Interactive Session

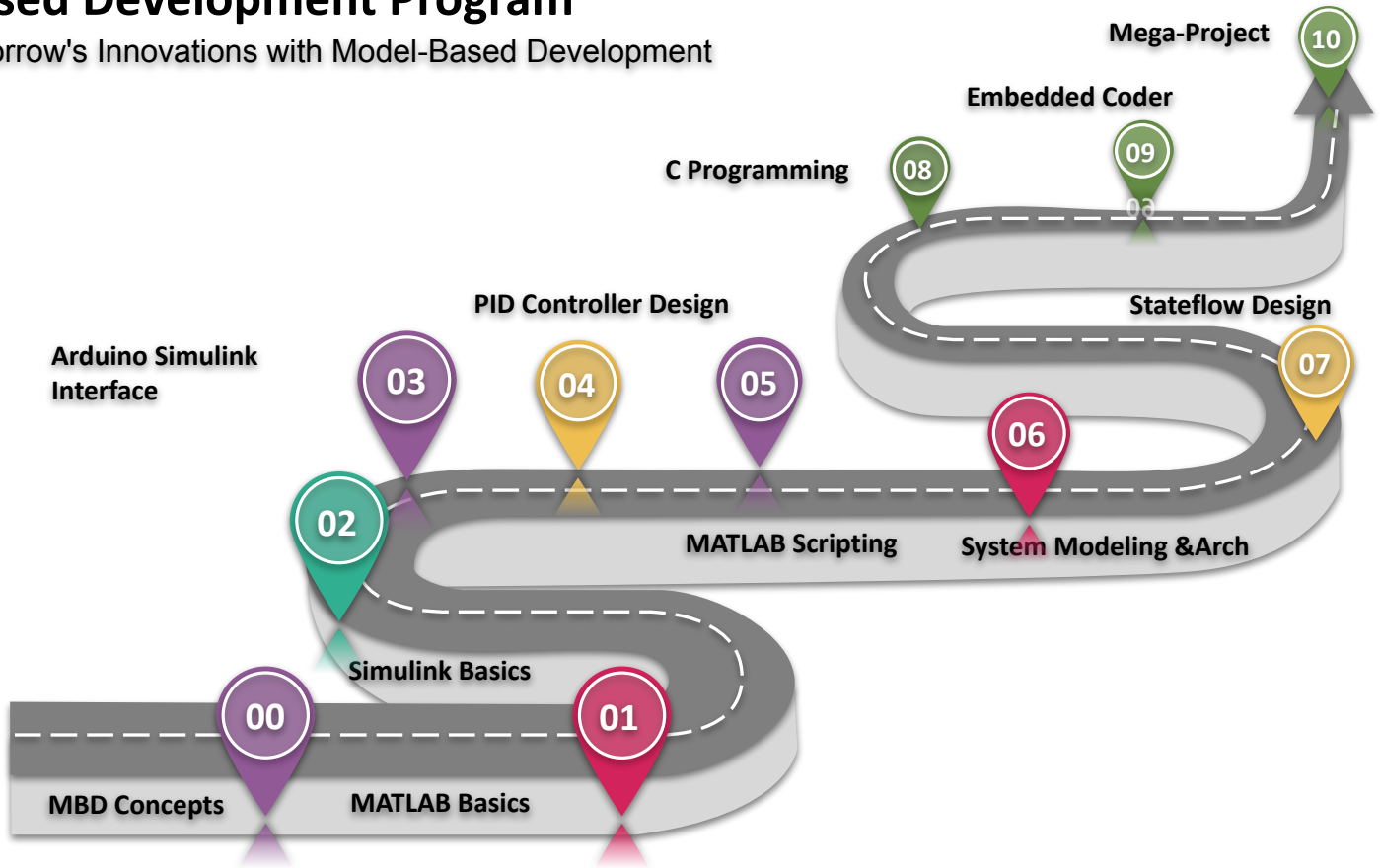
Interactive Sessions: Your Path to Success

Objectives

- Our goal with these *Interactive Sessions* is to provide you with a clear study plan to follow throughout the week. By the end of each week, we will hold an interactive session that will include the following:
  - **Sessions Summaries:** We'll summarize a set number of sessions to give you a clear overview of the key points.
  - **Tips and Tricks:** Share quick, effective tips to help you enhance your learning and improve your MBD skills.
  - **Interview Questions:** Review common interview questions to help you prepare for potential job opportunities in the field.
  - **Challenges:** Weekly challenges will be assigned, and you'll be required to solve them and upload your solutions to the diploma's Facebook group. We'll reserve the WhatsApp group for discussions and any questions you may have.

