SPRINT 6

9972 Dylan Groenewald

CTU training solutions
Core Web Development

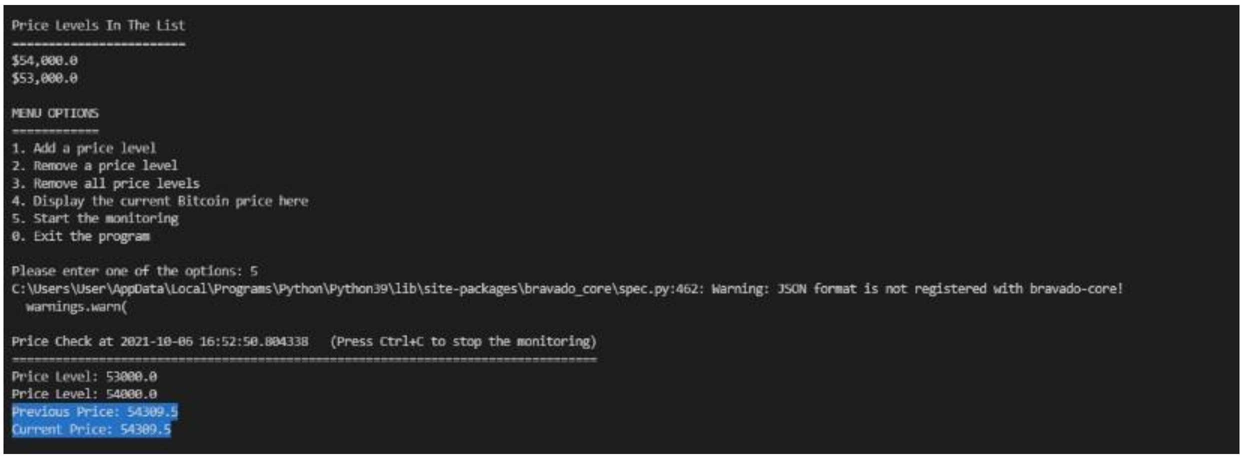
Contents

Code for Sprint 6 CryptoChecker.py

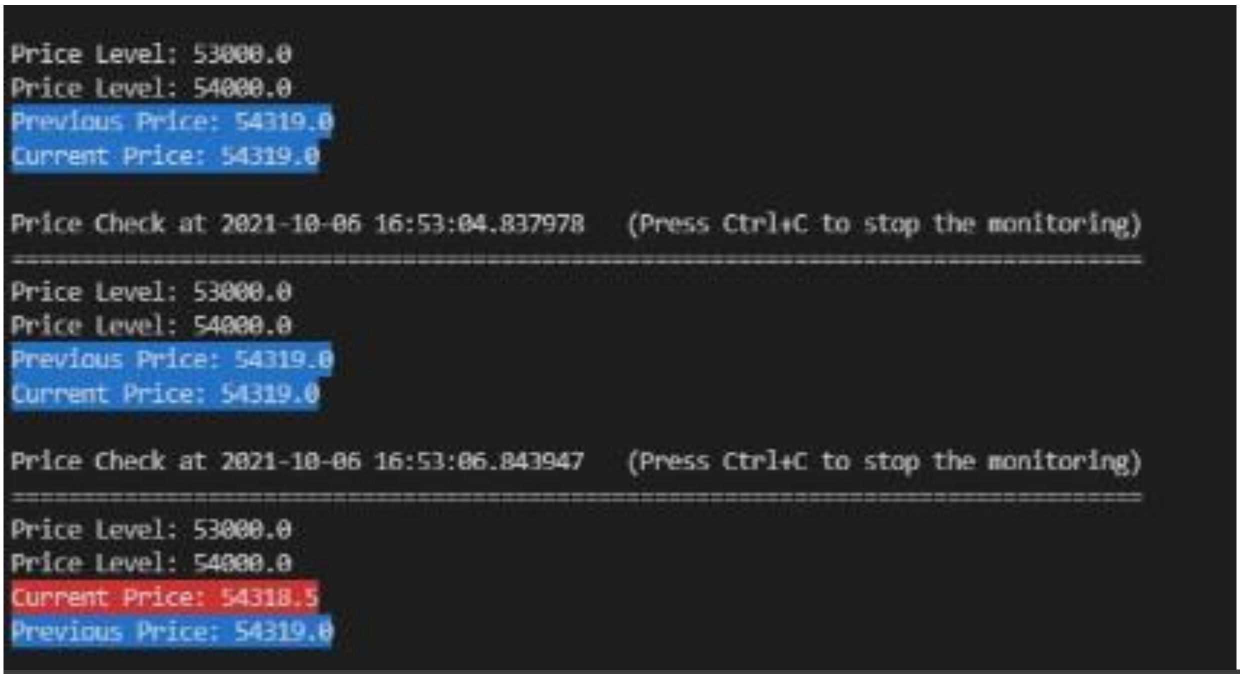
```python
import bitmex
import json
import threading
import winsound
from datetime import datetime
from colorama import init, Style, Fore, Back



class Tick():
    def __init__(self, time, price):
        self.time = time
        self.price = price


class PriceChecker():
    # Constructor
    def __init__(self):
        self.levelsList = []        # Call the @levelsList.setter method and pass it an empty list
        self.currentPrice  = 0.0    # Call the @currentPrice.setter method and pass it 0.0
        self.BitmexClient = bitmex.bitmex(test=False)
        self.previousPrice = 0.0    # Call the @previousPrice.setter method and pass it 0.0

    # Properties
    # A property is defined like a method, but you use it in your
    # code like a variable (no parentheses need to followed it when used in your code)
    # Refer: https://www.youtube.com/watch?v=jCzT9XFZ5bw
    # Refer BP411 slides: Week 2 - Chapter 10 - Slides about Encapsulation and properties
    @property
    def levelsList(self):
        return self.__levelsList    # Return the value of __levelsList
    @levelsList.setter
    def levelsList(self, newValue):
        self.__levelsList = newValue    # Set the value of __levelsList

    @property
    def currentPrice(self):
        return self.__currentPrice      # Return the value of __currentPrice
    @currentPrice.setter
    def currentPrice(self, newValue):
        self.__currentPrice = newValue  # Set the value of __currentPrice

    @property
    def previousPrice(self):
        return self.__previousPrice # Return the value of __previousPrice
    @previousPrice.setter
    def previousPrice(self, newValue):
        self.__previousPrice = newValue             # Set the value of __previousPrice

    # Class Methods
    # =============


    # Method: Sort and Display the levelsList
    def displayList(self):
        print(chr(27) + "[2J") # Clear the screen
        print("Price Levels In The List")
        print("========================")
        # Sort the list in reverse order
        self.levelsList.sort(reverse=True)
        # Print the items in the list (Based on the above sort, numbers should appear from large to
small.)
        for i in range(len(self.levelsList)):
            print('${:,.1f}'.format(self.levelsList[i]),)

    # Display the menu and get user input about what methods to execute next
    def displayMenu(self):
        min = 0
        max = 5
        errorMsg = "Please enter a valid option between " + str(min) + " and " + str(max)

        print("MENU OPTIONS")
        print("============")
        print("1. Add a price level")
        print("2. Remove a price level")
        print("3. Remove all price levels")
        if(self.currentPrice > 0):
            print("4. Display the current Bitcoin price here: " + f"${self.currentPrice:,}")
            print("5. Start the monitoring")
        else:
            print("4. Display the current Bitcoin price here")
            print("5. Start the monitoring")
            print("0. Exit the program")
        print(" ")

        # Get user input. Keep on requesting input until the user enters a valid number between min and
max
        selection = 99
        while selection < min or selection > max:
            try:
                selection = int(input("Please enter one of the options: "))
