

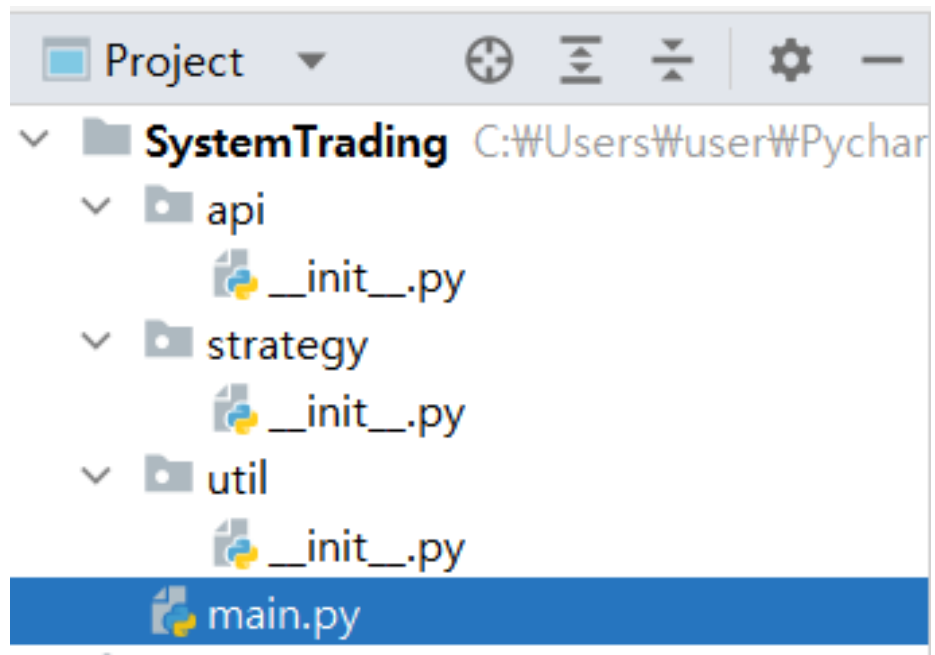
2022-07-26

[SW집중교육]

