



암호화폐 트레이딩 교육

Day 2 (1/2)

상승장 알리미

Kangwuk Heo
calvin.heo@gmail.com

Contents

상승장 알리미 3

SECTION 1: 가상화폐 거래소를 통한 거래 정보 가져오기4

SECTION 2: UI 와 연계 작업.....16

SECTION 3: 상승장 알리미 PERFORMANCE 개선20

Overview

암호화폐 트레이딩 교육 과정에서, 실습을 통해서, 트레이딩이 무엇인지, 트레이딩을 어떤 방식으로 구성하고 진행하는지를 배울 수 있습니다.

상승장 알리미

Section 1 은 가상화폐 거래소와의 연계를 통해 거래되고 있는 코인 정보 및 코인의 호가정보를 기반으로 상승장 알리미 프로그램을 순차적으로 개발해 보는 과정이 포함되어 있습니다.

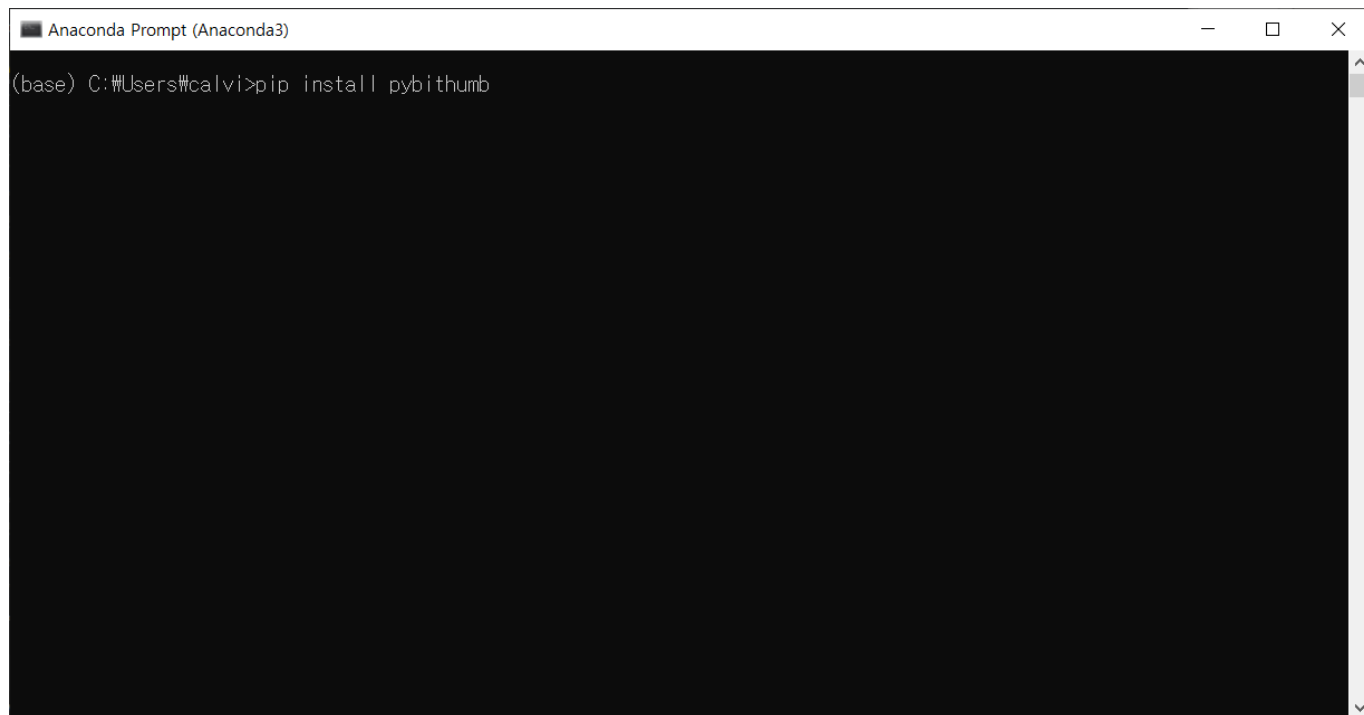
Section 1: 가상화폐 거래소를 통한 거래 정보 가져오기

가상화폐별로 거래되는 가상화폐(코인)이 다르고, 가상화폐별 네이밍되어 있는 이름(Ticker) 정보가 다르므로 인해, 가상화폐 거래소에서 사용되는 Ticker 정보를 확인하여, 관련 가상화폐 정보를 가져오는 작업을 진행합니다.

Section 1 에서 연계되는 가상화폐 거래소는 빗썸으로, 빗썸에서 제공하는 파이썬 모듈을 설치합니다.

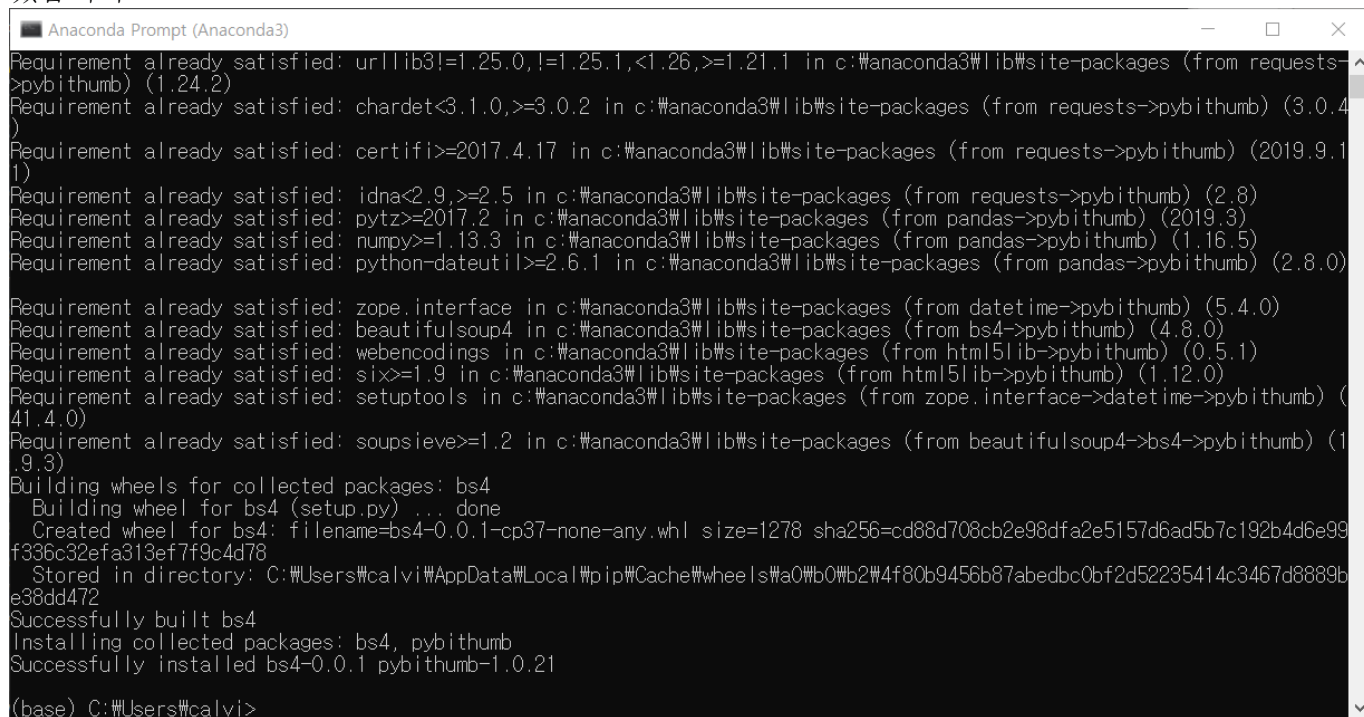
- __1. 아나콘다(Anaconda) 메뉴에서, Anaconda Prompt 아이콘을 클릭하여, 보여지는 Anaconda Prompt 창에서 빗썸 모듈을 설치합니다.

pip install pybithumb



```
Anaconda Prompt (Anaconda3)
(base) C:\Users\calvi>pip install pybithumb
```