            except:
                print(errorMsg) # user did not enter a number
                continue # skip the following if statement
            if(selection < min or selection > max):
                print(errorMsg) # user entered a number outside the required range
        return selection # When this return is finally reached, selection will have a value between (and
including) min and max

    # Method: Append a new price level to the levelsList
    def addLevel(self):
        try:
            # Let the user enter a new float value and append it to the list
            self.levelsList.append(float(input("Enter a price lever to add:")))

        except:
            # Print and error message if the user entered invalid input
            print('Enter a valid price')


    # Method: Remove an existing price level from the levelsList
    def removeLevel(self):
        try:
            # Let the user enter a new float value. If found in the list, remove it from the list
            remove_value = float(input("Enter a price to remove from the list: "))
            if remove_value in self.levelsList: # check if number is in the list
                self.levelsList.remove(remove_value)
            else:
                print('Number not found in the list') # user
        except:
            # Print and error message if the user entered invalid input
            print("Enter a valid price")
    # Method: Set levelsList to an empty list
    def removeAllLevels(self):
        # Set levelsList to an empty list
        self.levelsList.clear()

    #Method load levelslist using the data in levelsfile
    def readLevelsFromFile(self):
        try:
            # Set levelsList to an empty list
            self.levelsList.clear()
            # Open the file
            with open('Levels_file.txt', "r") as Levels_file:
            # Use the loop to read through the file line by line
                for sline in Levels_file:
                # If the last two characters in the line is "\n", remove them
                    if (sline[-1] == '\n') and (sline[-2] == '\n'):
                        sline.remove(sline[-1])
                        sline.remove(sline[-2])
                # Append the line to levelsList
                    self.levelsList.append(float(sline))
            # Close the file
            Levels_file.close()
        except:
            return

    # Method: Write levelsList to levelsFile (override the existing file)
    def writeLevelsToFile(self):
        # Open the file in a way that will override the existing file (if it already exists)
        with open('Levels_file.txt', "w") as Levels_file:
            # Use a loop to iterate over levelsList item by item
            for level in self.levelsList:
                # Convert everything in the item to a string and then add \n to it - before writing it to
 the file
                newLevel = str(str(level) + '\n')
                Levels_file.write(newLevel)
        # Close the file
        Levels_file.close()


    # Function: Display the Bitcoin price in the menu item – to assist the user when setting price levels
    def updateMenuPrice(self):
        # Get the latest Bitcoin info (as a Tick object) from getBitMexPrice(). Name it tickObj.
        tickObj = self.getBitMexPrice()
        # Update the currentPrice property with the Bitcoin price in tickObj.
        self.currentPrice = tickObj.price

    # Function: Call the Bitmex Exchange
    def getBitMexPrice(self):