주식매매자동 시스템

목차

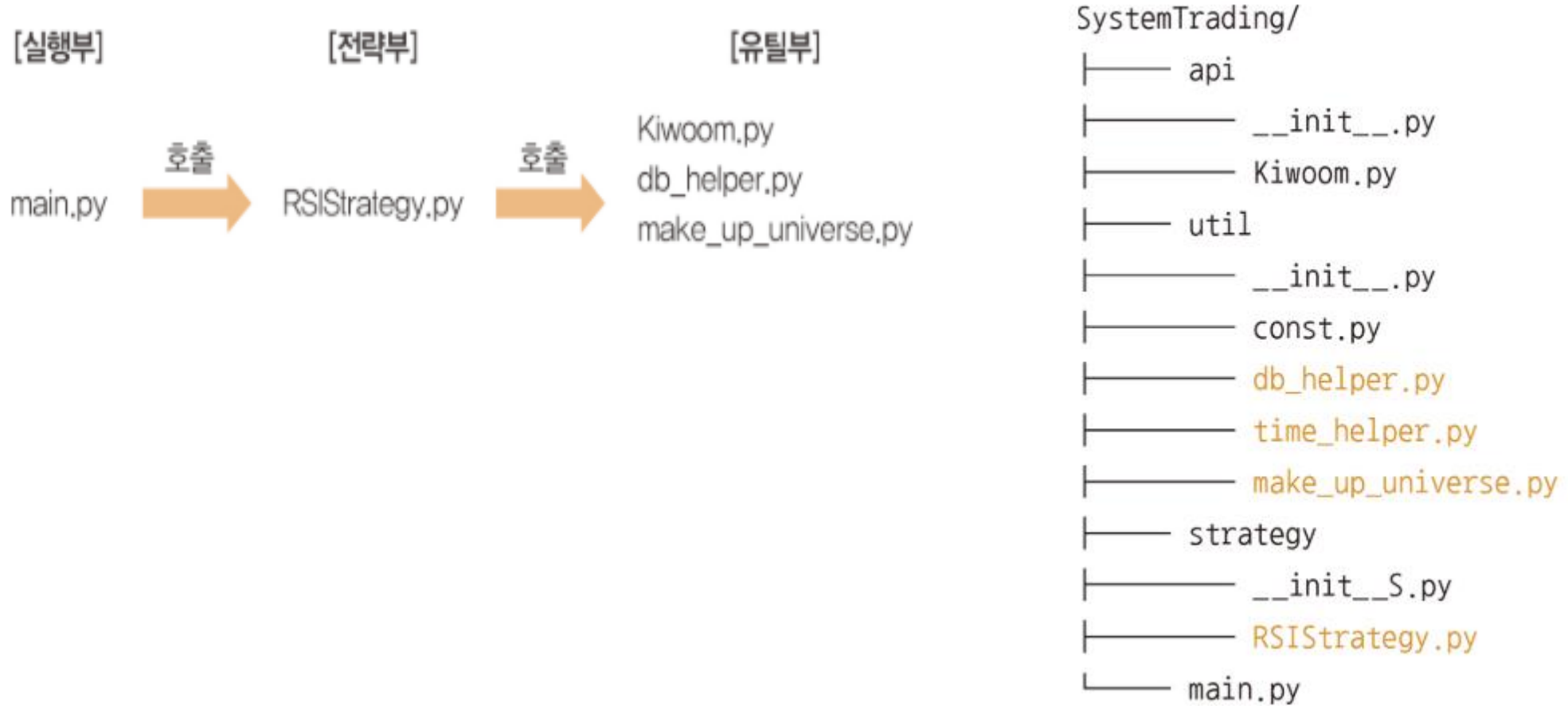
프로젝트 구조



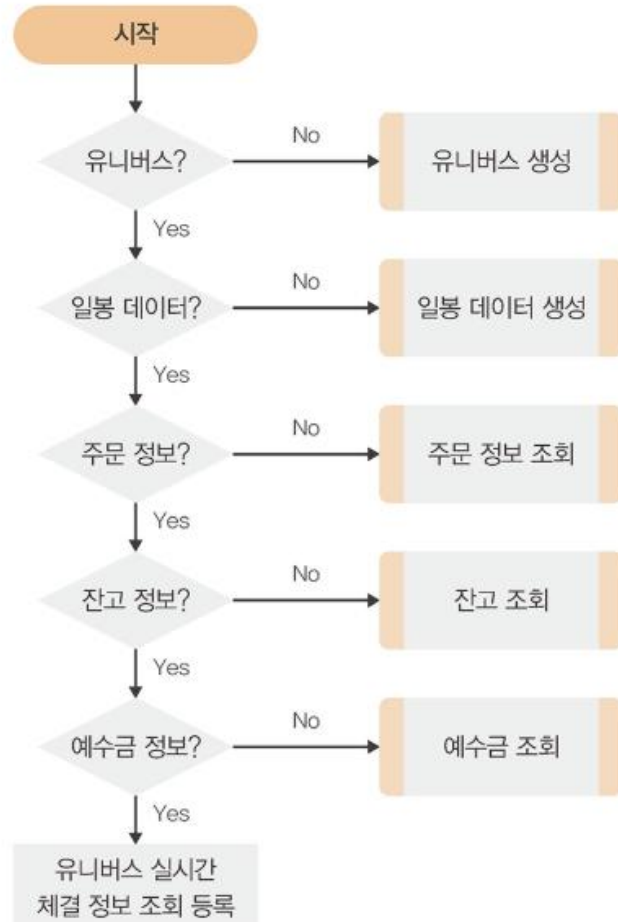
API 비동기식 처리방식