명령어를 실행하였다면, 관련 모듈이 설치가 되고, 마지막에 **Successfully installed** 로그를 확인하실 수 있습니다.



```
Anaconda Prompt (Anaconda3)
Requirement already satisfied: urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 in c:\anaconda3\lib\site-packages (from requests->pybithumb) (1.24.2)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\anaconda3\lib\site-packages (from requests->pybithumb) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\anaconda3\lib\site-packages (from requests->pybithumb) (2019.9.11)
Requirement already satisfied: idna<2.9,>=2.5 in c:\anaconda3\lib\site-packages (from requests->pybithumb) (2.8)
Requirement already satisfied: pytz>=2017.2 in c:\anaconda3\lib\site-packages (from pandas->pybithumb) (2019.3)
Requirement already satisfied: numpy>=1.13.3 in c:\anaconda3\lib\site-packages (from pandas->pybithumb) (1.16.5)
Requirement already satisfied: python-dateutil>=2.6.1 in c:\anaconda3\lib\site-packages (from pandas->pybithumb) (2.8.0)
Requirement already satisfied: zope.interface in c:\anaconda3\lib\site-packages (from datetime->pybithumb) (5.4.0)
Requirement already satisfied: beautifulsoup4 in c:\anaconda3\lib\site-packages (from bs4->pybithumb) (4.8.0)
Requirement already satisfied: webencodings in c:\anaconda3\lib\site-packages (from html5lib->pybithumb) (0.5.1)
Requirement already satisfied: six>=1.9 in c:\anaconda3\lib\site-packages (from html5lib->pybithumb) (1.12.0)
Requirement already satisfied: setuptools in c:\anaconda3\lib\site-packages (from zope.interface->datetime->pybithumb) (41.4.0)
Requirement already satisfied: soupsieve>=1.2 in c:\anaconda3\lib\site-packages (from beautifulsoup4->bs4->pybithumb) (1.9.3)
Building wheels for collected packages: bs4
  Building wheel for bs4 (setup.py) ... done
  Created wheel for bs4: filename=bs4-0.0.1-cp37-none-any.whl size=1278 sha256=cd88d708cb2e98dfa2e5157d6ad5b7c192b4d6e99f336c32efa313ef7f9c4d78
    Stored in directory: C:\Users\calvi\AppData\Local\pip\Cache\wheels#a0#b0#b2#4f80b9456b87abedbc0bf2d52235414c3467d8889be38dd472
Successfully built bs4
Installing collected packages: bs4, pybithumb
Successfully installed bs4-0.0.1 pybithumb-1.0.21
(base) C:\Users\calvi>
```

- __2. 가상화폐 거래소에서 사용되는 종목코드인 Ticker 정보를 확인하고, 거래되고 있는 가상화폐 수를 확인합니다.

```
import pybithumb

tickers = pybithumb.get_tickers()
print(tickers)
print(len(tickers))
print(type(tickers))
```

- __3. 거래되고 있는 가상화폐 중, 두개의 가상화폐의 호가정보를 정상적으로 가져오는지를 확인합니다.

```
import pybithumb
import time

while True:
    price = pybithumb.get_current_price("BTC")
    price2 = pybithumb.get_current_price("XRP")
    print(price)
    print(price2)
    time.sleep(1)
```

QTimer 와 같은 주기적으로 데이터를 가져오는 방식으로 Time.sleep(1) 을 추가하여, 1 초에 한번씩 조회하도록 반영해 봅니다.

- __4. 거래되고 있는 가상화폐 중, 두개의 가상화폐의 호가정보를 정상적으로 가져오는지를 확인합니다.

```
import pybithumb
import time

while True:
    price = pybithumb.get_current_price("BTC")
    price2 = pybithumb.get_current_price("XRP")
    print(price)
    print(price2)
    time.sleep(1)
```

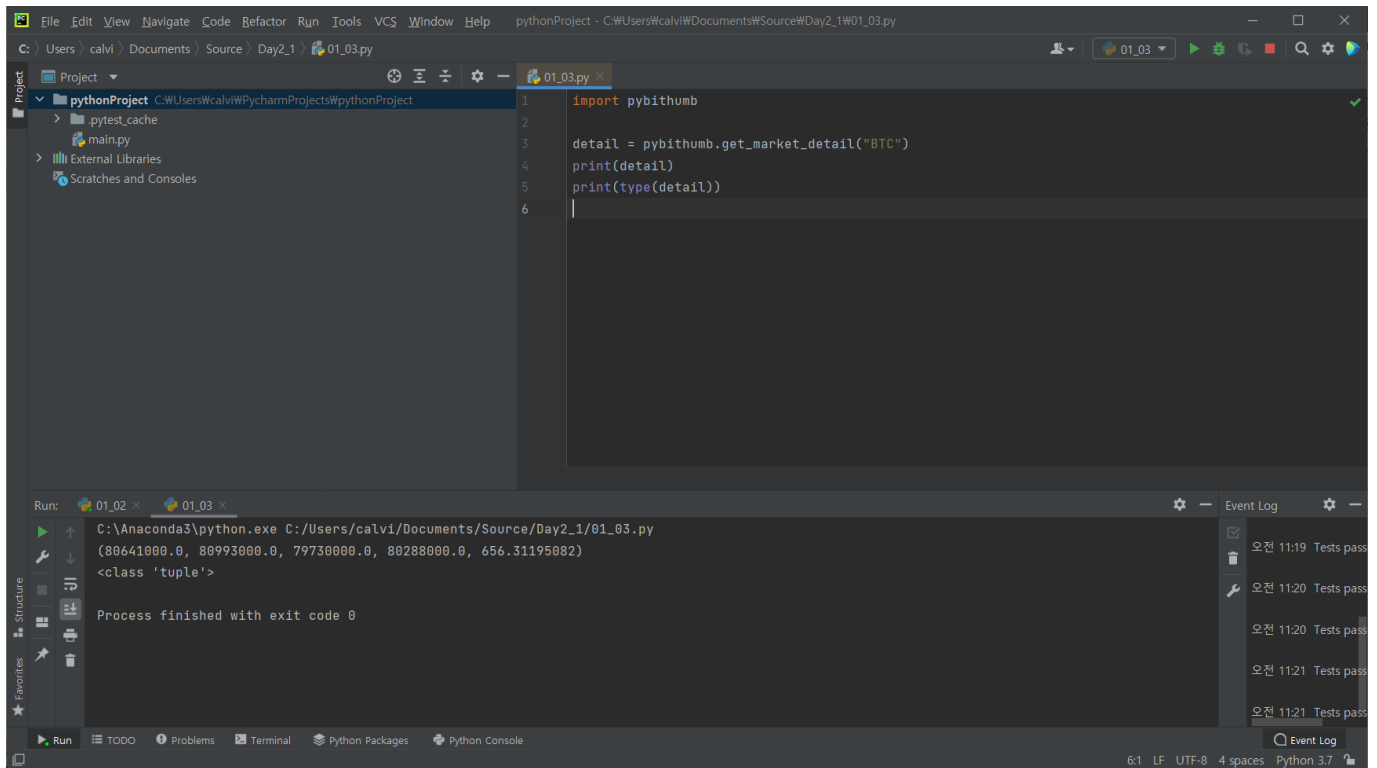
- __5. 가상화폐 거래소의 호가 정보를 가져옵니다.

```
import pybithumb

detail = pybithumb.get_market_detail("BTC")
print(detail)
print(type(detail))
```

가상화폐의 거래정보는 시가/고가/저가/종가/거래량으로 구성되어 있습니다.

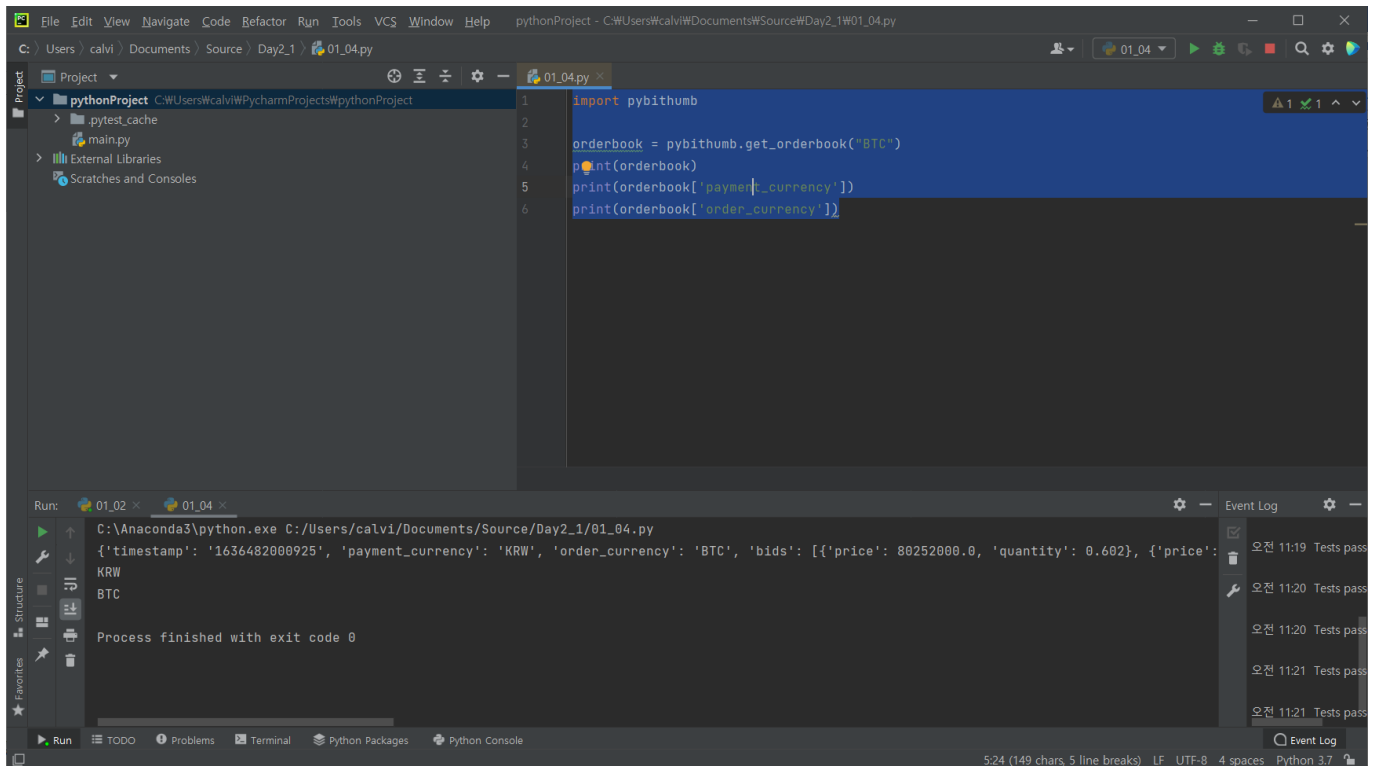
(80641000.0, 80993000.0, 79730000.0, 80280000.0, 634.6283693)