Model-Based Development Program

Unlocking Tomorrow's Innovations with Model-Based Development



Interactive Sessions: Your Path to Success

MBD Concepts Module:

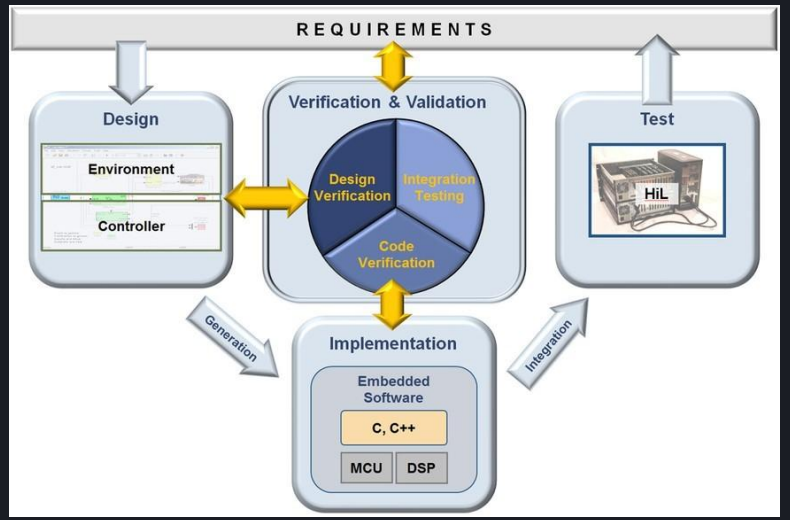
- ❑ What is a MBD & What is a Model?
- ❑ What is Model Development Process?
- ❑ What is a system?
- ❑ Automotive Market.
- ❑ MBD through V-Model.
- ❑ Software Architecture
- ❑ Why we need to shift to a MBD?
- ❑ Advantages of MBD.
- ❑ Model-Based Development Role.
- ❑ Use Cases for Model-Based Development of Automotive Embedded Software
- ❑ Guidelines
- ❑ Modeling Tools.
- ❑ Modeling Approach.
- ❑ in Loop.

MATLAB Basics Module:

- ❑ **Introduction To MATLAB**
  - ❑ what is a MATLAB?
  - ❑ Matrix Creation & Generation Data
  - ❑ Matrix Indexing
  - ❑ Mathematical Operation
  - ❑ Concatenation
  - ❑ Repating
- ❑ **Develop Programming skills and Proficiency in MATLAB**
  - ❑ Matrix OPeration
  - ❑ Input and Output Commands
  - ❑ Programming with MATLAB
  - ❑ Dealing with MATLAB functions

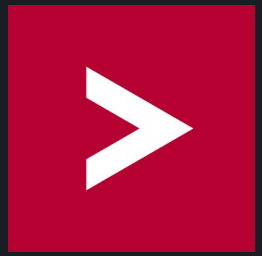
Interactive Sessions: MBD Concepts Module

- What is a MBD?
  - Model-based design/development is the practice of leveraging simulation to understand the behavior of a to-be-constructed or existing physical system.
- What is a Model?
  - A model is a representation of a real system focusing on some important aspects.
  - A model is a graphical representation of the system.
  - Models represent system behavior.
  - Models generate system code.
- Model Development Process



Interactive Sessions: MBD Concepts Module

- What is a system?
  - A "system" is a set of interconnected or interdependent components that work together to achieve a common purpose or goal.
- Automotive Market
  - OEMs
  - Tier 1
  - Tier 2



