Codes

1. Experiment: Rainbow LED Chaser

Tinkercard

Components Needed:

- Arduino Uno board
- Seven LEDs (Red, Orange, Yellow, Green, Blue, Indigo, Violet)
- Seven 220-ohm resistors
- Breadboard
- Jumper wires

Logic:

1. Initialize the array of LEDs and set the pin numbers for each LED.
2. Create a loop that will run indefinitely.
3. In each iteration of the loop:
    - Turn on each LED in sequence from left to right.
    - Wait for a short duration to allow the LED to be visible.
    - Turn off the LED after the wait.
    - Repeat the process in reverse order from right to left.
4. Continue this process to create a chaser effect.

Pseudo Code:

```
initialize ledPins = [2, 3, 4, 5, 6, 7, 8]  // Array of LED
pin numbers
initialize numLeds = length(ledPins)

loop:
    // Chase from left to right
    for i from 0 to numLeds - 1:
        turnOn(ledPins[i])  // Turn on the current LED
        wait(500)           // Wait for 500 milliseconds
```

```
        turnOff(ledPins[i]) // Turn off the current LED

    // Chase from right to left
    for i from numLeds - 1 down to 0:
        turnOn(ledPins[i])  // Turn on the current LED
        wait(500)           // Wait for 500 milliseconds
        turnOff(ledPins[i]) // Turn off the current LED
```

---

2. Experiment: LED and Push Button Application

Tinkercard

Components:

- 1 x Arduino Uno board
- 2 x LEDs (any color)
- 1 x Push button
- 3 x 220-ohm resistors
- Breadboard
- Jumper wires

Logic:

1. Initialize the two LEDs and the push button.
2. Set the initial state of both LEDs to OFF.
3. Continuously monitor the state of the push button.
4. If the push button is pressed and released:
    - If both LEDs are OFF, turn them ON.
    - If both LEDs are ON, turn them OFF.
5. If the push button is held down:
    - Alternate the state of the LEDs (ON/OFF) at a defined interval.

Pseudo Code:

```
initialize led1Pin = 9  // Pin for LED 1
initialize led2Pin = 10 // Pin for LED 2
```

```
initialize buttonPin = 2 // Pin for the push button

initialize led1State = OFF
initialize led2State = OFF
initialize buttonState = LOW

loop:
    buttonState = read(buttonPin) // Read the state of the
push button

    if buttonState == HIGH: // Button is pressed
        waitUntilReleased(buttonPin) // Wait until the butt
on is released

        if led1State == OFF and led2State == OFF:
            turnOn(led1Pin)  // Turn on LED 1
            turnOn(led2Pin)  // Turn on LED 2
            led1State = ON
            led2State = ON
        else if led1State == ON and led2State == ON:
            turnOff(led1Pin) // Turn off LED 1
            turnOff(led2Pin) // Turn off LED 2
            led1State = OFF
            led2State = OFF

    if buttonState == HIGH: // Button is held down
        toggleInterval = 500 // Set toggle interval (500 mi
lliseconds)
        while buttonState == HIGH: // While the button is h
eld down
            turnOn(led1Pin)  // Turn on LED 1
            turnOff(led2Pin) // Turn off LED 2
            wait(toggleInterval)
            turnOff(led1Pin) // Turn off LED 1
            turnOn(led2Pin)  // Turn on LED 2
            wait(toggleInterval)
```

3. Experiment: LED-RGB and Ultrasonic Distance Application

Tinkercard

Components:

- 1 x Arduino Uno board
- 1 x RGB LED
- 2 x Ultrasonic distance sensors (HC-SR04)
- 3 x 220-ohm resistors
- Breadboard
- Jumper wires

Logic:

1. Initialize the RGB LED and the two ultrasonic distance sensors.
2. Continuously read the distance values from both ultrasonic sensors.
3. Compare the distance values:
    - If the first sensor detects an object closer than the second sensor, set the RGB LED to one color (e.g., red).
    - If the second sensor detects an object closer than the first sensor, set the RGB LED to another color (e.g., blue).
    - If both sensors detect objects at the same distance, set the RGB LED to a third color (e.g., green).
4. Repeat the process to continuously update the RGB LED color based on the distance readings.

