Module Name: Basic Programming – BP411
Assessment Type: Group Activity 3
Total Mark Allocation: 50 Marks
Total Hours: 10 Hours

1. **No material may be copied from original sources, even if referenced correctly, unless it is a direct quote indicated with quotation marks.**
2. **No more than 10% of the assessment may consist of direct quotes.**
3. **No assessment with a similarity index of more than 25%, even if the sources are referenced correctly, will be accepted.**
4. **Make a copy of your assessment before handing it in.**
5. **All assessment must be typed unless otherwise specified.**
6. **All work must be adequately and correctly referenced.**
7. **All questions should be answered using Python.**

CRYPTO – WATCH DOG

In this group activity, you are going to create an app that keeps an eye on the Bitcoin price – and warns the user if the price goes below or above certain levels.

The app will contact the coindesk exchange to get the latest price – and then it will compare the Bitcoin price to the price-levels set by the user.

To enable the app to send the request across the internet, you have to install pip and the requests library (if not installed already).

Starting with Python 3.4, pip is included with the rest of Python. If you're using an earlier version of Python 3 and don't have pip, you can get it from http://www.pip-installer.org.

Once pip is installed, install the requests library: `pip install requests`

Use the following code to send a request to the coindesk exchange; and to retrieve the response:

```
import requests
bitcoin_api_url = 'https://api.coindesk.com/v1/bpi/currentprice.json'
response = requests.get(bitcoin_api_url)
response_json = response.json()
```

The above code requests coindesk.com to return the current Bitcoin price in json format.
If you print response_json, you will see that the received json object represents a dictionary that consists of four main entries being: time, disclaimer, chartName and bpi.

The bpi entry contains three sub-entries, which contains the Bitcoin pricing data in USD, GBP and EUR respectively. These entries are further sub-divided into code, symbol, rate, description and rate_float entries.

Example json


```
{
  'time': {
    'updated': 'May 22, 2020 08:58:00 UTC',
    'updatedISO': '2020-05-22T08:58:00+00:00',
    'updateduk': '...'
  },
```

```
  'disclaimer': '...',
  'chartName': 'Bitcoin',
  'bpi': {
    'USD': {
      'code': 'USD',
      'symbol': '&#36;',
      'rate': '9,083.4400',
      'description': 'United States Dollar',
      'rate_float': 9083.44},
    'GBP': {...},
    'EUR': {...}
  }
}
```

The following code will print the Bitcoin price in US Dollars (as a floating point). Use this value to store the prices in your code:

```
print(response_json["bpi"]["USD"]["rate_float"])
```

The following code will print the date and time that the data was retrieved:

```
print(response_json["time"]["updated"])
```

Note that the time is given in UTC (GMT+00).

APP INSTRUCTIONS:

1. Allow the user to enter as many price levels (prices) as he/she requires.

2. Store these price levels in a list called levelsList and in a file on disk called levelsFile.

3. Every time the app starts, it must read the price levels from levelsFile (if existing yet) and store them in levelsList.
It must then ask the user if he/she wants to add or delete more prices on the list.
The flow of this section of your app should thus be:

a) App starts
b) Read levelsFile from disk.
- If levelsFile does not exist yet, create an empty levelsList.
- If levelsFile does exist, load the prices into a new levelsList.
c) Display the levelsList's data to the user and ask if he/she wants to add/remove any items on the list.
d) Update the levelsList according to the user's inputs in (c).
e) Each time you make changes to levelsList, update the file on disk with the new values.
- If levelsFile does not exist yet, create it.
- If levelsFile does exist, replace all its data with the new data from the list.

Note: levelsList and levelsFile should store price values as floats. When displaying values to the user, display only two decimals.

4. After the user indicated that he/she does not want to make any further changes to the list, your app must start a timer function that checks the Bitcoin price every 5 seconds. Before each request to the exchange, save the Bitcoin price you received in the previous request as previousPrice. After retrieving the new Bitcoin price, save it as currentPrice.
Display the prices in the levelsList from highest to lowest - while also incorporating the previousPrice and CurrentPrice (see examples below).

a) The user's price levels must be displayed in black/white.
a) The previousPrice must always be displayed in blue.
c) If currentPrice is higher than previousPrice, then display currentPrice in green. If currentPrice is lower than previousPrice, then display currentPrice in red (you can choose appropriate background colours if you want). Tip: search how-to-print-colored-text-in-terminal-in-python.
d) If there is a price level between the previousPrice and currentPrice (the Bitcoin price crossed the price level while moving up or down), sound a beep or any other form of alarm.

The following examples assume that the user entered price levels of (9170.00, 9175.00, 9180.00, 9185.00)

Example of the display if the price went up	Example of the display if the price went down

Price Level:	9185.00		Price Level:	9185.00
Current Price:	9182.50		Price Level:	9180.00
Previous price:	9181.00		Previous price:	9177.00
Price Level:	9180.00		Price Level:	9175.00 (sound the alarm!)
Price Level:	9175.00		Current Price:	9174.50
Price Level:	9170.00		Price Level:	9170.00

Tip: If your alarm does not want to sound, it is probably because it needs to run on a separate thread than the UI. Build your 5 second timer function using threading.Timer

```
import threading
def myLoop():
   threading.Timer(5.0, myLoop).start()
   …
   soundTheAlarm()

myLoop()
```