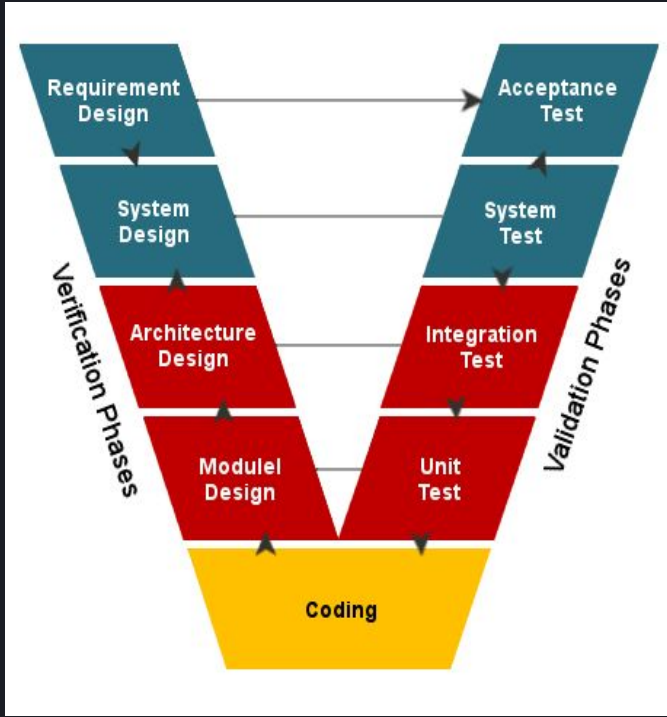
Interactive Sessions: MBD Concepts Module

- V-Model
  - The V-Model (Verification and Validation model) is a software development model used primarily in systems engineering and embedded software development.

- V-Model Overview
  - Sequential Development: The V-Model emphasizes a sequential development process where every development phase is directly linked to a corresponding testing phase.
  - Verification and Validation: The left side of the "V" represents the stages of verification (development), while the right side represents the stages of validation (testing).
  - Early Testing: Testing is planned early in the development process, ensuring that issues can be detected at each stage.

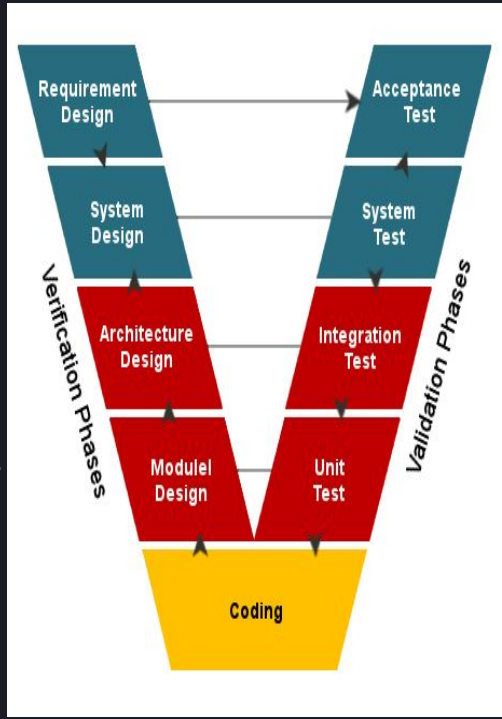
Interactive Sessions: MBD Concepts Module

- Key Phases of the V-Model
  - Left Side (Verification Phases)
    - Requirements Analysis:
      - Define system requirements from a functional perspective.
      - Outcome: Requirements Specification.
    - System Design:
      - Create high-level system architecture to meet the requirements.
      - Outcome: System Architecture Design.
    - Architectural Design
      - Break down the system into components and define interfaces.
      - Outcome: Detailed Design Specification.
    - Module Design:
      - Design individual modules with detailed logic and flow.
      - Outcome: Module Specifications.
    - Coding/Implementation:
      - Write the code for individual modules as per the design.
      - Outcome: Source Code.