Pseudo Code:

```
initialize rgbLedPins = [9, 10, 11] // Pins for Red, Green,
Blue of RGB LED
initialize ultrasonicSensor1 = {trigPin1, echoPin1} // Pins
for first ultrasonic sensor
initialize ultrasonicSensor2 = {trigPin2, echoPin2} // Pins
for second ultrasonic sensor

loop:
    distance1 = readUltrasonic(ultrasonicSensor1) // Read d
```

```
istance from first sensor
    distance2 = readUltrasonic(ultrasonicSensor2) // Read d
istance from second sensor

    if distance1 < distance2:
        setColor(rgbLedPins, RED) // Set RGB LED to red
    else if distance2 < distance1:
        setColor(rgbLedPins, BLUE) // Set RGB LED to blue
    else:
        setColor(rgbLedPins, GREEN) // Set RGB LED to green

function readUltrasonic(sensor):
    // Trigger the ultrasonic sensor and calculate distance
    trigger(sensor.trigPin)
    duration = pulseIn(sensor.echoPin, HIGH)
    distance = duration * 0.034 / 2 // Calculate distance i
n cm
    return distance

function setColor(rgbPins, color):
    if color == RED:
        write(rgbPins[0], HIGH) // Turn on red
        write(rgbPins[1], LOW)  // Turn off green
        write(rgbPins[2], LOW)  // Turn off blue
    else if color == BLUE:
        write(rgbPins[0], LOW)  // Turn off red
        write(rgbPins[1], LOW)  // Turn off green
        write(rgbPins[2], HIGH) // Turn on blue
    else if color == GREEN:
        write(rgbPins[0], LOW)  // Turn off red
        write(rgbPins[1], HIGH) // Turn on green
        write(rgbPins[2], LOW)  // Turn off blue
```

4. Experiment: Piezo Sensor, IR Sensor, and Servo Motor

Tinkercard

Tinkercard

Components:

- 1 x Arduino Uno board
- 1 x Piezo Sensor (Buzzer)
- 1 x IR Sensor
- 1 x Servo Motor
- Breadboard
- Jumper wires
- Resistors (as needed)

Logic:

1. Initialize the piezo sensor, IR sensor, and servo motor.
2. Continuously monitor the state of the IR sensor to detect motion.
3. If motion is detected by the IR sensor:
    - Activate the piezo sensor to play a doorbell sound.
    - Move the servo motor to simulate the door opening slightly.
4. Ensure that the servo motor returns to its original position after a short delay.
5. Repeat the process to continuously check for motion.

Pseudo Code:

```
initialize piezoPin = 8      // Pin for the piezo sensor
initialize irSensorPin = 2   // Pin for the IR sensor
initialize servoPin = 9      // Pin for the servo motor

initialize servoPosition = 0  // Initial position of the se
rvo motor

loop:
    motionDetected = read(irSensorPin) // Read the state of
the IR sensor

    if motionDetected == HIGH: // If motion is detected
        activatePiezo(piezoPin) // Activate the piezo senso
r to play sound
```

```
        moveServo(servoPin, 90)  // Move the servo motor to
open the door
        wait(1000)                // Wait for 1 second
        moveServo(servoPin, 0)    // Move the servo motor b
ack to original position

function activatePiezo(pin):
    write(pin, HIGH) // Turn on the piezo sensor
    wait(500)        // Wait for 500 milliseconds
    write(pin, LOW)  // Turn off the piezo sensor

function moveServo(pin, position):
    // Move the servo motor to the specified position
    write(pin, position) // Set the servo to the desired an
gle
```

5. Experiment: Photo-resistor Sensor, Temperature Sensor, and DC Motor

Tinkercard

Components:

- 1 x Arduino Uno board
- 1 x Photo-resistor Sensor (LDR)
- 1 x Temperature Sensor (LM35)
- 1 x DC Motor (for fan control)
- 1 x Transistor (e.g., NPN transistor like 2N2222)
- 1 x Diode (e.g., 1N4007)
- 1 x 10k ohm Resistor
- 1 x 220 ohm Resistor
- Breadboard
- Jumper wires

Logic:

1. Initialize the photo-resistor sensor, temperature sensor, and DC motor.

2. Continuously read the values from the photo-resistor and temperature sensor.
3. Determine the appropriate actions based on the sensor readings:
    - If the ambient light level is below a certain threshold, turn on the DC motor to activate the fan.
    - If the temperature exceeds a predefined limit, also turn on the DC motor to cool the environment.
    - If both conditions are met, keep the DC motor running.
    - If both conditions are not met, turn off the DC motor.
4. Repeat the process to continuously monitor and adjust the motor's state based on sensor readings.

