ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION)

📖 What is Programming? Understanding Logic & Algorithms

📌 Chapter 1: Introduction to Programming

1.1 What is Programming?

Programming is the process of **writing instructions** that a computer can follow to perform specific tasks. These instructions, written in a programming language, tell the computer what to do and how to do it.

📌 Definition:

"Programming is the art of designing and coding algorithms to automate tasks and solve problems using a computer."

1.2 Why is Programming Important?

✓ **Automation** – Reduces manual work by making computers perform tasks.

✓ **Problem-Solving** – Helps find solutions using logic and algorithms.

✓ **Creativity** – Allows programmers to build websites, games, and applications.

✓ **Future Careers** – A fundamental skill in fields like AI, robotics, and cybersecurity.

1.3 Common Programming Languages

Different languages are used for different tasks:

- **Python** – Used in AI, data science, and web development.
- **JavaScript** – Helps create interactive websites.
- **C++** – Used in game development and system programming.
- **Scratch** – A beginner-friendly, block-based programming language.

📌 Chapter 2: Understanding Logic in Programming

2.1 What is Logic in Programming?

Programming logic refers to the **rules and steps** that a program follows to reach a solution. It ensures that the computer processes information correctly and makes the right decisions.

📌 **Example:** If you are thirsty, you **drink water**. If not, you do nothing. This is logic applied to everyday life.

2.2 Logical Thinking in Programming

✓ **Breaking a problem into smaller parts**

✓ **Understanding cause and effect relationships**

✓ **Using conditions (if-else statements) to make decisions**

✓ **Repeating steps using loops**

📌 **Example of Logical Thinking in Code:**

```
TEMPERATURE = 30

IF TEMPERATURE > 25:

  PRINT("IT'S HOT OUTSIDE, WEAR LIGHT CLOTHES.")

ELSE:

  PRINT("THE WEATHER IS COOL, WEAR A JACKET.")
```

✓ **EFFECT:** THE PROGRAM CHECKS THE TEMPERATURE AND SUGGESTS WHAT TO WEAR.

2.3 COMMON LOGICAL STRUCTURES IN PROGRAMMING

- **SEQUENTIAL EXECUTION** – INSTRUCTIONS ARE EXECUTED ONE AFTER ANOTHER.
- **CONDITIONAL EXECUTION (IF-ELSE)** – THE PROGRAM MAKES A DECISION BASED ON CONDITIONS.
- **LOOPS (FOR, WHILE)** – REPEATING A BLOCK OF CODE MULTIPLE TIMES.

📌 CHAPTER 3: WHAT IS AN ALGORITHM?

3.1 DEFINITION OF AN ALGORITHM

AN **ALGORITHM** IS A **STEP-BY-STEP SET OF INSTRUCTIONS** DESIGNED TO SOLVE A SPECIFIC PROBLEM. IT IS THE **FOUNDATION OF ALL COMPUTER PROGRAMS**.

📌 EXAMPLE:

MAKING A CUP OF TEA INVOLVES AN ALGORITHM:

1. BOIL WATER
2. ADD TEA LEAVES
3. POUR INTO A CUP

4. Add sugar/milk

5. Stir and serve

3.2 Characteristics of a Good Algorithm

✓ **Well-Defined Steps** – Clear and precise instructions.

✓ **Definite End** – The process must stop after a certain number of steps.

✓ **Effectiveness** – Solves the problem efficiently.

✓ **Finite** – Should complete in a limited time.

3.3 Writing a Simple Algorithm

📌 Example: Algorithm to Find the Largest of Two Numbers

1. Start
2. Input two numbers, A and B
3. If A > B, print "A is larger"
4. Else, print "B is larger"
5. End

3.4 Algorithm vs. Program

Algorithm	Program
A logical sequence of steps	A set of instructions written in a programming language
Language-independent	Requires coding knowledge
Helps in problem-solving	Executes tasks on a computer

📌 Chapter 4: Flowcharts – Visualizing Algorithms

4.1 What is a Flowchart?

A **flowchart** is a diagram that represents an algorithm using symbols. It helps programmers **visualize** the steps before writing code.