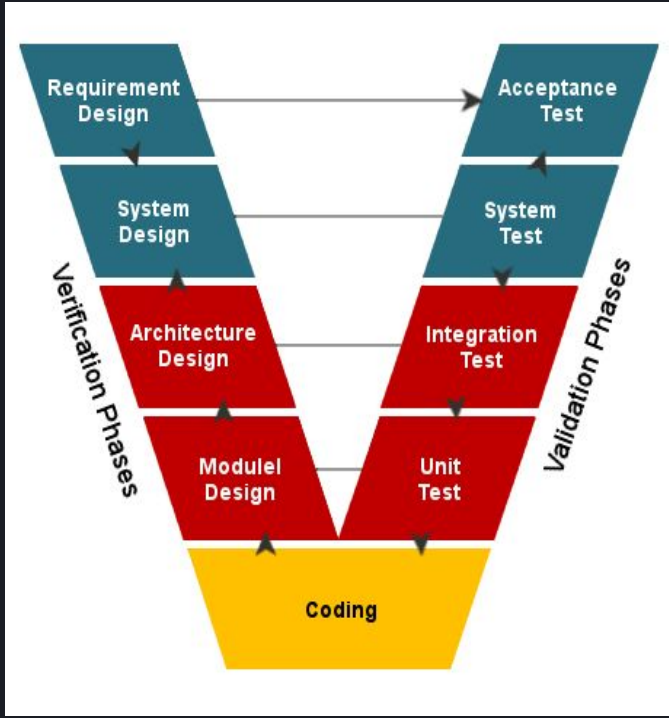
Interactive Sessions: MBD Concepts Module

- Key Phases of the V-Model
  - Right Side (Validation Phases)
    - Unit Testing:
      - Test individual modules to ensure they function as expected.
      - Linked to: Module Design phase.
    - Integration Testing:
      - Test interactions between integrated modules to verify correct communication.
      - Linked to: Architectural Design phase.
    - System Testing:
      - Test the complete system as a whole to verify it meets the system design.
      - Linked to: System Design phase.
    - Acceptance Testing:
      - Test the system against customer requirements to ensure it meets their needs.
      - Linked to: Requirements Analysis phase.



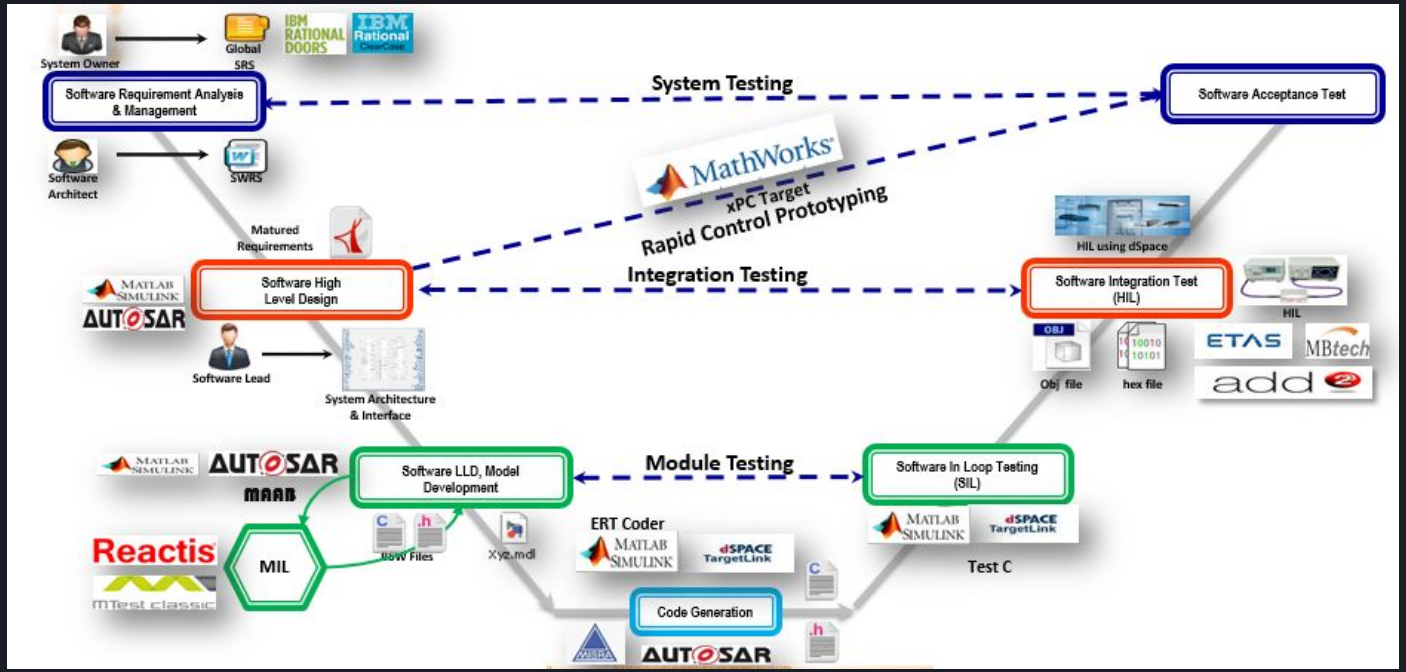
Interactive Sessions: MBD Concepts Module