Pseudo Code:

```
initialize photoResistorPin = A0  // Pin for the photo-resi
stor sensor
initialize temperatureSensorPin = A1 // Pin for the tempera
ture sensor
initialize dcMotorPin = 9          // Pin for the DC motor

initialize lightThreshold = 300     // Threshold for light
level
initialize temperatureThreshold = 25 // Threshold for tempe
rature in Celsius

loop:
    lightLevel = read(photoResistorPin) // Read the light l
evel from the photo-resistor
    temperature = readTemperature(temperatureSensorPin) //
Read the temperature

    if lightLevel < lightThreshold or temperature > tempera
tureThreshold:
        turnOnMotor(dcMotorPin) // Turn on the DC motor (fa
n)
    else:
        turnOffMotor(dcMotorPin) // Turn off the DC motor
(fan)
```

```
function readTemperature(pin):
    // Read the temperature from the temperature sensor
    rawValue = analogRead(pin) // Read the analog value
    temperature = (rawValue * 5.0 * 100.0) / 1024 // Conver
t to Celsius (example conversion)
    return temperature

function turnOnMotor(pin):
    write(pin, HIGH) // Set the pin to HIGH to turn on the
motor

function turnOffMotor(pin):
    write(pin, LOW) // Set the pin to LOW to turn off the m
otor
```

6. Experiment: Gas Sensor, PIR Motor Sensor, and DC Motor (for Fan Control)

Tinkercard

Components:

- 1 x Arduino Uno board
- 1 x Gas Sensor (MQ-2 or MQ-135)
- 1 x PIR Sensor
- 1 x DC Motor (for fan control)
- 1 x Transistor (e.g., NPN transistor like 2N2222)
- 1 x Diode (e.g., 1N4007)
- 1 x 10k ohm Resistor
- 1 x 220 ohm Resistor
- Breadboard
- Jumper wires

Logic:

1. Initialize the gas sensor, PIR sensor, and DC motor.

2. Continuously monitor the readings from the gas sensor and the PIR sensor.
3. If the gas sensor detects harmful gas levels:
    - Check if the PIR sensor detects motion (indicating room occupancy).
    - If motion is detected, activate the DC motor to turn on the fan for ventilation.
4. If no harmful gas is detected, or if the PIR sensor does not detect motion, turn off the DC motor.
5. Repeat the process to continuously monitor the sensors and control the fan accordingly.

Pseudo Code:

```
initialize gasSensorPin = A0      // Pin for the gas sensor
initialize pirSensorPin = 2        // Pin for the PIR senso
r
initialize dcMotorPin = 9          // Pin for the DC motor

initialize gasThreshold = 400       // Threshold for gas le
vel detection

loop:
    gasLevel = read(gasSensorPin)  // Read the gas level fr
om the gas sensor
    motionDetected = read(pirSensorPin) // Read the state o
f the PIR sensor

    if gasLevel > gasThreshold:      // If harmful gas is d
etected
        if motionDetected == HIGH:    // If motion is detec
ted
            turnOnMotor(dcMotorPin)   // Activate the DC mo
tor (fan)
        else:
            turnOffMotor(dcMotorPin)  // Turn off the motor
if no motion
    else:
        turnOffMotor(dcMotorPin)      // Turn off the motor
if no gas detected
```

```
function turnOnMotor(pin):
    write(pin, HIGH) // Set the pin to HIGH to turn on the
motor

function turnOffMotor(pin):
    write(pin, LOW) // Set the pin to LOW to turn off the m
otor
```

7. Experiment: IR Sensor, Temperature Sensor, and Servo Motor

Tinkercard

Components:

- 1 x Arduino Uno board
- 1 x IR Sensor (used as a soil moisture sensor)
- 1 x Temperature Sensor (LM35)
- 1 x Servo Motor
- Breadboard
- Jumper wires
- Resistors (as needed)

Logic:

1. Initialize the IR sensor, temperature sensor, and servo motor.
2. Continuously monitor the readings from the IR sensor and the temperature sensor.
3. If the IR sensor detects low moisture levels (indicating the need for watering):
    - Check the temperature reading.
    - If the temperature is above a certain threshold, activate the servo motor to open the water valve.
4. If the moisture level is adequate or the temperature is below the threshold, keep the servo motor in the closed position.
5. Repeat the process to continuously monitor the sensors and control the servo motor accordingly.