📌 Basic Flowchart Symbols:

- **Oval (Start/End)** – Represents the beginning or end of the process.
- **Rectangle (Process)** – Represents an operation (e.g., calculations, assignments).
- **Diamond (Decision)** – Represents a condition (e.g., if-else).
- **Arrow (Flowline)** – Indicates the flow of execution.

4.2 Example Flowchart: Checking if a Number is Even or Odd

[Start] ⟶ [Input Number] ⟶ [Is Number % 2 == 0?] ⟶ (Yes: Print "Even") ⟶ (No: Print "Odd") ⟶ [End]

✓ **Effect:** The program checks if a number is divisible by 2 and prints **"Even"** or **"Odd"** accordingly.

📌 Chapter 5: Exercises & Assignments

5.1 Multiple Choice Questions

1. What is programming?

- (a) Writing instructions for a computer ✅
- (b) Watching TV programs

- (C) Fixing computer hardware
- (D) None of the above

2. Which of the following is an example of a programming language?

- (A) Microsoft Word
- (B) Scratch ✅
- (C) Excel
- (D) Facebook

3. What is an algorithm?

- (A) A step-by-step process to solve a problem ✅
- (B) A type of computer virus
- (C) A hardware component
- (D) A mathematical formula

4. Which symbol is used for decision-making in a flowchart?

- (A) Rectangle
- (B) Oval
- (C) Diamond ✅
- (D) Arrow

5.2 Practical Assignments

📌 **Task 1:** Write an algorithm to make a peanut butter and jelly sandwich.

📌 **Task 2:** Draw a flowchart for an ATM withdrawal process.

📌 **Task 3:** Write a Python program to check if a number is even or odd.

```
num = int(input("Enter a number: "))

if num % 2 == 0:

  print("Even Number")

else:

  print("Odd Number")
```

📌 Chapter 6: Summary

✅ **Programming** is the process of writing instructions that a computer can understand.

✅ **Logic in programming** helps make decisions using conditions and loops.

✅ **Algorithms** are step-by-step instructions that solve problems.

✅ **Flowcharts** help visualize an algorithm before writing code.

✅ Learning **programming and logical thinking** is essential for problem-solving in robotics, AI, and automation.

📖 Introduction to Block-Based Coding (Scratch, LEGO Mindstorms)

📌 Chapter 1: What is Block-Based Coding?

1.1 Understanding Block-Based Coding

Block-based coding is a **visual programming method** where users create programs by **dragging and dropping code blocks** instead of writing text-based commands. Each block represents a specific function, such as movement, sound, loops, or conditions.

🔹 Why use block-based coding?

✓ Beginner-friendly – No need to memorize syntax

✓ Encourages logical thinking and problem-solving

✓ Reduces coding errors compared to text-based coding

✓ Ideal for young learners in robotics and game development

1.2 How Block-Based Coding Works

✓ **Blocks snap together** like puzzle pieces to create a sequence of commands.

✓ **Different block types** represent different programming concepts:

- **Motion Blocks** – Move objects in a specific direction.
- **Looks Blocks** – Change appearance or display messages.

- **Control Blocks** – Add loops and conditional statements.
- **Events Blocks** – Start a program when something happens.
- **Sound Blocks** – Add music and sound effects.

📌 **Example:** A simple block-based code in Scratch that makes a cat move:

[When Green Flag Clicked] → [Move 10 Steps] → [Say "Hello!"]

✓ **Effect:** The cat moves forward 10 steps and says "Hello!"

📌 Chapter 2: Introduction to Scratch

2.1 What is Scratch?

Scratch is a **block-based programming language** developed by MIT for young learners. It allows users to create animations, games, and interactive projects by connecting **color-coded blocks**.