- Advantages of the V-Model:
  - Clear Structure: Well-defined stages make the process easy to follow.
  - Early Detection of Defects: Testing is planned in parallel with each development stage.
  - Strong Verification and Validation: Ensures that the system meets both technical and user requirements.
- Disadvantages of the V-Model:
  - Rigidity: The V-Model is quite rigid and doesn't easily adapt to changes.
  - Late Prototype Availability: System integration is tested late in the process, after coding.



Interactive Sessions: MBD Concepts Module

- V-Model with MBD:



Interactive Sessions: MBD Concepts Module

- MBD Through the V-Model:
  - System Requirements Definition
    - In MBD, the first step is to capture system requirements, typically using high-level modeling tools like MATLAB and Simulink.
    - These requirements describe what the system should achieve and are the foundation for the entire development process
    - **MBD Role**: Requirements can be represented as models, helping to visualize the system's behavior and serving as the basis for further development.
  - System Design (High-Level Design):
    - The system is broken down into subsystems or components.
    - In MBD, this is done by creating a system-level model using Simulink, which provides a functional overview of the system.
    - **MBD Role**: The model represents the entire system architecture, and design decisions are validated through simulations

Interactive Sessions: MBD Concepts Module

- MBD Through the V-Model:
  - Component Design (Detailed Design):
    - The system components are designed in detail, with specific behavior and interfaces defined.
    - In MBD, each component can be represented by a detailed block or subsystem model in Simulink. The design is refined by running simulations to check component interaction and behavior.
    - **MBD Role**: Simulink models at this stage can be run through simulations for verification against requirements before coding begins.
  - Model Implementation (Code Generation):
    - MBD allows for automatic code generation (using tools like Simulink Coder or Embedded Coder) from the system model.
    - The model is transformed into C/C++ code that can be deployed onto the target hardware.
    - **MBD Role**: This step bridges the gap between simulation and implementation, ensuring that the code adheres to the validated model.

Interactive Sessions: MBD Concepts Module

- MBD Through the V-Model:
  - Unit Testing:
    - Individual components or subsystems are tested against their design specifications.
    - In MBD, this involves running simulations on each subsystem to ensure that it behaves correctly in isolation.
    - **MBD Role**: Model-in-the-Loop (MIL) testing is conducted, where the model is tested in a simulated environment.
  - Integration Testing:
    - Multiple components are integrated and tested to ensure they work together as intended.
    - **MBD Role**: Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL) testing is carried out to validate the behavior of the generated code in a simulated or real environment.