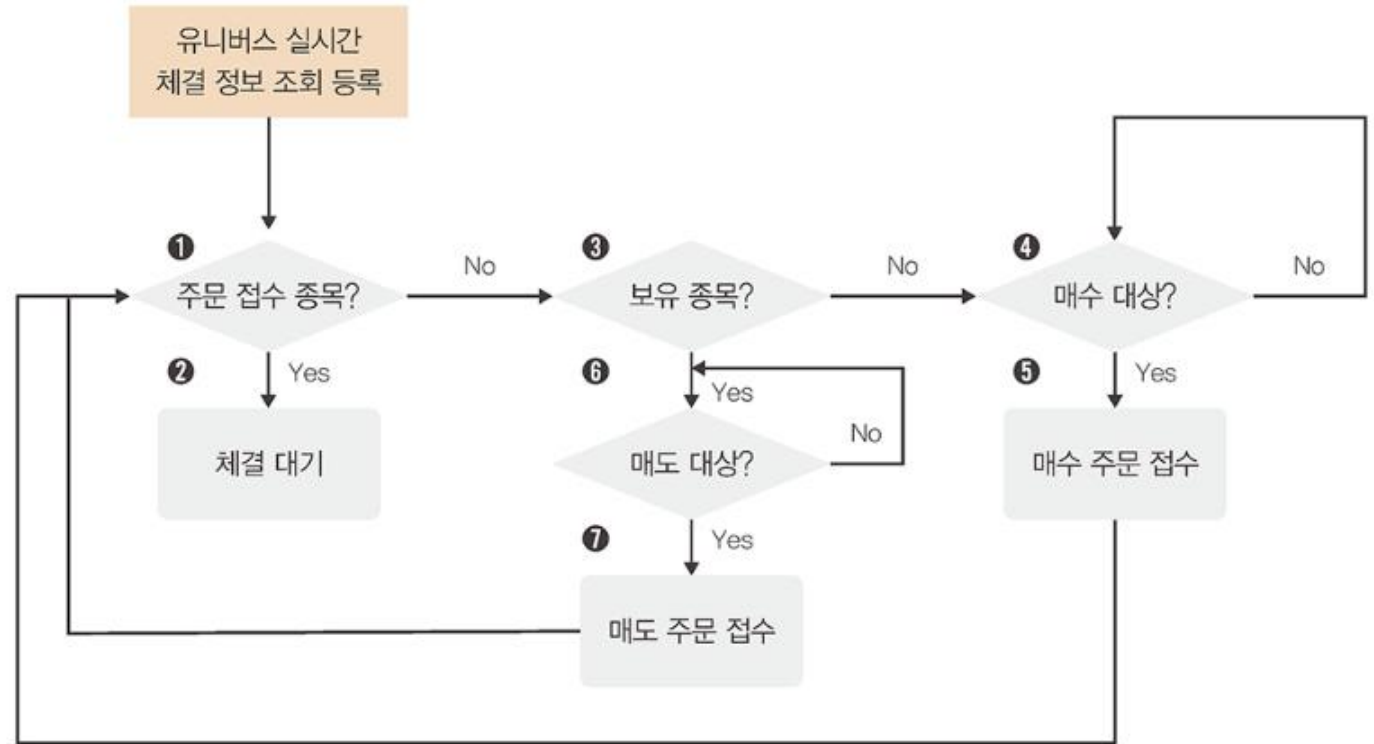
프로젝트 구조



흐름도

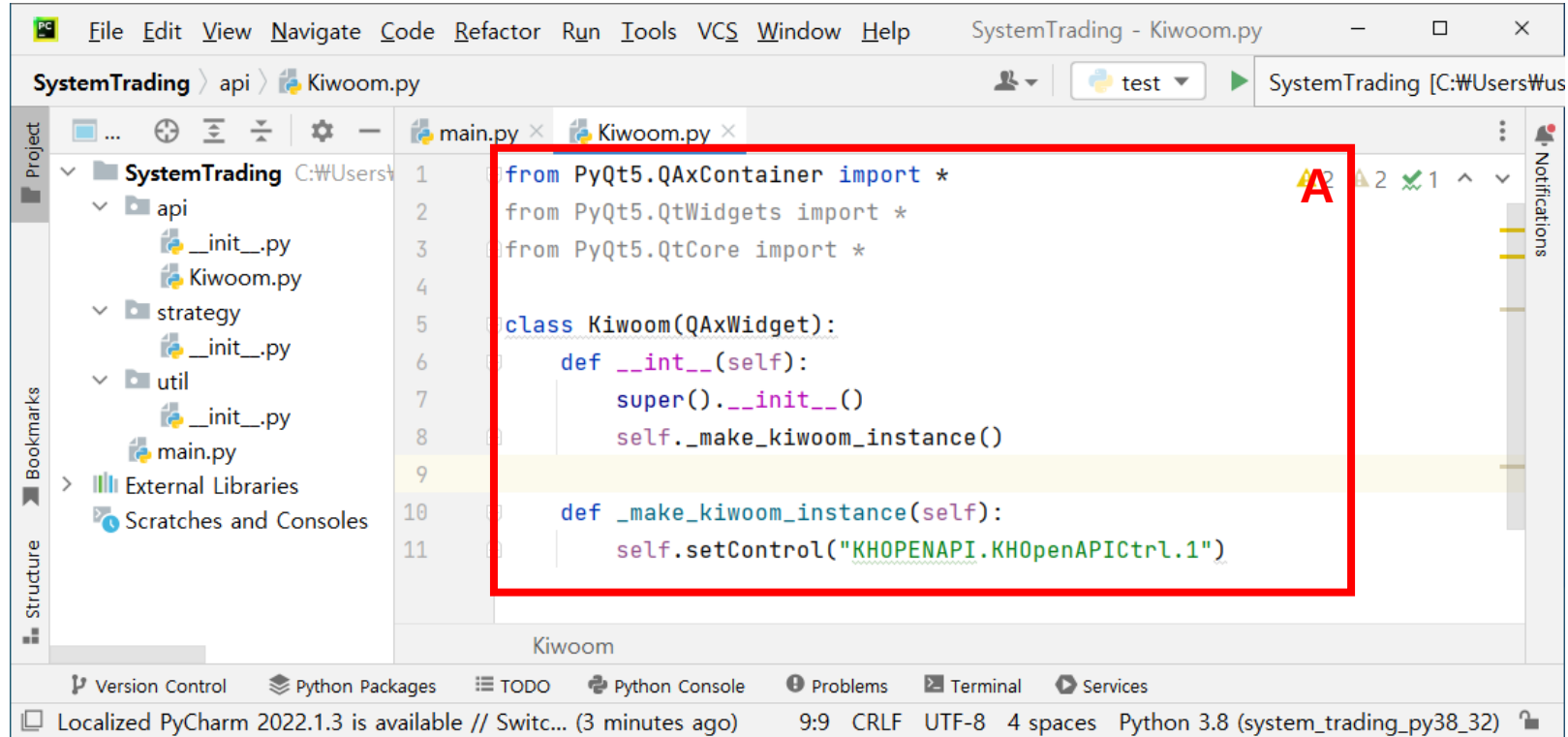


논리 구조 흐름