🔹 Key Features of Scratch:

✓ **Sprite-Based Programming** – Control characters (sprites) with blocks.
✓ **Event-Driven Programming** – Start actions when certain events occur.
✓ **Easy Sharing** – Publish projects online for others to see and remix.
✓ **Built-in Libraries** – Includes sounds, images, and animations.

2.2 Scratch Interface Overview

✓ **Stage** – Where sprites perform actions.
✓ **Sprite List** – Displays all characters in the project.
✓ **Blocks Palette** – Contains draggable blocks for coding.
✓ **Scripts Area** – Where code blocks are assembled.

📌 Example: Making a Sprite Jump in Scratch

1. Drag **[When Space Key Pressed]** block.
2. Attach **[Change y by 20]** block (moves the sprite up).
3. Attach **[Wait 0.5 seconds]** block.
4. Attach **[Change y by -20]** block (brings the sprite back down).

✓ **Effect:** The sprite jumps when the space key is pressed.

📌 Chapter 3: Introduction to LEGO Mindstorms

3.1 What is LEGO Mindstorms?

LEGO Mindstorms is a **robotics platform** that uses **block-based programming** to build and control robots. It combines:

✓ **LEGO Bricks** – To construct robot bodies.
✓ **Motors & Sensors** – For movement and interaction.
✓ **EV3 Programming Interface** – A visual coding system for beginners.

3.2 LEGO Mindstorms Programming

LEGO Mindstorms uses a **drag-and-drop interface** similar to Scratch but designed for robotics.

📌 Basic Programming Blocks in LEGO Mindstorms:

✓ **Move Block** – Controls motor movement.

✓ **Wait Block** – Adds a delay before the next action.

✓ **Loop Block** – Repeats an action multiple times.

✓ **Sensor Blocks** – React to touch, color, or distance.

3.3 Example: Making a LEGO Robot Move Forward and Stop

1. Drag **[When Start Button Pressed]** block.
2. Attach **[Move Forward 5 Seconds]** block.
3. Attach **[Wait Until Touch Sensor Pressed]** block.
4. Attach **[Stop Motors]** block.

✓ **Effect:** The robot moves forward and stops when the touch sensor is pressed.

📌 Chapter 4: Comparison Between Scratch & LEGO Mindstorms

Feature	Scratch	LEGO Mindstorms
Purpose	Game & animation creation	Robotics programming
Interface	Drag-and-drop blocks	Drag-and-drop blocks
Best for	Beginners in coding	Beginners in robotics
Requires Hardware?	No	Yes (LEGO EV3 or SPIKE Prime)

Sensors & Motors	No	Yes

🔹 **Conclusion:** Scratch is ideal for **game development and storytelling,** while LEGO Mindstorms is best for **robotics and automation**.

📌 Chapter 5: Exercises & Assignments

5.1 Multiple Choice Questions

1. What is block-based coding?

- (a) A type of computer hardware
- (b) A way to write programs using text
- (c) A visual programming method using draggable blocks ✅
- (d) A robot-building process

2. What is the main programming language used in LEGO Mindstorms?

- (a) Python
- (b) JavaScript
- (c) Block-based programming ✅
- (d) Assembly language

3. In Scratch, what does the **"Motion"** block do?

- (a) Changes the sprite's appearance

- (B) Moves the sprite in a specific direction ✅
- (C) Starts the program
- (D) Stops all actions

4. Which of the following is a component of LEGO Mindstorms?

- (A) Sprites
- (B) Sensors ✅
- (C) HTML
- (D) Text editors

5.2 Practical Assignments

📌 **Task 1:** Create a simple **Scratch animation** where a sprite moves and says "Hello World!".

📌 **Task 2:** Build and program a **LEGO Mindstorms robot to move forward and stop at an obstacle**.

📌 **Task 3:** Write a short essay on **the benefits of block-based coding for young learners**.

📌 Chapter 6: Summary

✅ **Block-based coding** is a beginner-friendly way to learn programming.

✅ **Scratch** allows users to create animations, games, and interactive projects.