가상화폐의 호가정보를 가져옵니다.

```
import pybithumb

orderbook = pybithumb.get_orderbook("BTC")
print(orderbook)
print(orderbook['payment_currency'])
print(orderbook['order_currency'])
```

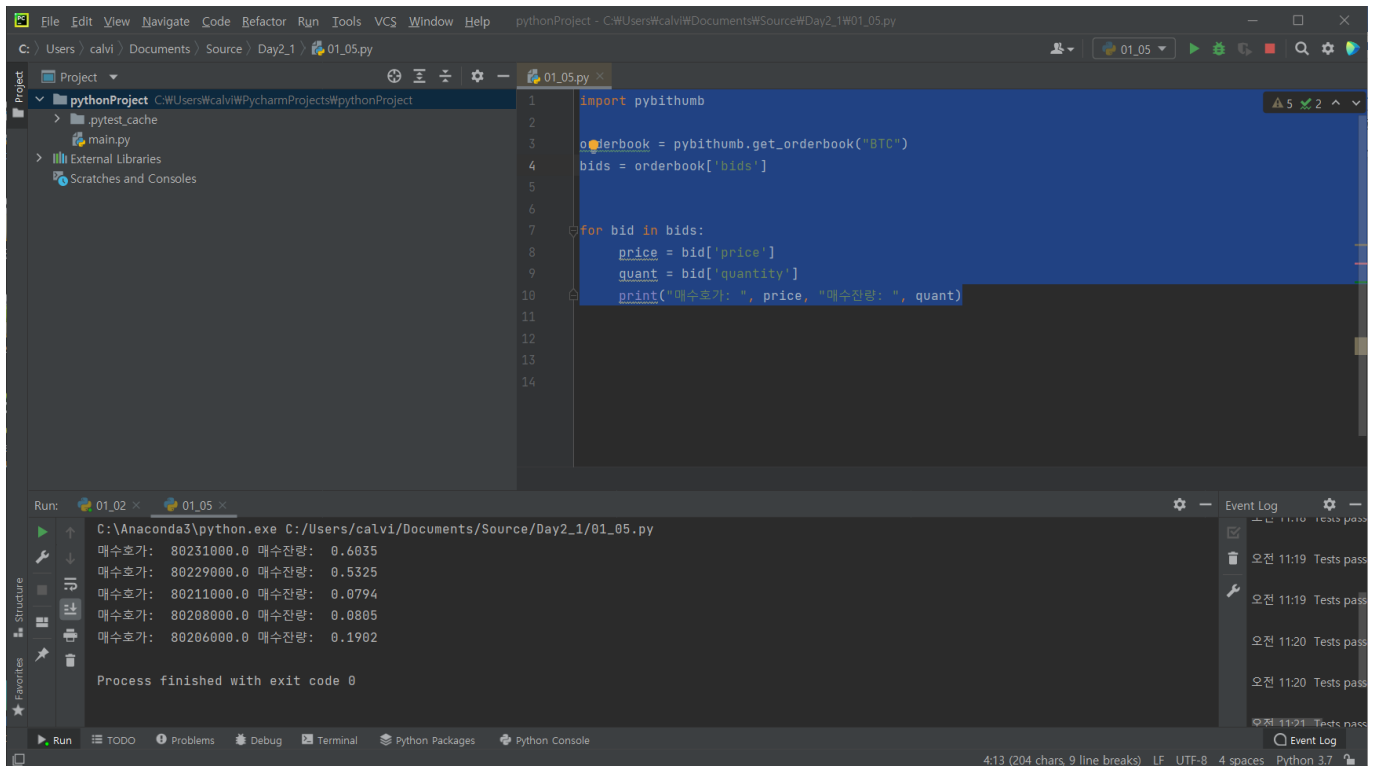
__6. 호가정보 중, 매수호가 정보를 가져옵니다.

```
import pybithumb

orderbook = pybithumb.get_orderbook("BTC")
bids = orderbook['bids']

for bid in bids:
    price = bid['price']
    quant = bid['quantity']
    print("매수호가: ", price, "매수잔량: ", quant)
```

매수호가: 80231000.0 매수잔량: 0.6035
매수호가: 80229000.0 매수잔량: 0.5325
매수호가: 80211000.0 매수잔량: 0.0794
매수호가: 80208000.0 매수잔량: 0.0805
매수호가: 80206000.0 매수잔량: 0.1902



__7. 호가정보 중, 매도호가 정보를 가져옵니다.

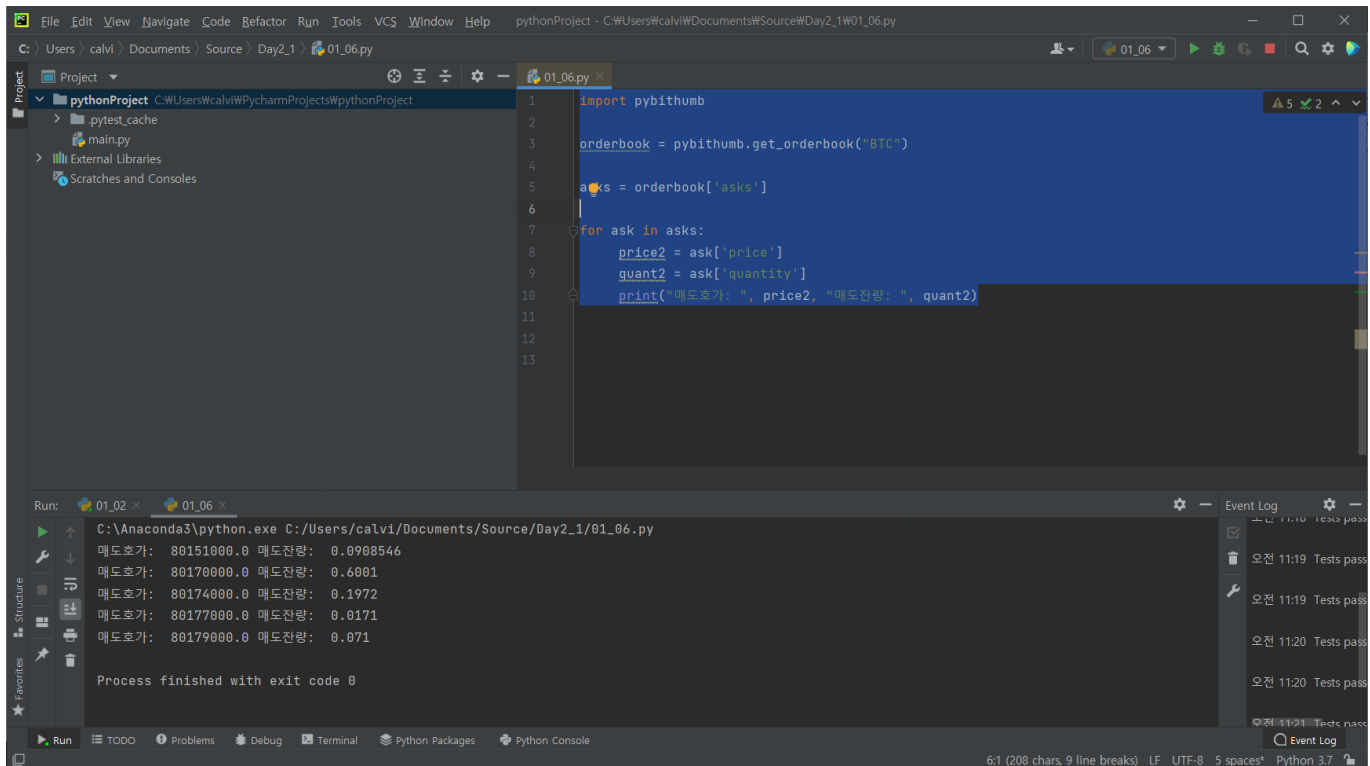
```
import pybithumb

orderbook = pybithumb.get_orderbook("BTC")

asks = orderbook['asks']

for ask in asks:
    price2 = ask['price']
    quant2 = ask['quantity']
    print("매도호가: ", price2, "매도잔량: ", quant2)
```