        # Send a request to the exchange for Bitcoin's data in $USD ('XBTUSD').
        # The json response is converted into a tuple which we name responseTuple.
        responseTuple = self.BitmexClient.Instrument.Instrument_get(filter=json.dumps({'symbol':
'XBTUSD'})).result()
        # The tuple consists of the Bitcoin information (in the form of a dictionary with key>value
pairs) plus
        # some additional meta data received from the exchange.
        # Extract only the dictionary (Bitcoin information) from the tuple.
        responseDictionary = responseTuple[0:1][0][0]
        # Create a Tick object and set its variables to the timestamp and lastPrice data from the
dictionary.
        return Tick(responseDictionary["timestamp"], responseDictionary['lastPrice'])
        # Once this method has been called, it uses a Timer to execute every 2 seconds

        # Once this method has been called, it uses a Timer to execute every 2 seconds
    def monitorLevels(self):
        # Create timer to call this method every 2 seconds
        threading.Timer(2.0, self.monitorLevels).start()

        # Since we will obtain the latest current price from the exchange,
        # store the existing value of currentPrice in previousPrice
        self.previousPrice = self.currentPrice

        # Similar to updateMenuPrice(), call the getBitMexPrice() method to get
        # a Ticker object containing the latest Bitcoin information. Then store
        # the Bitcoin price in currentPrice

        tickObj = self.getBitMexPrice()
        self.currentPrice = tickObj.price

        # During the first loop of this method, previousPrice will still be 0 here,
        # because it was set to currentPrice above, which also was 0 before we updated
        # it above via getBitMexPrice().
        # So, when we reach this point during the first loop, previousPrice will be 0
        # while currentPrice would have just been updated via getBitMexPrice().
        # We don't want to create the impression that the price shot up from 0 to
        # currentPrice.
        # Therefore, if previousPrice == 0.0, it must be set equal to currentPrice here.
        if self.previousPrice == 0.0:
            self.previousPrice = self.currentPrice

        # Print the current date and time plus instructions for stopping the app while this
        # method is looping.
        print('')
        print('Price Check at ' + str(datetime.now()) + '   (Press Ctrl+C to stop the monitoring)')
        print('================================================================================')

        # Each time this method executes, we want to print the items in levelsList together with
previousPrice
        # and currentPrice in the right order. However, as we loop through levelsList, how do we know
where to
        # insert previousPrice and currentPrice - especially if currentPrice crossed one or two of our
price
        # levels?
        # We could try to use an elaborate set of IF-statements (I dare you to try this), but a much
easier
        # way is to simply add previousPrice and currentPrice to the list and then sort the list.
        #
        # However, we cannot simply use levelsList for this purpose, because it only stores values, while
we
        # also want to print labeling text with these values - such as 'Price Level', 'Current Price' and
        # 'Previous Price'.
        # Therefore, we need to create a temporary list - called displayList - used for displaying
purposes only.