Interactive Sessions: MBD Concepts Module

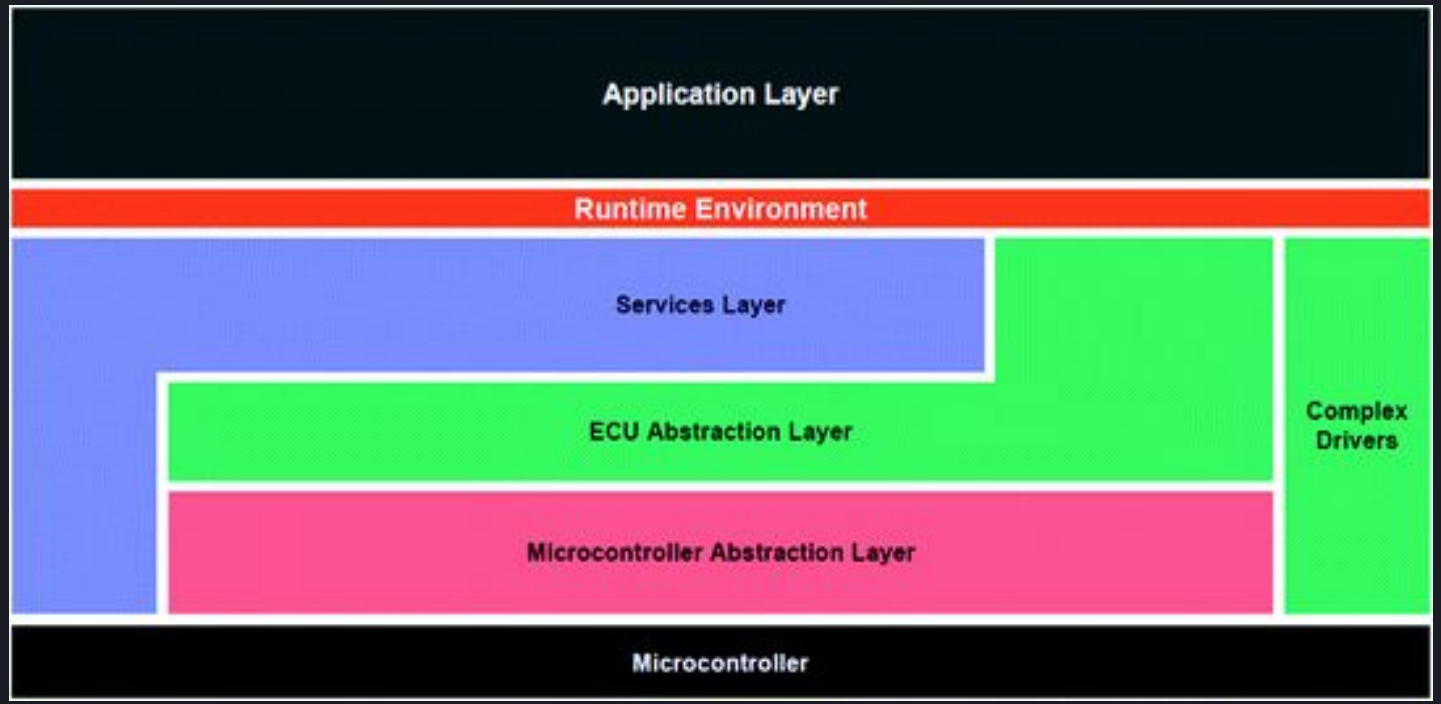
- MBD Through the V-Model:
  - System Testing:
    - The fully integrated system is tested against the system requirements.
    - **MBD Role**: Hardware-in-the-Loop (HIL) testing is conducted, where the generated code runs on real hardware, and the system is tested in real-time to ensure it meets the overall requirements.
  - Acceptance Testing:
    - The final product is validated against the original user requirements..
    - **MBD Role**: The system model and code are tested in real operational conditions to verify performance and safety.

Interactive Sessions: MBD Concepts Module

- Advantages of Using V-Model with MBD:
  - Early Verification: In MBD, simulations allow for early verification of designs even before the actual code is generated, aligning well with the verification phases in the V-Model.
  - Seamless Integration of Code and Models: MBD supports automatic code generation, ensuring that the design and implementation phases are tightly linked, reducing human error.
  - Continuous Testing: The combination of MIL, SIL, PIL, and HIL testing in MBD provides thorough verification and validation at every stage of the V-Model.
  - Requirement Traceability: In MBD, models can be linked directly to requirements, making it easier to trace and validate them through simulations and tests.

Interactive Sessions: MBD Concepts Module

- Software Architecture:



Interactive Sessions: MBD Concepts Module

- Testing-in-the-loop
  - **MIL (Model-in-the-Loop)**: Tests the system model entirely in a simulated environment without any hardware. It's useful for early-stage functional testing.
  - **SIL (Software-in-the-Loop)**: Tests the generated code in a simulated environment but without actual hardware. It checks the correctness of the code.
  - **PIL (Processor-in-the-Loop)**: Tests the generated code on the actual target processor, but still in a simulated environment. It helps verify code execution on the real processor.
  - **HIL (Hardware-in-the-Loop)**: Involves testing the actual hardware (embedded system or control unit) in interaction with a simulated real-time environment.