매도호가: 80151000.0 매도잔량: 0.0908546
매도호가: 80170000.0 매도잔량: 0.6001
매도호가: 80174000.0 매도잔량: 0.1972
매도호가: 80177000.0 매도잔량: 0.0171
매도호가: 80179000.0 매도잔량: 0.071



8. 가상화폐 거래소에 제공하는 가상화폐의 호가정보를 가져오기를 실행합니다.

```
import pybithumb

all = pybithumb.get_current_price("ALL")
for k, v in all.items():
    print(k, v)
```

가상화폐 거래소에 거래되고 있는 모든 가상화폐 정보를 Key, value 형태로 받아오는 작업입니다.

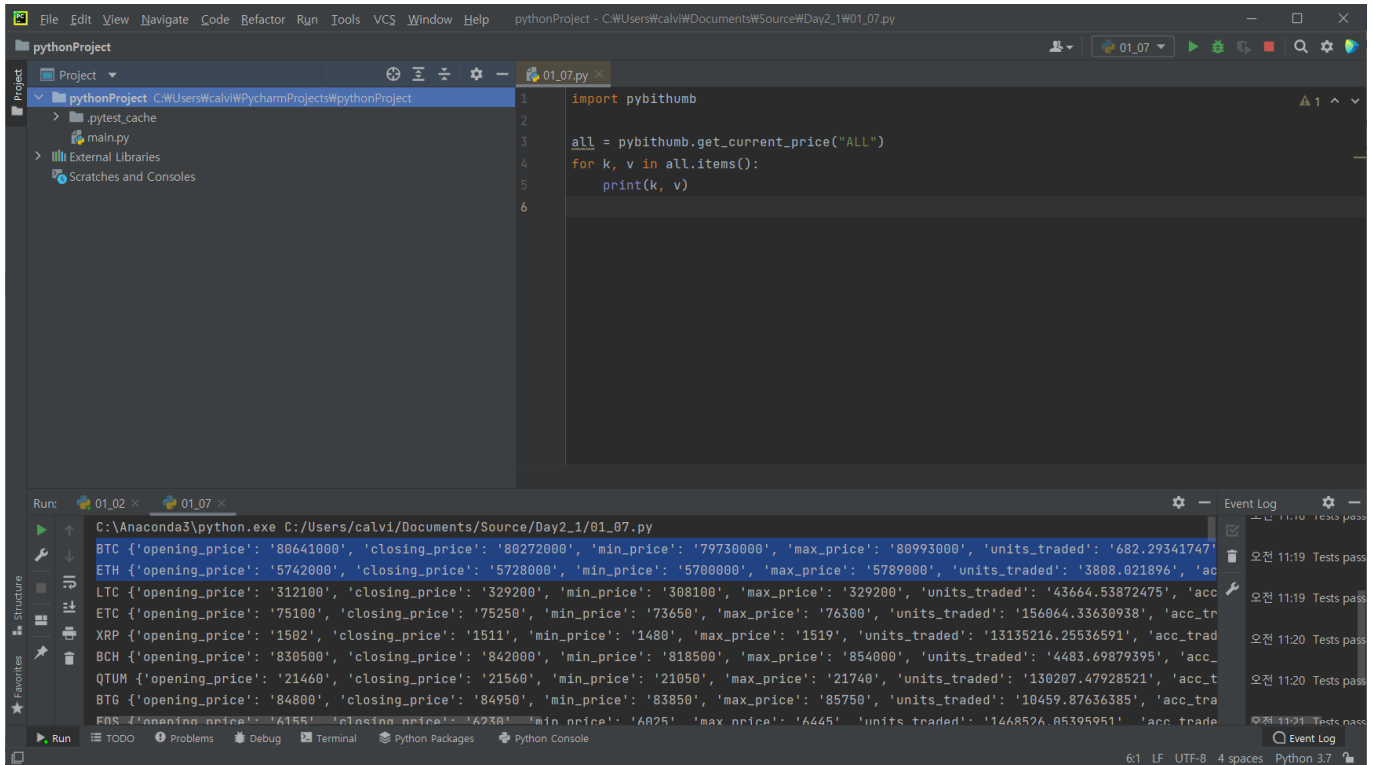
BTC {'opening_price': '80641000', 'closing_price': '80272000', 'min_price': '79730000', 'max_price': '80993000', 'units_traded': '682.29341747', 'acc_trade_value': '54725240799.4875', 'prev_closing_price': '80611000', 'units_traded_24H': '4211.71434852', 'acc_trade_value_24H': '341722681523.7614', 'fluctate_24H': '760000', 'fluctate_rate_24H': '0.96'}

ETH {'opening_price': '5742000', 'closing_price': '5728000', 'min_price': '5700000', 'max_price': '5789000', 'units_traded': '3808.021896', 'acc_trade_value': '21849161004.4018', 'prev_closing_price': '5741000', 'units_traded_24H': '24559.57157032', 'acc_trade_value_24H': '141600954179.3715', 'fluctate_24H': '-45000', 'fluctate_rate_24H': '-0.78'}

가상화폐 가격정보 전체의 데이터를 받게 되면, 위와 같은 형태의 데이터구조로 구성되어 있습니다.

Opening_price : 시작가, / Closing_price : 종가 / min_price : 저가 / max_price : 고가 / units_traded : 거래량 / acc_trade_value, : 거래금액 / prev_closing_price : 전일종가 / units_traded_24H : 24 시간 거래량 / acc_trade_value_24H : 24 시간 거래금액 / fluctate_24H : 24 시간 변동가 / flucate_rate_24H :

24 시간 변동률



The screenshot shows the PyCharm IDE with a project named 'pythonProject'. The main editor displays a file '01_07.py' with the following code:

```
1 import pybithumb
2
3 all = pybithumb.get_current_price("ALL")
4 for k, v in all.items():
5     print(k, v)
6
```

The Run window at the bottom shows the output of the script, which is a JSON object containing market data for various cryptocurrencies:

```
Run: C:\Anaconda3\python.exe C:\Users\calvi\Documents\Source\Day2_1\01_07.py
BTC {'opening_price': '80641000', 'closing_price': '80272000', 'min_price': '79730000', 'max_price': '80993000', 'units_traded': '682.29341747'}
ETH {'opening_price': '5742000', 'closing_price': '5728000', 'min_price': '5700000', 'max_price': '5789000', 'units_traded': '3808.021896', 'acc_trad': '14483.69879395'}
LTC {'opening_price': '312100', 'closing_price': '329200', 'min_price': '308100', 'max_price': '329200', 'units_traded': '43664.53872475', 'acc_trad': '14483.69879395'}
ETC {'opening_price': '75100', 'closing_price': '75250', 'min_price': '73650', 'max_price': '76300', 'units_traded': '156064.33630938', 'acc_trad': '14483.69879395'}
XRP {'opening_price': '1502', 'closing_price': '1511', 'min_price': '1480', 'max_price': '1519', 'units_traded': '13135216.25536591', 'acc_trad': '14483.69879395'}
QTUM {'opening_price': '21460', 'closing_price': '21560', 'min_price': '21050', 'max_price': '21740', 'units_traded': '130207.47928521', 'acc_trad': '14483.69879395'}
BTG {'opening_price': '84800', 'closing_price': '84950', 'min_price': '83850', 'max_price': '85750', 'units_traded': '10459.87636385', 'acc_trad': '14483.69879395'}
```

- __9. 가상화폐 거래소의 호가나 거래 정보를 연계한 로직상에서 여러요인으로 인해 오류 발생시, 트레이딩 프로그램에서는 에러처리를 통해 보완된 로직을 반영하는 컨셉을 가져옵니다..

```
import pybithumb
import time

while True:
    price = pybithumb.get_current_price("BTC")
    try:
        print(price/10)
    except:
        print("error", price)
        time.sleep(0.2)
```

- __10. 가상화폐의 거래정보를 기반으로 이동평균 값을 구성해 봅니다.

