Write a program to implement a simple form of a recurrent neural network. **a**. E.g. (4-to-1 RNN) to show that the quantity of rain on a certain day also depends on the values of the previous day

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Define sequence of 50 days of rain data
rain_data = np.array([2.3, 1.5, 3.1, 2.0, 2.5, 1.7, 2.9, 3.5, 3.0, 2.1,
                      2.5, 2.2, 2.8, 3.2, 1.8, 2.7, 1.9, 3.1, 3.3, 2.0,
                      2.5, 2.2, 2.4, 3.0, 2.1, 2.5, 3.2, 3.1, 1.9, 2.7,
                      2.2, 2.8, 3.1, 2.0, 2.5, 1.7, 2.9, 3.5, 3.0, 2.1,
                      2.5, 2.2, 2.8, 3.2, 1.8, 2.7, 1.9, 3.1, 3.3, 2.0])

# Create input and output sequences for training
time_steps = 4
x_train = np.array([rain_data[i:i+time_steps] for i in range(len(rain_data)-time_steps)])
y_train = rain_data[time_steps:]

# Define RNN model
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(8, input_shape=(time_steps, 1)),
    tf.keras.layers.Dense(1)
])

# Compile and train model
model.compile(optimizer="adam", loss="mse")
history = model.fit(x_train.reshape(-1, time_steps, 1), y_train, epochs=100)

# Plot loss over time
plt.plot(history.history["loss"], "bo", label="Training loss")
plt.title("Training loss")
plt.legend()
plt.show()

# Test model on new sequence
test_sequence = np.array([2.5, 2.2, 2.8, 3.2])
x_test = np.array([test_sequence])
y_test = model.predict(x_test.reshape(-1, time_steps, 1))

# Print input, output, and prediction
print("Previous days' rain data:", test_sequence)
print("Expected rain amount for next day:", y_test[0][0])
prediction = model.predict(np.array([test_sequence]).reshape(1, time_steps, 1))
print("Prediction:", prediction[0][0])
```

```
2/2 [==============================] - 0s 10ms/step - loss: 0.2721
Epoch 95/100
2/2 [==============================] - 0s 10ms/step - loss: 0.2720
Epoch 96/100
2/2 [==============================] - 0s 9ms/step - loss: 0.2719
```

b. LSTM for sentiment analysis on datasets like UMICH SI650 for similar

```
Epoch 98/100
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Load data
data = pd.read_csv("training.txt", delimiter="\t", names=["label", "text"])
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data["text"],data["label"], test_size=0.2, random_state=42)

# Tokenize words
tokenizer = Tokenizer(num_words=5000, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)

# Convert words to sequences
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

# Pad sequences to have same length
max_length = 100
X_train_pad = pad_sequences(X_train_seq, maxlen=max_length, padding="post",truncating="post")
X_test_pad = pad_sequences(X_test_seq, maxlen=max_length, padding="post",truncating="post")

# Build LSTM model
model = tf.keras.models.Sequential([
tf.keras.layers.Embedding(input_dim=5000, output_dim=32,input_length=max_length),
tf.keras.layers.LSTM(units=64, dropout=0.2, recurrent_dropout=0.2),
tf.keras.layers.Dense(1, activation="sigmoid")
])

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

# Train model
history = model.fit(X_train_pad, y_train, epochs=10, batch_size=32,validation_split=0.1)

# Evaluate model on test data
loss, accuracy = model.evaluate(X_test_pad, y_test)
print("Test loss:", loss)
print("Test accuracy:", accuracy)

# Plot training and validation accuracy over time
plt.plot(history.history["accuracy"], label="Training accuracy")
plt.plot(history.history["val_accuracy"], label="Validation accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

# Make predictions on test data
predictions = model.predict(X_test_pad)

# Print input, output, and prediction for random example
index = np.random.randint(0, len(X_test_pad))
text = tokenizer.sequences_to_texts([X_test_pad[index]])[0]
label = y_test.values[index]
prediction = predictions[index][0]
print("Text:", text)
print("Actual label:", label)
print("Predicted label:", round(prediction))
```