Interactive Sessions: MATLAB Basics Module

- Manual Matrix Creation
  - Row Vector

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.
```

  - Column Vector

```
colVec = [1; 2; 3; 4]; % Column vector with 4 elements
```

  - 2D Matrix

```
mat = [1 2 3; 4 5 6; 7 8 9]; % 3x3 matrix
```

Interactive Sessions: MATLAB Basics Module

- Zeros, Ones, and Identity Matrix
  - **Zeros Matrix**: Creates a matrix filled with zeros.

OSAcademy.m

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

OSAcademy.m

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

  - **Ones Matrix**: Creates a matrix filled with ones.

OSAcademy.m

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

OSAcademy.m

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

  - **Identity Matrix**: Creates an identity matrix (square matrix with ones on the diagonal).

OSAcademy.m

```
% Creating a 3x3 identity matrix
identityMatrix = eye(3);
```

OSAcademy.m

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

Interactive Sessions: MATLAB Basics Module

- Random Matrices
  - **Uniformly Distributed Random Numbers**:

```
R = rand(3, 3);  % 3x3 matrix of random numbers between 0 and 1
```

  - **Normally Distributed Random Numbers**:

```
N = randn(4, 4); % 4x4 matrix of normally distributed random numbers
```

  - **Random Integers**:

```
RI = randi([1, 10], 3, 3); % 3x3 matrix of random integers between 1 and 10
```

Interactive Sessions: MATLAB Basics Module

- Linearly Spaced Matrices
  - **Linearly spaced vector**:

```
L = linspace(0, 1, 5); % 5 points between 0 and 1, inclusive
```

  - **Colon Operator for sequences**:

```
C = 1:2:10; % Vector from 1 to 10, with a step size of 2 [1 3 5 7 9]
```

Interactive Sessions: MATLAB Basics Module

- Matrix of Special Values
  - **Diagonal Matrix**:

```
D = diag([1,2,3]); % 3x3 matrix with diagonal elements 1, 2, 3
```

  - **Triangular Matrices**:
    - Upper triangular

```
U = triu(rand(3, 3)); % Upper triangular matrix from random 3x3 matrix
```

    - Lower triangular

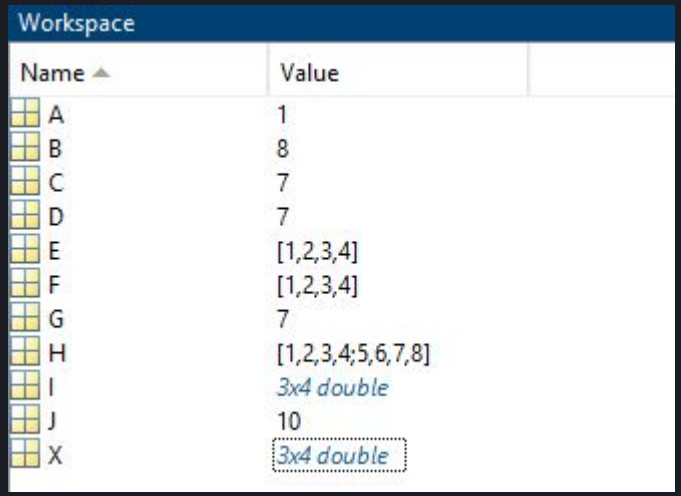
```
L = tril(rand(3, 3)); % Lower triangular matrix from random 3x3 matrix
```

Interactive Sessions: MATLAB Basics Module

- 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);
```

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



Interactive Sessions: MATLAB Basics Module

- **Horizontal Concatenation** (Column-wise)
  - Horizontal concatenation combines two or more matrices by placing them **side by side** (i.e., joining columns).
  - For horizontal concatenation, the number of rows in the matrices must match.

```
A = [1 2; 3 4]; % 2x2 matrix
B = [5 6; 7 8]; % 2x2 matrix

C = [A, B]; % Concatenates A and B horizontally
```

OSAcademy.m

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

```

OSAcademy.m

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