이동평균값을 계산하기 위해서는 가상화폐의 거래정보 데이터가 필요합니다. `get_ohlc` 함수를 통해서 가상화폐의 과거 시세정보를 가져옵니다.

```
import pybithumb

btc = pybithumb.get_ohlcv("BTC")
print(btc)

close = btc['close']
print((close[0] + close[1] + close[2] + close[3] + close[4])/5)
print((close[1] + close[2] + close[3] + close[4] + close[5])/5)
print((close[2] + close[3] + close[4] + close[5] + close[6])/5)

print('=====')

window = close.rolling(5)
ma5 = window.mean()
print(ma5)
```

```
time
2013-12-27 00:00:00    ...    755000.0    3.780000
2013-12-28 00:00:00    750000.0    750000.0    ...    750000.0    12.000000
2013-12-29 00:00:00    750000.0    750000.0    ...    739000.0    19.058000
2013-12-30 00:00:00    740000.0    772000.0    ...    768000.0    9.488973
2013-12-31 00:00:00    768000.0    800000.0    ...    768000.0    18.650350
...
2021-11-06 00:00:00    74216000.0    74440000.0    ...    74029000.0    3174.142669
2021-11-07 00:00:00    74026000.0    75771000.0    ...    74914000.0    2544.977262
2021-11-08 00:00:00    74901000.0    80124000.0    ...    79217000.0    3826.749156
2021-11-09 00:00:00    79253000.0    82477000.0    ...    80611000.0    4027.476441
2021-11-10 04:00:00    80641000.0    80993000.0    ...    80038000.0    721.440791
```

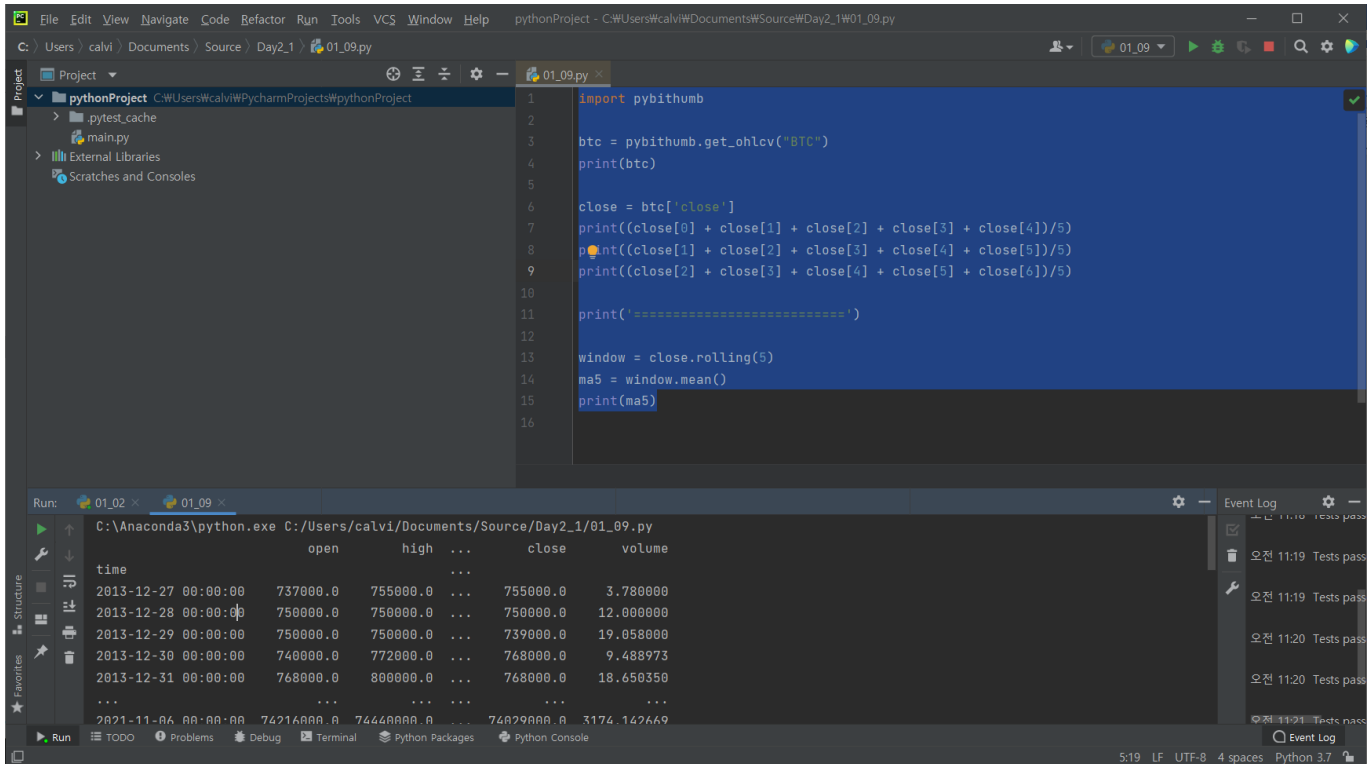
모든 과거 데이터 정보를 가져오게 되며, 이 데이터 중에서 종가(Close) 데이터 기준으로 데이터를 슬라이싱 처리하여 그중에 최신 데이터 5 개 기준으로 평균값을 계산합니다.

```
[2786 rows x 5 columns]
756000.0
760400.0
766000.0
```

이동평균값을 일일이 로직으로 구현하는 방법보다는 `rolling()`과 `mean()` 함수를 통해서 이동평균을 계산해주는 역할을 합니다.

```
time
2013-12-27 00:00:00    NaN
2013-12-28 00:00:00    NaN
2013-12-29 00:00:00    NaN
2013-12-30 00:00:00    NaN
2013-12-31 00:00:00    756000.0
...
2021-11-06 00:00:00    73964800.0
2021-11-07 00:00:00    74039000.0
2021-11-08 00:00:00    75292400.0
```

2021-11-09 00:00:00 76598800.0
2021-11-10 04:00:00 77761800.0
Name: close, Length: 2786, dtype: float64



The screenshot shows the PyCharm IDE interface. The top pane displays a Python script named `01_09.py` with the following code:

```
1 import pybithumb
2
3 btc = pybithumb.get_ohlcv("BTC")
4 print(btc)
5
6 close = btc['close']
7 print((close[0] + close[1] + close[2] + close[3] + close[4])/5)
8 print((close[1] + close[2] + close[3] + close[4] + close[5])/5)
9 print((close[2] + close[3] + close[4] + close[5] + close[6])/5)
10
11 print('=====')
12
13 window = close.rolling(5)
14 ma5 = window.mean()
15 print(ma5)
16
```

The bottom pane shows the output of the script, which is a table of OHLCV data for Bitcoin from December 27, 2013, to December 31, 2013. The columns are `time`, `open`, `high`, `low`, `close`, and `volume`.

time	open	high	low	close	volume
2013-12-27 00:00:00	737000.0	755000.0	...	755000.0	3.780000
2013-12-28 00:00:00	750000.0	750000.0	...	750000.0	12.000000
2013-12-29 00:00:00	750000.0	750000.0	...	739000.0	19.058000
2013-12-30 00:00:00	740000.0	772000.0	...	768000.0	9.488973
2013-12-31 00:00:00	768000.0	800000.0	...	768000.0	18.650350
...
2021-11-11 00:00:00	74216000.0	74460000.0	...	74020000.0	3174.147669