        # This new list must consist of sub-lists. Each sub-List will contain two items.
        # The first item will be the label we want to print - consisting of the labeling text and the
price.
        # The second item consists of the price only.
        # We will use the second item to sort the list - since it makes no sense to sort the list based
on
        # the label (the first item).
        #
        # Example of displayList (containing sub-lists) after it was sorted:
        #
        #       [
        #           ['Price Level:    9700.00',    9700.00],
        #           ['Price Level:    9690.00',    9690.00],
        #           ['Current Price:  9689.08',    9689.08],
        #           ['Previous Price: 9688.69',    9688.69],
        #           ['Price Level:    9680.00',    9680.00],
        #       ]

        # Create displayList
        displayList = []

        # Loop through the prices in levelsList.
        # During each loop:
        # - Create a variable called priceLevelLabel consisting of the text 'Price Level:    ' followed
        #   by the price.
        # - Add priceLevelLabel and the price as two separate items to a new list (the sub-List).
        # - Append the sub-List to displayList.
        for price in self.levelsList:
            priceLevelLabel = "Price Level: " + str(price)
            subList = []
            subList.append(priceLevelLabel)
            subList.append(price)
            displayList.append(subList)

         # Create a variable called previousPriceLabel consisting of the text 'Previous Price: ' followed
        # by previousPrice.
        # Format the background colour of previousPriceLabel to be blue. Refer to the following site:
        # https://stackoverflow.com/questions/287871/how-to-print-colored-text-in-terminal-in-python
        # Follow the above site to add special text characters to the label, which the console will
interpret
        # as background colour settings. If the above site's code does not work for your console (I am
using
        # Visual Studio Code), research a different way for setting the background colour.
        # Add previousPriceLabel and previousPrice as two separate items to a new list (the sub-List).
        # Append the sub-List to displayList.
        subList= []
        price = self.previousPrice
        previousPriceLabel = (Back.BLUE + 'Previous Price: ' + str(price) + Style.RESET_ALL + Back.RESET)
        label = previousPriceLabel
        subList.append(label)
        subList.append(price)
        displayList.append(subList)

        # Create a variable called currentPriceLabel consisting of the text 'Current Price:  ' followed
        # by currentPrice.
        # Format the background colour of currentPriceLabel as follows:
        # - If currentPrice > previousPrice: set currentPriceLabel background colour to green
        # - If currentPrice < previousPrice: set currentPriceLabel background colour to red
        # - If currentPrice == previousPrice: set currentPriceLabel background colour to blue
        # Add currentPriceLabel and currentPrice as two separate items to a new list (the sub-List).
        # Append the sub-List to displayList.
        subList= []
        price = self.currentPrice
        if self.currentPrice > self.previousPrice:
            currentPriceLabel = (Back.GREEN + 'Current Price: ' + str(price)+ Style.RESET_ALL +
Back.RESET)

        if self.currentPrice < self.previousPrice:
            currentPriceLabel = (Back.RED + 'Current Price: ' + str(price)+ Style.RESET_ALL + Back.RESET)

        if self.currentPrice == self.previousPrice:
            currentPriceLabel = (Back.BLUE + 'Current Price: ' + str(price)+ Style.RESET_ALL +
Back.RESET)

        label = currentPriceLabel
        subList.append(label)
        subList.append(price)
        displayList.append(subList)

        # Sort displayList using the SECOND item (price) in its sub-lists
```

```python
        displayList = sorted(displayList, reverse=False ,key= lambda x: x[1])

        # For each sub-List in displayList, print only the label (first item) in the sub-List
        for subList in displayList:
            print(subList[0])

        # Loop through displayList
        for i in range(0,len(displayList)):