Pseudo Code:

```
initialize irSensorPin = A0        // Pin for the IR sensor
(moisture sensor)
initialize temperatureSensorPin = A1 // Pin for the tempera
ture sensor
initialize servoPin = 9             // Pin for the servo mo
tor

initialize moistureThreshold = 300   // Threshold for moist
ure level
initialize temperatureThreshold = 25  // Threshold for temp
erature in Celsius

loop:
    moistureLevel = read(irSensorPin) // Read the moisture
level from the IR sensor
    temperature = readTemperature(temperatureSensorPin) //
Read the temperature

    if moistureLevel < moistureThreshold: // If moisture is
low
        if temperature > temperatureThreshold: // If temper
ature is high
            openWaterValve(servoPin) // Activate the servo
motor to open the valve
        else:
            closeWaterValve(servoPin) // Keep the valve clo
sed if temperature is low
    else:
        closeWaterValve(servoPin) // Keep the valve closed
if moisture is adequate

function readTemperature(pin):
    // Read the temperature from the temperature sensor
    rawValue = analogRead(pin) // Read the analog value
    temperature = (rawValue * 5.0 * 100.0) / 1024 // Conver
t to Celsius (example conversion)
    return temperature
```

```
function openWaterValve(pin):
    write(pin, 90) // Move the servo to open position (90 d
egrees)

function closeWaterValve(pin):
    write(pin, 0) // Move the servo to closed position (0 d
egrees)
```

8. Experiment: PIR Motor Sensor, Piezo Sensor, and Photo-resistor Sensor

Tinkercard

Components:

- 1 x Arduino Uno board
- 1 x PIR Sensor
- 1 x Piezo Sensor (Buzzer)
- 1 x Photo-resistor Sensor (LDR)
- Breadboard
- Jumper wires
- Resistors (as needed)

Logic:

1. Initialize the PIR sensor, piezo sensor, and photo-resistor sensor.
2. Continuously monitor the readings from the PIR sensor and the photo-resistor sensor.
3. If the PIR sensor detects motion:
    - Check the reading from the photo-resistor sensor.
    - If the photo-resistor detects a sudden change in light (indicating a potential intruder or alert condition), activate the piezo sensor to sound an alarm.
4. If no motion is detected or the light level is stable, keep the piezo sensor off.
5. Repeat the process to continuously monitor the sensors and control the piezo sensor accordingly.

Pseudo Code:

```
initialize pirSensorPin = 2          // Pin for the PIR sen
sor
initialize piezoSensorPin = 8        // Pin for the piezo s
ensor (buzzer)
initialize photoResistorPin = A0     // Pin for the photo-r
esistor sensor (LDR)

initialize lightChangeThreshold = 100 // Threshold for dete
cting light change

loop:
    motionDetected = read(pirSensorPin) // Read the state o
f the PIR sensor
    lightLevel = read(photoResistorPin)  // Read the light
level from the photo-resistor

    if motionDetected == HIGH:            // If motion is d
etected
        if lightLevel < lightChangeThreshold: // If light l
evel indicates a change
            activateAlarm(piezoSensorPin) // Activate the p
iezo sensor (sound alarm)
        else:
            deactivateAlarm(piezoSensorPin) // Deactivate t
he alarm if light is stable
    else:
        deactivateAlarm(piezoSensorPin) // Deactivate the a
larm if no motion is detected

function activateAlarm(pin):
    write(pin, HIGH) // Set the pin to HIGH to turn on the
piezo sensor (alarm)

function deactivateAlarm(pin):
    write(pin, LOW) // Set the pin to LOW to turn off the p
iezo sensor (alarm)
```

9. Resume

Report on Resume Screening Experiment

This report outlines the steps taken in an experiment to analyze resumes using Python. The process includes importing libraries, loading the dataset, processing the data, preparing features, performing natural language processing (NLP) tasks, generating a word cloud, handling stopwords, and training and testing a model.