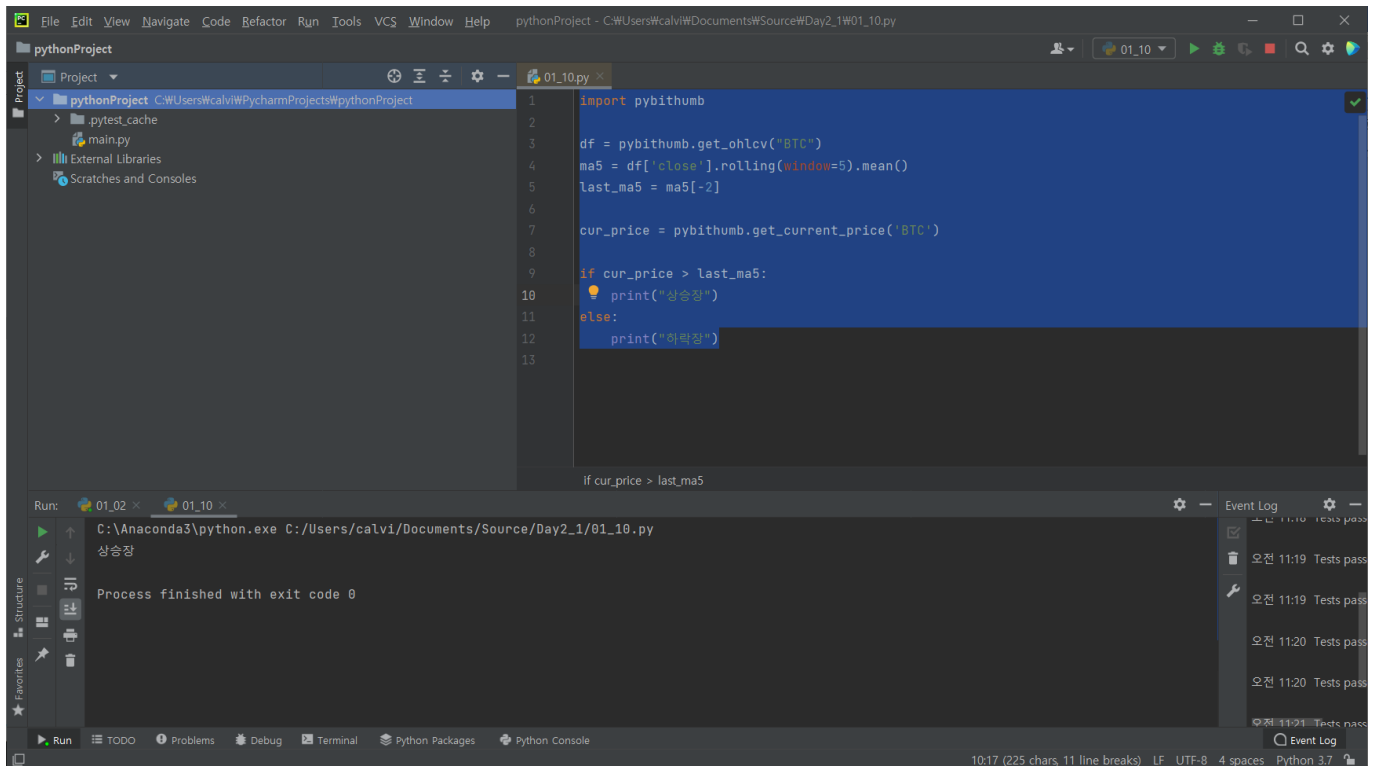
__11. 전일 이동평균 가격을 기준으로, 호출 기준의 가상화폐 가격과 비교하여, 상승장과 하락장을 표시해 봅니다.

```
import pybithumb

df = pybithumb.get_ohlcv("BTC")
ma5 = df['close'].rolling(window=5).mean()
last_ma5 = ma5[-2]

cur_price = pybithumb.get_current_price('BTC')

if cur_price > last_ma5:
    print("상승장")
else:
    print("하락장")
```



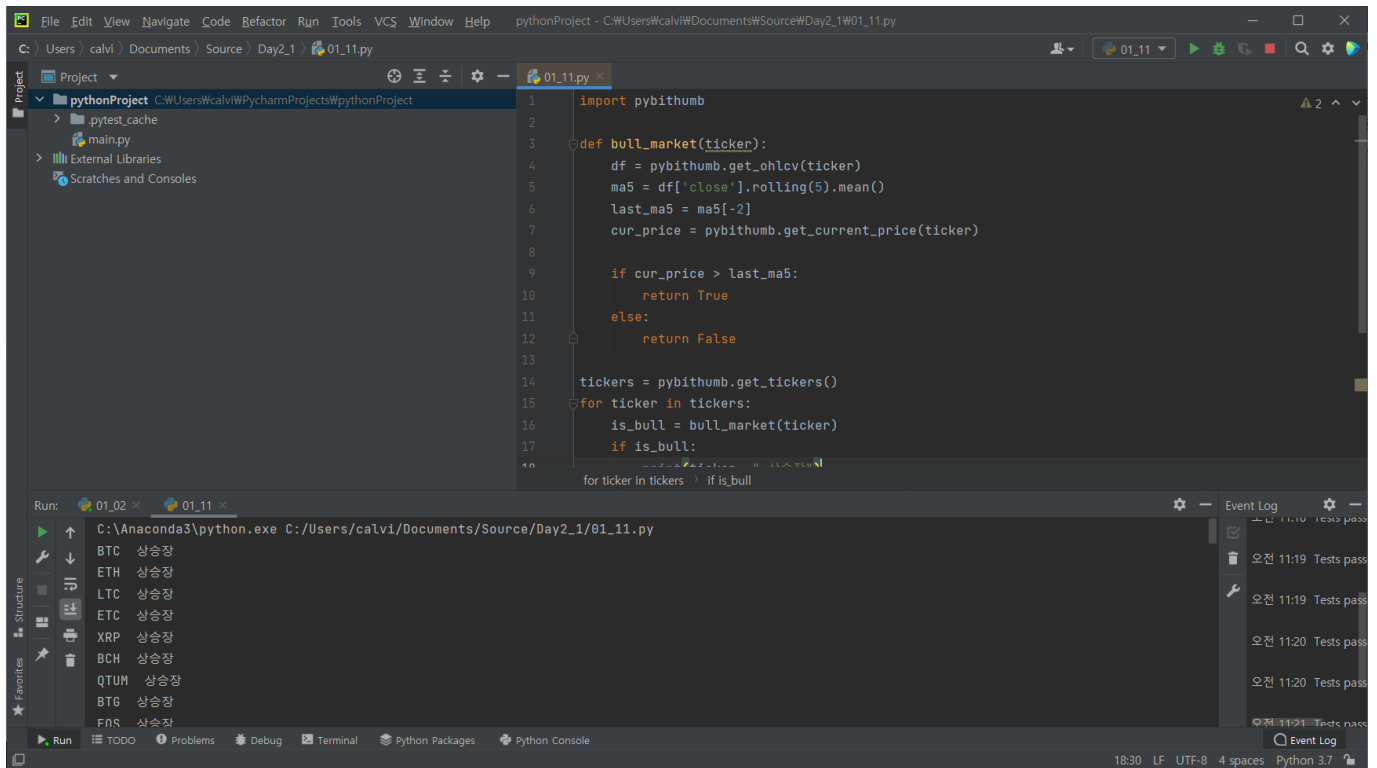
12. 이동평균값을 처리하는 로직을 반영하여, 가상화폐 거래소에 제공하는 전체 가상화폐를 대상으로 전일 평균가격과의 비교를 통해 상승장과 하락장인지 확인해 봅니다.

```
import pybithumb

def bull_market(ticker):
    df = pybithumb.get_ohlc(ticker)
    ma5 = df['close'].rolling(5).mean()
    last_ma5 = ma5[-2]
    cur_price = pybithumb.get_current_price(ticker)

    if cur_price > last_ma5:
        return True
    else:
        return False

tickers = pybithumb.get_tickers()
for ticker in tickers:
    is_bull = bull_market(ticker)
    if is_bull:
        print(ticker, " 상승장")
    else:
        print(ticker, " 하락장")
```



Section 2: UI 와 연계 작업

가상화폐 거래소의 데이터를 기반으로 이동평균가격과 상승장, 하락장의 로직을 기반으로 UI 상에서 보여주는 작업을 진행합니다.

__1. QT Designer 를 실행하고, Main Window 기반 UI 를 생성합니다.

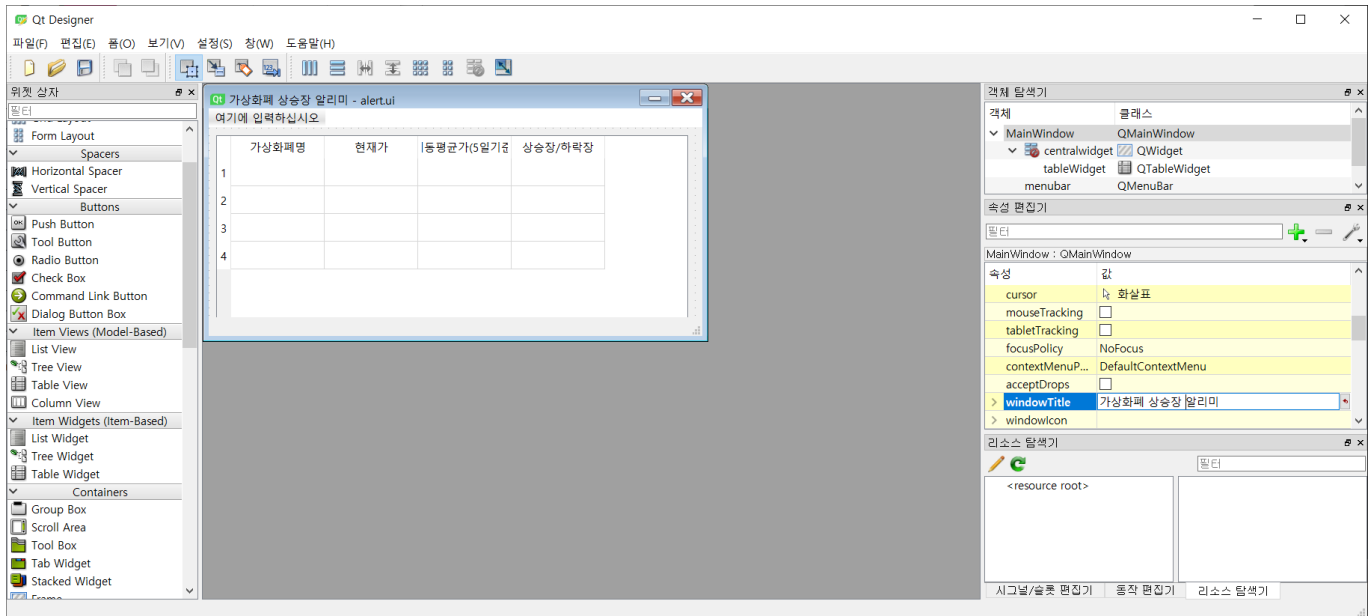


Table Widget 를 Drag&Drop 을 통해서 알리미 UI 를 구성 후, 저장합니다.

