Ex1:

```
function sysCall_init()
  RM=sim.getObject("../RM")
  LM=sim.getObject("../LM")
   -- do some initialization here
end

function sysCall_actuation()
   message, data, data2 = sim.getSimulatorMessage()
   print(message, data)

      if(message==sim.message_keypress)then
      if(data[1]==2007)then
         -- Forward
         sim.setJointTargetVelocity(RM,-5)
         sim.setJointTargetVelocity(LM,-5)
      end

      if(data[1]==2008)then
         -- Backward
         sim.setJointTargetVelocity(RM,5)
         sim.setJointTargetVelocity(LM,5)
      end

      if(data[1]==2009)then
         -- LeftTurn
         sim.setJointTargetVelocity(RM,-5)
         sim.setJointTargetVelocity(LM,5)
      end

      if(data[1]==2010)then
```

```
        -- RightTurn
        sim.setJointTargetVelocity(RM,5)
        sim.setJointTargetVelocity(LM,-5)
    end

    else
        -- Stop
        sim.setJointTargetVelocity(RM,0)
        sim.setJointTargetVelocity(LM,0)
    end

end

function sysCall_sensing()
    -- put your sensing code here
end

function sysCall_cleanup()
    -- do some clean-up here
end

-- See the user manual or the available code snippets for additional callback functions and details
```

Ex2:

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT); // Initialize the in-built LED pin as an output
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // Turn the LED on
  delay(1000);                     // Wait for 1 second (1000 milliseconds)
  digitalWrite(LED_BUILTIN, LOW);  // Turn the LED off
  delay(1000);                     // Wait for 1 second
}
```

Ex3:

```
// Define Motor Pins
#define IN1 2
#define IN2 3
#define IN3 4
#define IN4 5
#define IN5 6
#define IN6 7
#define IN7 8
#define IN8 9

char command;  // To store the incoming Bluetooth data

void setup() {
  // Set motor pins as output
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(IN5, OUTPUT);
  pinMode(IN6, OUTPUT);
  pinMode(IN7, OUTPUT);
  pinMode(IN8, OUTPUT);

  // Start Serial Communication with Bluetooth module
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
```

```
    command = Serial.read();
    moveRobot(command);
  }
}

void moveRobot(char cmd) {
  switch (cmd) {
    case '1': // Forward
      moveForward();
      break;
    case '2': // Backward
      moveBackward();
      break;
    case '3': // Left
      turnLeft();
      break;
    case '4': // Right
      turnRight();
      break;
    case '5': // Stop
      stopRobot();
      break;
    default:
      stopRobot();
      break;
  }
}

void moveForward() {
  // Motor Driver 1
  digitalWrite(IN1, HIGH);
```

```
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);

  // Motor Driver 2
  digitalWrite(IN5, HIGH);
  digitalWrite(IN6, LOW);
  digitalWrite(IN7, HIGH);
  digitalWrite(IN8, LOW);
}

void moveBackward() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);

  digitalWrite(IN5, LOW);
  digitalWrite(IN6, HIGH);
  digitalWrite(IN7, LOW);
  digitalWrite(IN8, HIGH);
}

void turnLeft() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);

  digitalWrite(IN5, LOW);
  digitalWrite(IN6, HIGH);
```

```
  digitalWrite(IN7, HIGH);
  digitalWrite(IN8, LOW);
}

void turnRight() {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);

  digitalWrite(IN5, HIGH);
  digitalWrite(IN6, LOW);
  digitalWrite(IN7, LOW);
  digitalWrite(IN8, HIGH);
}

void stopRobot() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);

  digitalWrite(IN5, LOW);
  digitalWrite(IN6, LOW);
  digitalWrite(IN7, LOW);
  digitalWrite(IN8, LOW);
}
```

Ex4:

```
// Motor Driver 1 Pins
#define IN1 2
#define IN2 3
#define IN3 4
#define IN4 5

// Motor Driver 2 Pins
#define IN5 6
#define IN6 7
#define IN7 8
#define IN8 9

// Ultrasonic Sensor Pins
#define trigPin 10
#define echoPin 11

// Distance threshold (in cm)
int distanceThreshold = 20;

void setup() {
  // Set motor pins as output
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(IN5, OUTPUT);
  pinMode(IN6, OUTPUT);
  pinMode(IN7, OUTPUT);
  pinMode(IN8, OUTPUT);
```

```
  // Set ultrasonic pins
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  // Begin Serial (optional, for debugging)
  Serial.begin(9600);
}

void loop() {
  int distance = readDistance();

  if (distance > 0 && distance < distanceThreshold) {
    // Obstacle detected
    stopRobot();
    delay(500);      // Pause
    turnRight();
    delay(600);      // Adjust turning time
  } else {
    // No obstacle
    moveForward();
  }
}

// Function to read distance from ultrasonic sensor
int readDistance() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
```

```
  long duration = pulseIn(echoPin, HIGH);
  int distance = duration * 0.034 / 2;  // cm

  Serial.print("Distance: ");
  Serial.println(distance);

  return distance;
}

// Functions for Motor Movement
void moveForward() {
  // Motor Driver 1
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);

  // Motor Driver 2
  digitalWrite(IN5, HIGH);
  digitalWrite(IN6, LOW);
  digitalWrite(IN7, HIGH);
  digitalWrite(IN8, LOW);
}

void turnRight() {
  // Motor Driver 1
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
```

```
  // Motor Driver 2
  digitalWrite(IN5, LOW);
  digitalWrite(IN6, HIGH);
  digitalWrite(IN7, HIGH);
  digitalWrite(IN8, LOW);
}

void stopRobot() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);

  digitalWrite(IN5, LOW);
  digitalWrite(IN6, LOW);
  digitalWrite(IN7, LOW);
  digitalWrite(IN8, LOW);
}
```

Ex5:

```
# Import necessary libraries
import tensorflow as tf
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Labels (0, 1, 2)

# Binary classification: Make it 0 (Setosa) vs 1 (Not Setosa)
y_binary = (y == 0).astype(int)  # Setosa -> 1, Others -> 0

# Split into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y_binary, test_size=0.2, random_state=42
)

# Feature Scaling (important for faster training)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1, activation='sigmoid', input_shape=(4,))
])
```

```
# Compile the model
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Train the model
model.fit(X_train, y_train, epochs=100, verbose=1)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"\nTest Accuracy: {accuracy:.2f}")
```