Here To Start

Quiz 2: Click Here To Start

Introduction To MATLAB

Lab 1: Click Here To Start

Quiz 2: Click Here To Start


Thank You!