__2. 생성한 UI 를 호출하고, Table Widget 의 컬럼명 중에 가상화폐명 가져와서 UI 에 반영하는 작업을 진행합니다.

UI 상에서 pushbutton 없이, QTimer 를 통해서 5 초 간격으로 데이터를 호출하도록 구성합니다.

```

import sys
from PyQt5.QtWidgets import *
from PyQt5 import uic
from PyQt5.QtCore import *

tickers = ["BTC", "ETH", "XRP", "ETC"]
form_class = uic.loadUiType("alert.ui")[0]

class MyWindow(QMainWindow, form_class):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

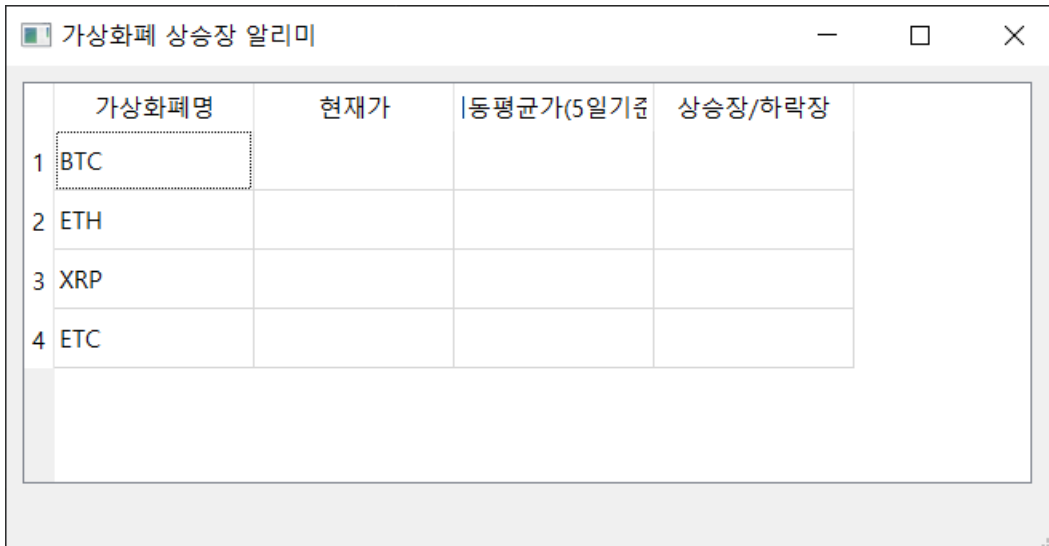
        timer = QTimer(self)
        timer.start(5000)
        timer.timeout.connect(self.timeout)

    def timeout(self):
        for i, ticker in enumerate(tickers):
            item = QTableWidgetItem(ticker)
            self.tableWidget.setItem(i, 0, item)

app = QApplication(sys.argv)
window = MyWindow()
window.show()
app.exec_()

```

QTableWidgetItem 을 통해서, Tickers 정보를 받아서, UI 컬럼에 입력하는 작업이 정상적으로 진행되었는지 확인합니다.



	가상화폐명	현재가	이동평균가(5일기준)	상승장/하락장
1	BTC			
2	ETH			
3	XRP			
4	ETC			

- __3. 가상화폐의 현재가, 이동평균가격, 상승장, 하락장 여부를 판단하기 위한 데이터를 가져오기 위해, `get_market_infos` 함수에서 과거시세 정보 조회하고, 이를 기반으로 이동평균가격계산 및 상승장, 하락장 여부를 판단합니다.

```
import sys
from PyQt5.QtWidgets import *
from PyQt5 import uic
from PyQt5.QtCore import *
import pybithumb

tickers = ["BTC", "ETH", "XRP", "ETC"]
form_class = uic.loadUiType("alert.ui")[0]

class MyWindow(QMainWindow, form_class):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        timer = QTimer(self)
        timer.start(5000)
        timer.timeout.connect(self.timeout)

    def get_market_infos(self, ticker):
        df = pybithumb.get_ohlc(ticker)
        ma5 = df['close'].rolling(window=5).mean()
        last_ma5 = ma5[-2]
        price = pybithumb.get_current_price(ticker)

        state = None
        if price > last_ma5:
            state = "상승장"
        else:
            state = "하락장"

        return price, last_ma5, state

    def timeout(self):
        for i, ticker in enumerate(tickers):
            item = QTableWidgetItem(ticker)
            self.tableWidget.setItem(i, 0, item)

            price, last_ma5, state = self.get_market_infos(ticker)
            self.tableWidget.setItem(i, 1, QTableWidgetItem(str(price)))
            self.tableWidget.setItem(i, 2, QTableWidgetItem(str(last_ma5)))
            self.tableWidget.setItem(i, 3, QTableWidgetItem(state))

app = QApplication(sys.argv)
window = MyWindow()
window.show()
app.exec_()
```

Timeout 함수를 통해서, 5 초 간격으로 데이터를 가져오는 구조로 구성하여, 5 초 간격으로 UI 에 데이터가 업데이트가 됩니다.

가상화폐 상승장 알리미				
	가상화폐명	현재가	동평균가(5일기준)	상승장/하락장
1	BTC	80159000.0	76598800.0	상승장
2	ETH	5702000.0	5566800.0	상승장
3	XRP	1510.0	1452.6	상승장
4	ETC	74000.0	67070.0	상승장

Section 3: 상승장 알리미 Performance 개선

가상화폐 거래소에서 거래정보를 받아오는 과정에서 QTimer 기반의 로직은 파이썬 인터프리터가 데이터를 조회하고 UI에 입력하는 작업을 반복적으로 수행함으로 인해, 느려지는 현상이 발생하게 됩니다.

네트워크 문제나, 기타 다른 문제로 인해 데이터를 조회하지 못하는 문제가 발생한다면 이를 개선하기 위해 PyQt5에서 제공하는 QThread를 적용합니다.

__1. Qthread가 동작되는 방식을 확인해 봅니다.

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

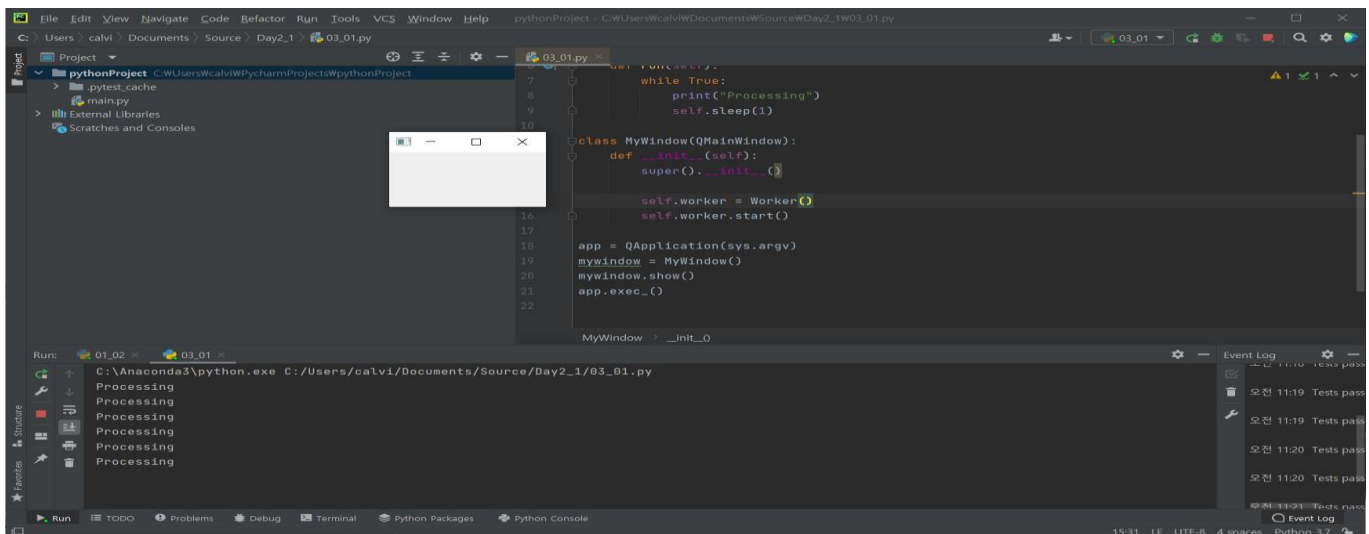
class Worker(QThread):
    def run(self):
        while True:
            print("Processing")
            self.sleep(1)

class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.worker = Worker()
        self.worker.start()

app = QApplication(sys.argv)
mywindow = MyWindow()
mywindow.show()
app.exec_()
```

QThread를 상속받아, Worker 클래스를 정의하고, 이 클래스를 기반으로 start를 수행합니다. 여기서 수행 간격은 1초로 (self.sleep(1))으로 지정하고, 처리되는 방식입니다



__2. Qthread 를 가상화폐 상승장 알리미 로직에 추가하는 작업을 진행합니다.

