```
#  ----- Exercise 2 - Decode CAN data -----
# Version: 2022
# Course: TME 192 Active Safety
#         Chalmers
# Author: Rahul Rajendra Pai (rahul.pai@chalmers.se)
#         Alberto Morando (morando@chalmers.se)
#         Alexander Rasch (arasch@chalmers.se)
#         Marco Dozza (dozza@chalmers.se)
#
# Group: Group 5

# Group member: [Yahui Wu]
# Group member: [Tianshuo Xiao]
# Group member: [Nishanth Suresh]

import pandas as pd
import json
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import interp1d

# Auxiliary functions
# Function to translate hex to binary
def hex2bin(hex_string):
    return bin(int(hex_string, 16))[2:].zfill(8)

# Import the data
with open('CANdata.json') as json_file:
    data = json.load(json_file)

CAN_data = pd.DataFrame.from_dict(data)

# Add variable names. This would be helpful for analysing the data
CAN_data.rename(columns={'var1':'timestamp', 'var3':'identifier'}, inplace=True)

# Decode the signals
# Select the frames based on the identifier.
# You could use string comparison on the `Identifier` column

# Find array of index of the vehicle speed signal to decode
identifiers = CAN_data['identifier']
speed_index = []
for index, identifier_value in enumerate(identifiers):
    if identifier_value == '401':
        speed_index.append(index)


# Extract the bytes of interest into an array
speed_hex = np.array(CAN_data.loc[speed_index, 'var14'])


# Convert the hex to binary
speed_bin = []
for i in speed_hex:
    speed_binary = hex2bin(i)
    speed_bin.append(speed_binary)


# Convert the binary to decimal. Apply the 'factor' and `offset`
```

```
speed_a = []

for binary_value in speed_bin:
    decimal_value = int(binary_value, 2)
    speed_a.append(decimal_value)


speed = speed_a ## no 'factor' and `offset`


# Find array of index of the accelerator pedal position signal to decode

accelerator_index = []

for index_acc, identifier_acc in enumerate(identifiers):
    if identifier_acc == '1CC':
        accelerator_index.append(index_acc)


# Extract the bytes of interest into an array
accelerator_hex = np.array(CAN_data.loc[accelerator_index , ['var11','var12']])

#  Convert the hex to binary
var11_hex = accelerator_hex[:,0]
var12_hex = accelerator_hex[:,1]

var11_binarry = []
var12_binarry = []

for hex_value in var11_hex:
    binary_value = hex2bin(hex_value)
    var11_binarry.append(binary_value)

for hex_value in var12_hex:
    binary_value = hex2bin(hex_value)
    var12_binarry.append(binary_value)

var_com = []

for var11, var12 in zip(var11_binarry, var12_binarry):
    var11_last_two = var11[-2:]
    merged_var = var11_last_two + var12
    var_com.append(merged_var)



accelerator_bin = var_com

# Convert the binary to decimal. Apply the 'factor' and `offset`
accelerator = []
for j in accelerator_bin:
    acc_decimal_value = int(j, 2)
    accelerator.append(acc_decimal_value*0.098)




# Plot the signals with respect to time. Include labels and legend
speed_time = np.array(CAN_data.loc[speed_index, 'timestamp'])
```

```
accelerator_time = np.array(CAN_data.loc[accelerator_index, 'timestamp'])
plt.figure(figsize=(10, 6))

plt.plot(speed_time, speed, label = 'Speed')
plt.plot(accelerator_time, accelerator, label = 'Accelerator pedal position')
plt.xlabel('Time[ms]')
plt.ylabel('Speed [km/h]/Accelerator pedal position [%]')
plt.legend(loc = 'upper right')

plt.show()

# Downsample the signals
# Remove the delay between the two signals.
# That is, make sure both signals starts from zero.
speed_time_from_0 = speed_time
accelerator_time_from_0 = accelerator_time - 0.0002

# Create a master clock, at 1Hz, common to both signals
sync_time = np.linspace(0, 39, 40)


# Downsample by interpolation
speed_sync = []
f = interp1d(speed_time, speed, kind = 'linear')
speed_sync = f(sync_time)


accelerator_sync = []
f = interp1d(accelerator_time_from_0, accelerator, kind = 'linear')
accelerator_sync = f(sync_time)

# Plot the signals with respect to time. Include labels and legend. Limit the axis.



plt.figure(figsize=(10, 6))
point_x =  [29,30,31,32,33,34,35]
plt.xlim(28, 36)
plt.ylim(15, 30)
plt.plot(sync_time, speed_sync, label = 'Speed synced', linestyle = '--')
plt.plot(speed_time, speed, label = 'Speed')
plt.plot(sync_time, accelerator_sync, label = 'Accelerator pedal position synced',
linestyle = '--')
plt.plot(accelerator_time, accelerator, label = 'Accelerator pedal position')
plt.xlabel('Time[ms]')
plt.ylabel('Speed [km/h]/Accelerator pedal position [%]')
plt.legend(loc = 'upper right')



plt.show()
# Issue a warning when the driver is about to exceed the speed limit
#  Be careful about the units!
SPEED_LIMIT = 50;       # [km/h]
TOLERANCE = 10;         # [km/h]
REACTION_TIME = 1.0;    # [s]

# Predicted speed
predicted_speed = []
```

```
acc_first = [0]
acc_diff = np.diff(speed_sync)
acc = np.concatenate((acc_first,acc_diff), axis = 0)
#acc = list(acc)
predicted_speed = speed_sync+acc

# Find time when predicted speed is greater than speed limit + tolerance

warning_time = int(np.where(predicted_speed>60)[0][0])

print('Warning issued at: ', warning_time, ' seconds')



## ----------------- Optional Grader tool to verify the output ----------------- ##
# If dill is missing, install it using `pip install dill`
import dill
check_output = dill.load(open('check_output.pkl', 'rb'))
check_output(speed_index, speed_hex, speed_bin, speed, accelerator_index,
accelerator_hex, accelerator_bin, accelerator, speed_time, accelerator_time,
speed_time_from_0, accelerator_time_from_0, sync_time, speed_sync,
accelerator_sync, predicted_speed, warning_time)
```