            # Test if the first item in the current sub-list contains the text "Price Level"
            # Tip: Remember that each sub-list is a list within a list (displayList). So you have
            #       to access its items via displayList followed by TWO indexes.
             if ('Price level' in displayList[i][0]):
                # Extract the second item from the current sub-list into a variable called priceLevel
                pricelevel = subList[i][1]
                # Test if priceLevel is between previousPrice and currentPrice OR
                #         priceLevel == previousPrice OR
                #         priceLevel == currentPrice
                if(
                    (pricelevel == self.currentPrice) or (pricelevel == self.previousPrice)
                    or (pricelevel in range(self.currentPrice, self.previousPrice))
                ):
                    # Sound the alarm. Pass in the frequency and duration.
                    if self.currentPrice > self.previousPrice:
                        frequency = 800
                        duration = 700
                    else:
                        frequency = 400
                        duration = 700
                    winsound.Beep(frequency, duration)

                    # Print the text 'Alarm' with a green background colour, so that the user
                    # can go back into the historical data to see what prices raised the alarm.
                    print(Back.GREEN + "Alarm" + Style.RESET_ALL + Back.RESET)



# *************************************************************************************************
#                                           Main Code Section
# *************************************************************************************************

# Create an object based on the PriceChecker class
checkerObj = PriceChecker()

# Load levelsList from the records in levelsFile
checkerObj.readLevelsFromFile()

# Display the levelsList and Menu; and then get user input for what actions to take
userInput = 99
while userInput != 0:
    checkerObj.displayList()
    userInput = checkerObj.displayMenu()
    if(userInput == 1):
        checkerObj.addLevel()
        checkerObj.writeLevelsToFile() # Write levelsList to LevelsFile
    elif(userInput == 2):
        checkerObj.removeLevel()
        checkerObj.writeLevelsToFile() # Write levelsList to LevelsFile
    elif(userInput == 3):
        checkerObj.removeAllLevels()
        checkerObj.writeLevelsToFile() # Write levelsList to LevelsFile
    elif(userInput == 4):
        checkerObj.updateMenuPrice()
    elif(userInput == 5):
        userInput = 0 # prevent the app from continuing if the user pressed Ctrl+C to stop it
        checkerObj.monitorLevels()
```

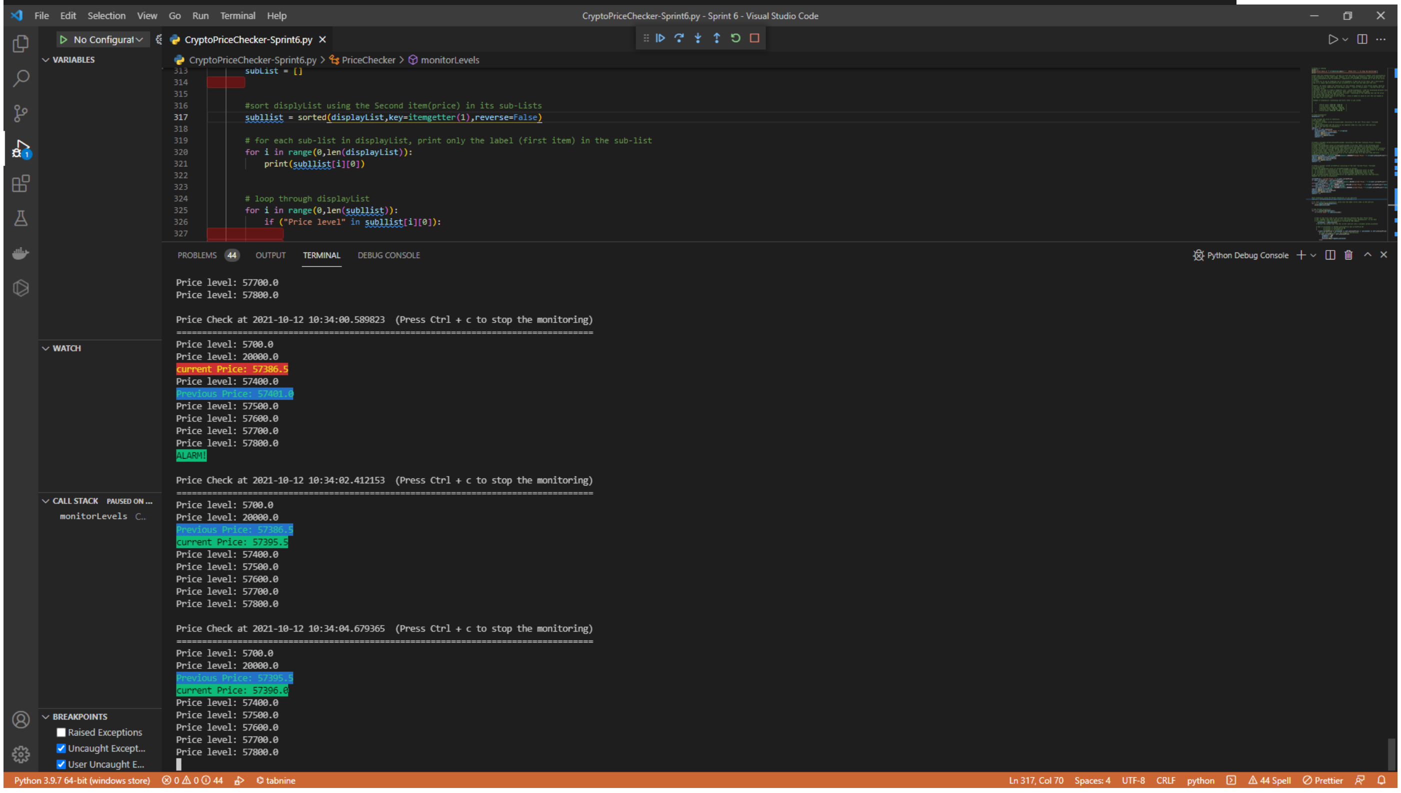
Output of Code



```
Price Levels In The List
========================
$54,000.0
$53,000.0

MENU OPTIONS
============
1. Add a price level
2. Remove a price level
3. Remove all price levels