✅ **LEGO Mindstorms** is used for programming robots with

MOTORS AND SENSORS.

✅ BOTH TOOLS HELP DEVELOP **PROBLEM-SOLVING, LOGICAL THINKING, AND CREATIVITY** IN STUDENTS.

📖 Writing Basic Movement Commands for Robots

📌 Chapter 1: Introduction to Robot Movement

1.1 What is Robot Movement?

Robot movement refers to the ability of a robot to navigate, turn, and interact with its environment based on programmed instructions. Robots move using **motors, wheels, legs, or actuators,** depending on their design and purpose.

1.2 Why is Movement Important in Robotics?

✓ **Navigation** – Helps robots move from one point to another.
✓ **Automation** – Enables robots to complete tasks like picking up objects.
✓ **Interaction** – Allows robots to respond to their surroundings.
✓ **Problem-Solving** – Movement is essential for real-world applications like self-driving cars and delivery robots.

1.3 Types of Robot Movements

- **Wheeled Movement** – Used in mobile robots, cars, and drones.
- **Legged Movement** – Used in humanoid and quadruped robots.
- **Arm Movements** – Used in robotic arms for

MANUFACTURING AND SURGERY.

- **FLYING MOVEMENT** – USED IN DRONES AND AERIAL ROBOTS.

📌 CHAPTER 2: UNDERSTANDING BASIC ROBOT MOVEMENT COMMANDS

2.1 COMPONENTS INVOLVED IN ROBOT MOVEMENT

✓ **MOTORS** – CONVERT ELECTRICAL ENERGY INTO MOVEMENT.

✓ **WHEELS/LEGS** – PROVIDE STABILITY AND MOTION.

✓ **MICROCONTROLLER** – THE BRAIN OF THE ROBOT THAT PROCESSES MOVEMENT COMMANDS.

✓ **SENSORS** – HELP ROBOTS DETECT OBSTACLES AND ADJUST MOVEMENT.

2.2 MOVEMENT CONTROL IN ROBOTICS PROGRAMMING

IN PROGRAMMING, MOVEMENT COMMANDS ARE WRITTEN TO CONTROL:

✓ **SPEED** – HOW FAST THE ROBOT MOVES.

✓ **DIRECTION** – FORWARD, BACKWARD, LEFT, RIGHT.

✓ **ROTATION** – TURNING AT ANGLES OR SPINNING.

✓ **STOPPING** – HALTING MOVEMENT WHEN REQUIRED.

📌 **EXAMPLE:** IN LEGO MINDSTORMS BLOCK-BASED PROGRAMMING, A BASIC MOVEMENT COMMAND:

[START] → [MOVE FORWARD 5 SECONDS] → [STOP]

✓ **EFFECT:** THE ROBOT MOVES FORWARD FOR 5 SECONDS AND THEN STOPS.

📌 Chapter 3: Writing Basic Movement Commands in Block-Based Coding

3.1 Moving a Robot Forward and Backward

In **Scratch or LEGO Mindstorms,** movement blocks control robot direction:

📌 Scratch Example:

[When Green Flag Clicked] → [Move 10 Steps Forward]

✓ **Effect:** The robot moves forward by 10 steps.

📌 LEGO Mindstorms Example:

1. Drag **Move Forward** block.
2. Set **Speed = 50%** and **Time = 3 seconds**.
3. Attach a **Stop Motors** block.

✓ **Effect:** The robot moves forward for 3 seconds and then stops.

3.2 Making a Robot Turn

To change direction, we program the robot to **turn left or right**:

📌 LEGO Mindstorms Example:

1. Drag **Turn Right** block.
2. Set **Rotation = 90 degrees**.
3. Attach a **Move Forward** block.

✓ **Effect:** The robot turns right by 90 degrees and then moves forward.

📌 Python Example for Turning

```
robot.turn_left(90)  # Robot turns left by 90 degrees
```

3.3 Using Loops to Repeat Movement

Loops allow robots to repeat movements **continuously** or for a set number of times.