```
import sys
from PyQt5.QtWidgets import *
from PyQt5 import uic
from PyQt5.QtCore import *
import pybithumb
import time

tickers = ["BTC", "ETH", "XRP", "ETC"]
form_class = uic.loadUiType("alert.ui")[0]

class Worker(QThread):
    def run(self):
        while True:
            data = {}

            for ticker in tickers:
                data[ticker] = self.get_market_infos(ticker)

            print(data)
            time.sleep(5)

    def get_market_infos(self, ticker):
        try:
            df = pybithumb.get_ohlc(ticker)
            ma5 = df['close'].rolling(window=5).mean()
            last_ma5 = ma5[-2]
            price = pybithumb.get_current_price(ticker)

            state = None
            if price > last_ma5:
                state = "상승장"
            else:
                state = "하락장"

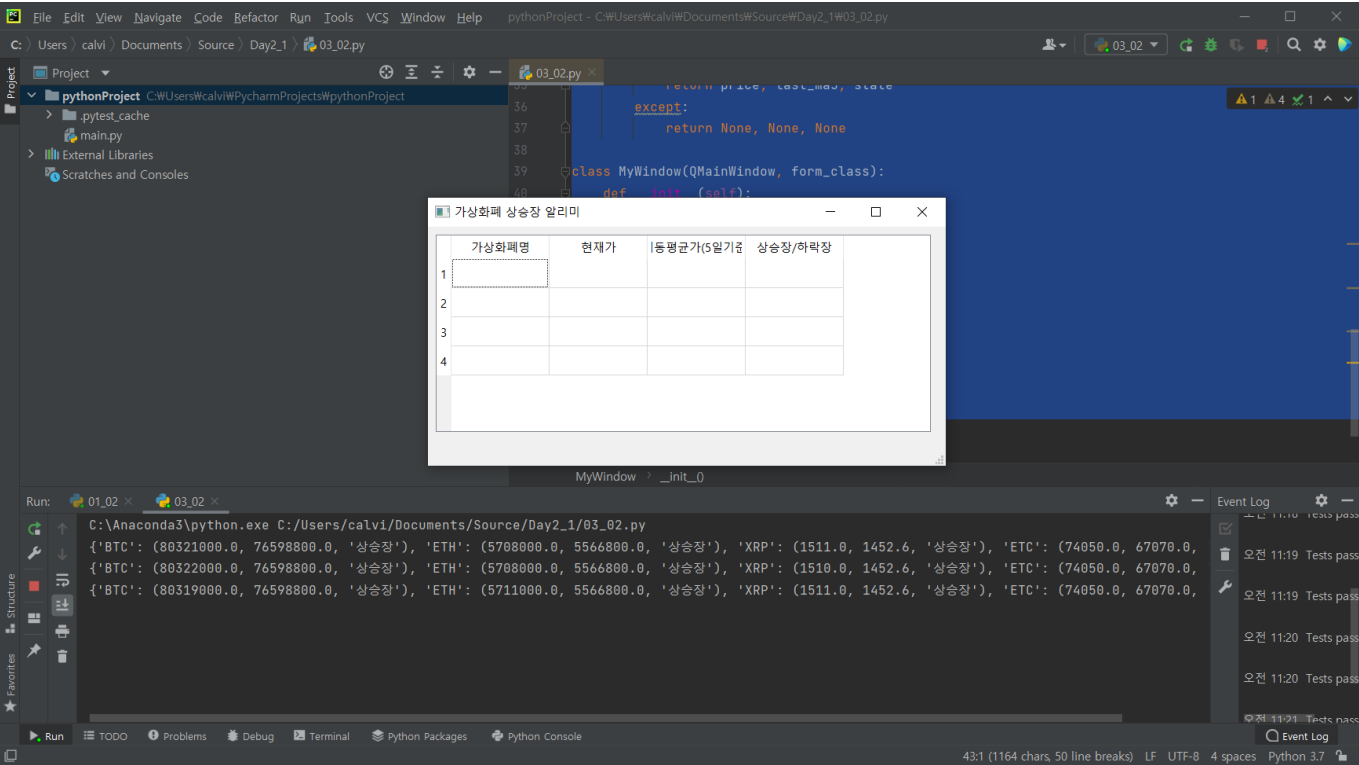
            return price, last_ma5, state
        except:
            return None, None, None

class MyWindow(QMainWindow, form_class):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        self.worker = Worker()
        self.worker.start()

app = QApplication(sys.argv)
window = MyWindow()
window.show()
app.exec_()
```

데이터를 갱신하는 영역에 **QThread**를 반영하여, 간격주기를 지정하고 처리되는 구조로 업데이트 합니다.



__3. Qthread에 포함된 데이터 조회 및 처리 로직에서 데이터 처리시점을 인식하기 위한 이벤트 발생(emit)을 반영합니다.

```
import sys
from PyQt5.QtWidgets import *
from PyQt5 import uic
from PyQt5.QtCore import *
import pybithumb
import time

tickers = ["BTC", "ETH", "XRP", "ETC"]
form_class = uic.loadUiType("alert.ui")[0]

class Worker(QThread):
    finished = pyqtSignal(dict)

    def run(self):
        while True:
            data = {}

            for ticker in tickers:
                data[ticker] = self.get_market_infos(ticker)

            self.finished.emit(data)
            time.sleep(2)

    def get_market_infos(self, ticker):
        try:
            df = pybithumb.get_ohlc(ticker)
            ma5 = df['close'].rolling(window=5).mean()
            last_ma5 = ma5[-2]
            price = pybithumb.get_current_price(ticker)

            state = None
            if price > last_ma5:
                state = "상승장"
            else:
                state = "하락장"

            return price, last_ma5, state
        except:
            return None, None, None

class MyWindow(QMainWindow, form_class):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        self.worker = Worker()
        self.worker.finished.connect(self.update_table_widget)
        self.worker.start()
```

데이터 처리가 완료된 시점에서, UI에 데이터를 반영하는 로직을 호출하도록 추가합니다.


```

@pyqtSlot(dict)
def update_table_widget(self, data):
    try:
        for ticker, infos in data.items():
            index = tickers.index(ticker)

            self.tableWidget.setItem(index, 0,
QTableWidgetItem(ticker))
            self.tableWidget.setItem(index, 1,
QTableWidgetItem(str(infos[0])))
            self.tableWidget.setItem(index, 2,
QTableWidgetItem(str(infos[1])))
            self.tableWidget.setItem(index, 3,
QTableWidgetItem(str(infos[2])))
        except:
            pass

app = QApplication(sys.argv)
window = MyWindow()
window.show()
app.exec_()

```

가상화폐 상승장 알리미에서 데이터가 정상적으로 보이는 것을 확인할 수 있습니다.

The screenshot shows the PyCharm IDE interface. The main editor displays a Python script with a function `update_table_widget` that updates a Qt table widget with cryptocurrency data. The script uses `QTableWidgetItem` to populate the table. Below the editor, a terminal window titled "가상화폐 상승장 알리미" (Cryptocurrency Bull Market Alert) displays the following data:

	가상화폐명	현재가	1동평균가(5일기준)	상승장/하락장
1	BTC	80324000.0	76598800.0	상승장
2	ETH	5709000.0	5566800.0	상승장
3	XRP	1509.0	1452.6	상승장
4	ETC	73950.0	67070.0	상승장

The terminal output also shows the command `C:\Anaconda3\python.exe C:/Users/calvi/Documents/Source/Day2_1/03_03.py` and the status "Tests pass" in the Event Log.

연습문제

지금까지 실습을 진행하면서, 가상화폐 상승장 알리미에, 5 개의 가상화폐를 추가해서 UI를 구동해 보세요