4. Display the current Bitcoin price here
5. Start the monitoring
0. Exit the program

Please enter one of the options: 5
C:\Users\User\AppData\Local\Programs\Python\Python39\lib\site-packages\bravado_core\spec.py:462: Warning: JSON format is not registered with bravado-core!
  warnings.warn(

Price Check at 2021-10-06 16:52:50.804338   (Press Ctrl+C to stop the monitoring)
=================================================================================
Price Level: 53000.0
Price Level: 54000.0
Previous Price: 54309.5
Current Price: 54309.5
```



```
Price Level: 53000.0
Price Level: 54000.0
Previous Price: 54319.0
Current Price: 54319.0

Price Check at 2021-10-06 16:53:04.837978   (Press Ctrl+C to stop the monitoring)
=================================================================================
Price Level: 53000.0
Price Level: 54000.0
Previous Price: 54319.0
Current Price: 54319.0

Price Check at 2021-10-06 16:53:06.843947   (Press Ctrl+C to stop the monitoring)
=================================================================================
Price Level: 53000.0
Price Level: 54000.0
Current Price: 54318.5
Previous Price: 54319.0
```



```python
        subList = []


        #sort displyList using the Second item(price) in its sub-Lists
        subllist = sorted(displayList,key=itemgetter(1),reverse=False)

        # for each sub-list in displayList, print only the label (first item) in the sub-list
        for i in range(0,len(displayList)):
            print(subllist[i][0])


        # loop through displayList
        for i in range(0,len(subllist)):
            if ("Price level" in subllist[i][0]):
```

```
Price level: 57700.0
Price level: 57800.0

Price Check at 2021-10-12 10:34:00.589823  (Press Ctrl + c to stop the monitoring)
==================================================================================
Price level: 5700.0
Price level: 20000.0
current Price: 57386.5
Price level: 57400.0
Previous Price: 57401.0
Price level: 57500.0
Price level: 57600.0
Price level: 57700.0
Price level: 57800.0
ALARM!

Price Check at 2021-10-12 10:34:02.412153  (Press Ctrl + c to stop the monitoring)
==================================================================================
Price level: 5700.0
Price level: 20000.0
Previous Price: 57386.5
current Price: 57395.5
Price level: 57400.0
Price level: 57500.0
Price level: 57600.0
Price level: 57700.0
Price level: 57800.0

Price Check at 2021-10-12 10:34:04.679365  (Press Ctrl + c to stop the monitoring)
==================================================================================
Price level: 5700.0
Price level: 20000.0
Previous Price: 57395.5
current Price: 57396.0
Price level: 57400.0
Price level: 57500.0
Price level: 57600.0
Price level: 57700.0
Price level: 57800.0
```

The above code shows how the values in the list are saved and showed once the program runs again. All the values are stored in the text file and then put into the list. It also displays all the bitmex prices and all price levels.