Step 1: Importing Libraries

The first step involves importing necessary libraries for data manipulation, visualization, and natural language processing.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords
from wordcloud import WordCloud
import re
```

Step 2: Loading the Dataset

The next step is to load the resume dataset from a CSV file into a Pandas DataFrame. This allows for easy manipulation and analysis of the data.

```
from google.colab import files
uploaded = files.upload()
resumeDataSet = pd.read_csv('resume.csv', encoding='ISO-885
9-1')
```

Step 3: Initial Data Exploration

Exploration of the dataset helps in understanding its structure and the categories of resumes present.

```
print("Displaying the distinct categories of resume -")
print(resumeDataSet['Category'].unique())
```

Step 4: Data Cleaning and Processing

A function is defined to clean the resume text by removing URLs, hashtags, mentions, and punctuation. This step is crucial for preparing the text for analysis.

```
def cleanResume(resumeText):
    resumeText = re.sub('http\\S+\\s*', ' ', resumeText)  # remove URLs
    resumeText = re.sub('RT|cc', ' ', resumeText)  # remove RT and cc
    resumeText = re.sub('#\\S+', '', resumeText)  # remove hashtags
    resumeText = re.sub('@\\S+', ' ', resumeText)  # remove mentions
    resumeText = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-./:;<=>?@[$^_`{|}~"""), ' ', resumeText)  # remove punctuations
    resumeText = re.sub(r'[^\\x00-\\x7f]', r' ', resumeText)  # remove non-ASCII characters
    resumeText = re.sub('\\s+', ' ', resumeText)  # remove extra whitespace
    return resumeText

resumeDataSet['cleaned_resume'] = resumeDataSet.Resume.apply(lambda x: cleanResume(x))
```

Step 5: Feature Extraction

The TfidfVectorizer is used to convert the cleaned resume text into numerical features suitable for modeling.

```
from sklearn.feature_extraction.text import TfidfVectorizer
word_vectorizer = TfidfVectorizer(sublinear_tf=True, stop_words='english', max_features=1500)
```

```
word_vectorizer.fit(resumeDataSet['cleaned_resume'].values)
WordFeatures = word_vectorizer.transform(resumeDataSet['cle
aned_resume'].values)
```

Step 6: Generating a Word Cloud

A word cloud is created to visualize the most common words in the resumes. This helps in understanding key skills and terms frequently mentioned.

```
cleanedSentences = " ".join(resumeDataSet['cleaned_resum
e'].values)
wc = WordCloud(width=800, height=400, background_color='whi
te').generate(cleanedSentences)
plt.figure(figsize=(15, 8))
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Step 7: Handling Stopwords

A list of stopwords is downloaded and used to filter out common words that do not add significant meaning to the analysis.

```
nltk.download('stopwords')
oneSetOfStopWords = set(stopwords.words('english') + ['``',
"''"])
```

Step 8: Model Training and Testing

The dataset is split into training and testing sets, and a K-Nearest Neighbors classifier is trained on the training data.

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.multiclass import OneVsRestClassifier

requiredTarget = resumeDataSet['Category'].values
X_train, X_test, y_train, y_test = train_test_split(WordFea
```

```
tures, requiredTarget, random_state=0, test_size=0.2)

clf = OneVsRestClassifier(KNeighborsClassifier())
clf.fit(X_train, y_train)
```

Step 9: Evaluating the Model

The model is tested on the test set, and accuracy and classification reports are printed to evaluate performance.

```
from sklearn import metrics
prediction = clf.predict(X_test)
print('Accuracy of KNeighbors Classifier on training set:
{:.2f}'.format(clf.score(X_train, y_train)))
print('Accuracy of KNeighbors Classifier on test set: {:.2
f}'.format(clf.score(X_test, y_test)))
print("\\n Classification report for classifier %s:\\n%s
\\n" % (clf, metrics.classification_report(y_test, predicti
on)))
```

Conclusion

The experiment successfully loaded, processed, and analyzed resume data. The text was cleaned, features were extracted, a word cloud was generated, and a machine learning model was trained to categorize resumes. The results indicated high accuracy, demonstrating the model's effectiveness in classifying resumes based on their content.