📌 Example in LEGO Mindstorms:

1. Drag **Loop Block** (Repeat 3 times).
2. Inside loop, add **Move Forward** (2 seconds) and **Turn Right 90°**.

✓ **Effect:** The robot moves in a square pattern.

📌 Python Example:

```
for i in range(4):
    robot.move_forward(2)
    robot.turn_right(90)
```

✓ **Effect:** The robot moves in a square loop.

3.4 Stopping the Robot

Robots need stopping conditions to avoid collisions or complete tasks.

📌 Example of Stopping a Robot in Python:

```
robot.move_forward(5)
robot.stop()
```

✓ **Effect:** The robot moves forward for 5 seconds and stops.

📌 **LEGO Mindstorms Example:**

1. Drag **Wait Until Touch Sensor Pressed** block.
2. Attach **Stop Motors** block.

✓ **Effect:** The robot stops when it touches an obstacle.

📌 Chapter 4: Exercises & Assignments

4.1 Multiple Choice Questions

1. What is the purpose of motors in a robot?
   - (a) To power the sensors
   - (b) To convert electrical energy into movement ✅
   - (c) To store programming commands
   - (d) To increase battery life
2. Which programming block makes a robot move forward?
   - (a) Turn Block
   - (b) Move Block ✅
   - (c) Stop Block
   - (d) Rotate Block
3. What is the purpose of a loop in robot programming?
   - (a) To stop the robot from moving
   - (b) To repeat movements ✅

- (C) To slow down the robot
- (D) To detect obstacles

4.2 Practical Assignments

📌 **Task 1:** Write a program to move a robot forward, stop for 2 seconds, and then move backward.

📌 **Task 2:** Draw a flowchart for programming a robot to **turn left and avoid an obstacle**.

📌 **Task 3:** In LEGO Mindstorms, write a block-based program to make the robot **move in a square path using loops**.

📌 Chapter 5: Summary

✅ Robots move using **motors, wheels, sensors, and programming commands**.

✅ Basic movement commands include **moving forward, backward, turning, and stopping**.

✅ Loops allow **repeating movements**, while **conditions help robots make decisions**.

✅ Block-based coding (Scratch, LEGO Mindstorms) makes programming robot movements easy.

📖 Loops & Conditional Statements in Robot Programming

📌 Chapter 1: Introduction to Loops & Conditional Statements

1.1 What are Loops & Conditional Statements?

Loops and conditional statements are essential programming concepts used in **robot programming** to control robot actions efficiently.

✓ **Loops** – Allow robots to repeat an action multiple times.
✓ **Conditional Statements** – Enable robots to make decisions based on conditions.

1.2 Why are Loops & Conditions Important in Robotics?

🔹 **Automation** – Robots can perform repetitive tasks without human input.
🔹 **Decision-Making** – Robots can respond to environmental changes using sensors.
🔹 **Efficiency** – Reduces code complexity by eliminating redundant instructions.

📌 Example in a Self-Driving Car:

- If the car detects an obstacle, **stop**.
- Otherwise, **keep moving forward**.

```
IF OBSTACLE_DETECTED:
  STOP_CAR()
```

```
ELSE:
  MOVE_FORWARD()
```

📌 Chapter 2: Understanding Loops in Robot Programming

2.1 What is a Loop?

A **loop** is a programming structure that repeats a set of instructions multiple times. Loops help robots perform tasks **continuously** or **until a condition is met**.

2.2 Types of Loops in Robot Programming

🔹 1. While Loop – Repeats a task as long as a condition is true.

📌 **Example:** A robot moves forward **until** it detects an obstacle.

```
WHILE NOT OBSTACLE_DETECTED():
  MOVE_FORWARD()
STOP()
```

✓ **Effect:** The robot moves forward and stops when an obstacle appears.

🔹 2. For Loop – Repeats a task a fixed number of times.