```

Interactive Sessions: MATLAB Basics Module

- **Vertical Concatenation** (Row-wise)
  - Vertical concatenation combines two or more matrices by placing them **one on top of the other** (i.e., joining rows).
  - For vertical concatenation, the number of columns in the matrices must match.

```
A = [1 2; 3 4]; % 2x2 matrix
B = [5 6; 7 8]; % 2x2 matrix

C = [A; B]; % Concatenates A and B vertically
```

OSAcademy.m

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

OSAcademy.m

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

Interactive Sessions: MATLAB Basics Module

- **What is the basic data type of MATLAB?**
  - double
  - single
  - uint8
  - int8
- **the result of size(m) command if m = 2**
  - 1
  - [1 1]
  - [1 2]
  - [2 2]
- **To define a matrix in MATLAB, use a set of**
  - **{ }**
  - **( )**
  - **[ ]**
  - **;**

Interactive Sessions: MATLAB Basics Module

- When inputting a matrix, each new element in a row is separated by a
  - :
  - ,
  - &
  - space and ,
- To combine 2 matrices in MATLAB, use set of
  - { }
  - ( )
  - [ ]
  - ;
- for the command m = zeros(3), what is the value of m
  - Square Matrix of zeros [0 0 0; 0 0 0; 0 0 0]
  - Row vector of three zeros [0 0 0]
  - Column vector of three zeros [0; 0; 0]
  - two dimensional matrix of zeros

Interactive Sessions: MATLAB Basics Module

- which of the following commands access the element with value ‘5’ of the matrix A where A = [1 2 3; 4 5 6; 7 8 9]
  - A(2,1)
  - A(2,2)
  - A(2,3)
  - A(9)
- A = [8 25; 42 11; -2 105] B = [52 0 13 -9; 44 62 31 102] What is the result of C = [A B]
- A = [8 25; 42 11; -2 105] B = [52 0 13 -9; 44 62 31 102] What is the result of C = [A; B]

- what is the output of command A = linspace(0.5,6.5,25)
  - create numeric array A that is a sequence of 25 numbers starting with 6.5 and with step 0.25
  - create numeric array A that is a sequence of 25 numbers starting with 0.5 and with step 0.5
  - create numeric array A that is a sequence of 25 numbers starting with 0.5 and with step 0.25
  - create numeric array A that is a sequence of 25 numbers starting with 25 and with step 0.5

Interactive Sessions: MATLAB Basics Module

- **Which of the following is not a property of MATLAB Script**
  - written inside .m file.
  - Accept input arguments and produce output arguments>
  - Execute a series of MATLAB statements
  - can be debugged using breakpoints
- What is the output of

```
for m = 1.0:-0.3:0.0
    disp(m)
end
```

Interactive Sessions: MATLAB Basics Module

- What is the output of

```
for m = [1 6 9 18]
    disp(m)
    if m == 9
        break;
    end
end
```

```
for m = [1 6 9 18]
    if m == 9
        continue;
    end
    disp(m)
end
```

Interactive Sessions: MATLAB Basics Module

- What is the output of

```
A = [1, 5, 3];
B = [2, 4, 6];
if all(A < B)
    result = 'All elements of A are less than corresponding elements of B';
else
    result = 'At least one element of A is not less than corresponding element of B';
end
disp(result);
```

Interactive Sessions: MATLAB Basics Module

- What is the output of

```
x = [4, 7, 1];
if any(x < 5) && all(x > 0)
    result = 'All elements are positive and at least one element is less than 5';
else
    result = 'Condition not met';
end
disp(result);
```

Interactive Sessions: MATLAB Basics Module

- What is the output of

```
status = 'error';
switch status
    case {'success', 'completed'}
        message = 'Operation was successful';
    case 'warning'
        message = 'Operation completed with warnings';
    case 'error'
        message = 'Operation failed';
    otherwise
        message = 'Unknown status';
end
disp(message);
```

Interactive Sessions: MATLAB Basics Module

- What is the output of

```
result = 0;
for i = 1:5
    if mod(i, 2) == 0
        result = result + i;
    else
        result = result - i;
    end
end
disp(result);
```

```
x = 1;
while x <= 10
    switch x
        case {2, 4, 6, 8, 10}
            disp([num2str(x), ' is even']);
        otherwise
            disp([num2str(x), ' is odd']);
    end
    x = x + 1;
end
```


Thank You!