📌 **Example:** A robot blinks an LED light **5 times**.

```
FOR I IN RANGE(5):
  TURN_ON_LED()
  TURN_OFF_LED()
```

✓ **Effect:** The LED turns on and off 5 times.

🔹 **3. Infinite Loop – Runs forever until manually stopped.**

📌 **Example:** A robot keeps scanning for objects indefinitely.

```
WHILE TRUE:
  SCAN_FOR_OBJECTS()
```

✓ **Effect:** The robot **continuously** scans for objects.

📌 Chapter 3: Understanding Conditional Statements in Robot Programming

3.1 What is a Conditional Statement?

A **conditional statement** is used to make decisions based on conditions (true/false). In robotics, conditions often depend on **sensor inputs**.

3.2 Common Conditional Statements in Robotics

🔹 **1. If-Else Statement**

Used when a robot needs to **choose between two actions**.

📌 **Example:** A robot checks if an obstacle is ahead.

```
IF OBSTACLE_DETECTED():
  STOP()
ELSE:
  MOVE_FORWARD()
```

✓ **Effect:** If an obstacle is detected, the robot stops. Otherwise, it moves forward.

🔹 2. If-Elif-Else Statement

Used when there are **multiple conditions**.

📌 **Example:** A robot detects different colors and performs different actions.

```
IF COLOR_SENSOR() == "RED":
  STOP()
ELIF COLOR_SENSOR() == "GREEN":
  MOVE_FORWARD()
ELSE:
  TURN_LEFT()
```

✓ **Effect:** The robot **stops on red, moves on green,** and **turns left on other colors**.

🔹 3. Nested If Statements

A condition inside another condition.

📌 **Example:** A robot stops **only** if an obstacle is near and the speed is high.

```
IF OBSTACLE_DETECTED():
  IF SPEED > 5:
    STOP()
```

✓ **Effect:** The robot stops only if both conditions are met.

📌 Chapter 4: Combining Loops & Conditionals in Robot Programming

4.1 Using Loops & Conditions Together

✓ **Robots need both loops and conditionals** to function effectively.

✓ **Loops** handle repetitive actions, while **conditions** decide when to stop or change behavior.

📌 Example: A Line-Following Robot

```
WHILE TRUE:
  IF COLOR_SENSOR() == "BLACK":
    MOVE_FORWARD()
  ELSE:
    TURN_RIGHT()
```

✓ **Effect:** The robot keeps following a black line and turns when the color changes.

📌 Chapter 5: Exercises & Assignments

5.1 Multiple Choice Questions

1. What does a loop do in robot programming?

- (a) Makes a robot stop
- (b) Repeats a set of instructions ✅
- (c) Makes a robot jump
- (d) Detects obstacles

2. Which loop runs **until a condition becomes false**?

- (A) For Loop
- (B) While Loop ✅
- (C) Infinite Loop
- (D) Nested Loop

3. What does an **if-else statement** do?

- (A) Repeats an action
- (B) Selects an action based on conditions ✅
- (C) Stops the program
- (D) Stores data

4. What is the purpose of an infinite loop?

- (A) To repeat a task forever ✅
- (B) To stop the robot
- (C) To slow down a robot
- (D) To detect objects

5.2 Practical Assignments

📌 **Task 1:** Write an algorithm for a robot that moves forward until it detects an obstacle, then turns right.

📌 **Task 2:** Draw a **flowchart** for a robot that turns on its LED light **only when it's dark**.

📌 **Task 3:** Modify the following code so the robot stops when it detects a red object:

```
while True:
  move_forward()
```

📌 Chapter 6: Summary

✅ **Loops** help robots **repeat** actions automatically.

✅ **Conditional statements** allow robots to **make decisions** based on sensor data.

✅ **While loops** run **until a condition is met,** while **for loops** repeat a fixed number of times.

✅ **If-else statements** help robots choose between different actions.

✅ **Loops & conditions** work together to create **smart and autonomous robots.**

📌 ASSIGNMENT:

☑ CREATE **A FLOWCHART** SHOWING HOW A ROBOT FOLLOWS AN OBSTACLE PATH.

📌 Assignment Solution: Create a Flowchart Showing How a Robot Follows an Obstacle Path

🎯 Objective:

This step-by-step guide will help you design a flowchart that illustrates how a robot navigates an obstacle path using sensors and programmed logic.

🛠 Step 1: Understand the Problem

Before creating the flowchart, analyze how a robot moves and avoids obstacles. The robot must:

✓ Move forward until it detects an obstacle.
✓ If an obstacle is detected, stop and decide a new direction.
✓ Turn left or right to find a clear path.
✓ Continue moving forward and repeat the process.

📌 **Example:** Imagine a **self-driving vacuum cleaner** navigating around furniture.

📌 Step 2: Identify Key Steps in the Process

Break the obstacle-following process into simple **decision-based steps**:

1. **START**
2. MOVE FORWARD
3. CHECK FOR OBSTACLES USING A **SENSOR (ULTRASONIC, INFRARED, OR TOUCH)**
4. **IF NO OBSTACLE DETECTED → CONTINUE MOVING FORWARD**
5. **IF AN OBSTACLE IS DETECTED → STOP**
6. **DECIDE NEW DIRECTION:**
   - TURN **LEFT** IF THE LEFT PATH IS CLEAR.
   - ELSE, TURN **RIGHT** IF THE RIGHT PATH IS CLEAR.
   - IF BOTH PATHS ARE BLOCKED, **TURN AROUND (180°)**.
7. RESUME MOVING FORWARD.
8. REPEAT UNTIL THE ROBOT REACHES THE DESTINATION.
9. **END**

📌 STEP 3: DRAW THE FLOWCHART SYMBOLS

A FLOWCHART USES **STANDARD SYMBOLS** TO REPRESENT DIFFERENT ACTIONS:

🔹 **OVAL (START/END)** – REPRESENTS THE BEGINNING OR END OF THE PROCESS.

🔹 **RECTANGLE (PROCESS)** – REPRESENTS AN ACTION (E.G., MOVE FORWARD, STOP).

🔹 **DIAMOND (DECISION/CONDITION)** – REPRESENTS A CONDITION (E.G., IS THERE AN OBSTACLE?).

🔹 **ARROWS (FLOWLINES)** – SHOW THE DIRECTION OF THE PROCESS.

📌 Step 4: Create the Flowchart

Now, arrange the symbols in a logical order. Here's the **flowchart structure**:

[Start]

↓

[Move Forward]

↓

[Obstacle Detected?] → No → [Continue Moving Forward]

↓ Yes

[Stop Robot]

↓

[Left Path Clear?] → Yes → [Turn Left]

↓ No

[Right Path Clear?] → Yes → [Turn Right]

↓ No

[Turn Around]

↓

[Move Forward]

↓

[Repeat Until Destination Reached]

↓

[End]

📌 Step 5: Create a Digital or Hand-Drawn Flowchart

You can create the flowchart using:

✓ **Paper & Pencil** – Draw using flowchart symbols.
✓ **Online Tools** – Use Lucidchart, Draw.io, or Canva.
✓ **MS PowerPoint/Word** – Use built-in flowchart shapes.

📌 **Example of a Simple Flowchart:** (Imagine a **robotic vacuum cleaner moving and avoiding furniture**)

● Start

↓

■ Move Forward

↓

🔷 Is there an obstacle?

→ No → Continue Forward

→ Yes → Stop

↓

🔷 Is left clear?

→ Yes → Turn Left

→ No → Check Right

↓

🔷 Is right clear?

→ Yes → Turn Right

→ No → Turn Around

↓

■ Resume Moving

↓

● End

📌 Step 6: Review and Submit the Assignment

✓ **Check for missing steps** and ensure all decisions are covered.

✓ **Label flowchart symbols correctly** (Start, Stop, Decision, Action).

✓ **Submit the flowchart as a hand-drawn image or a digital file.**