자동매매 프로그램의 동작 흐름도

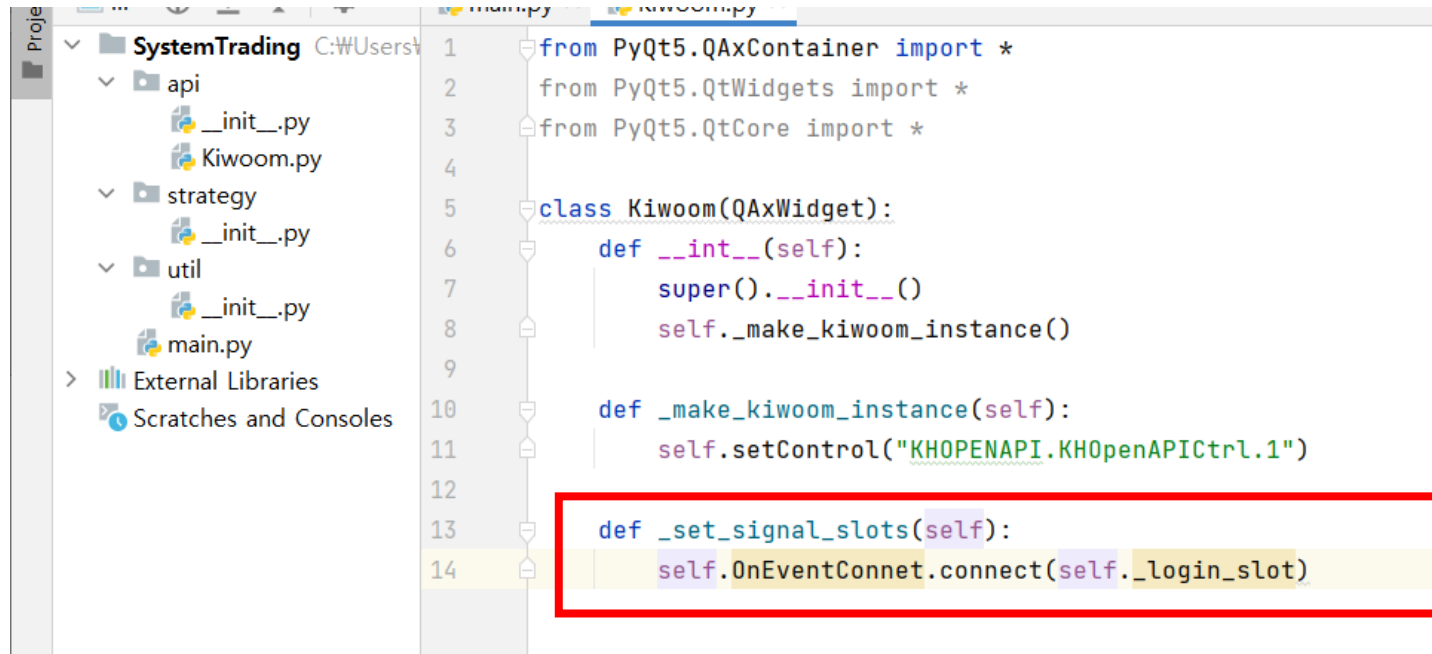
1. 증권사 로그인하기



1. Kiwoom.py파일 만들기
2. A의 내용 코딩하기

■ 동작과정

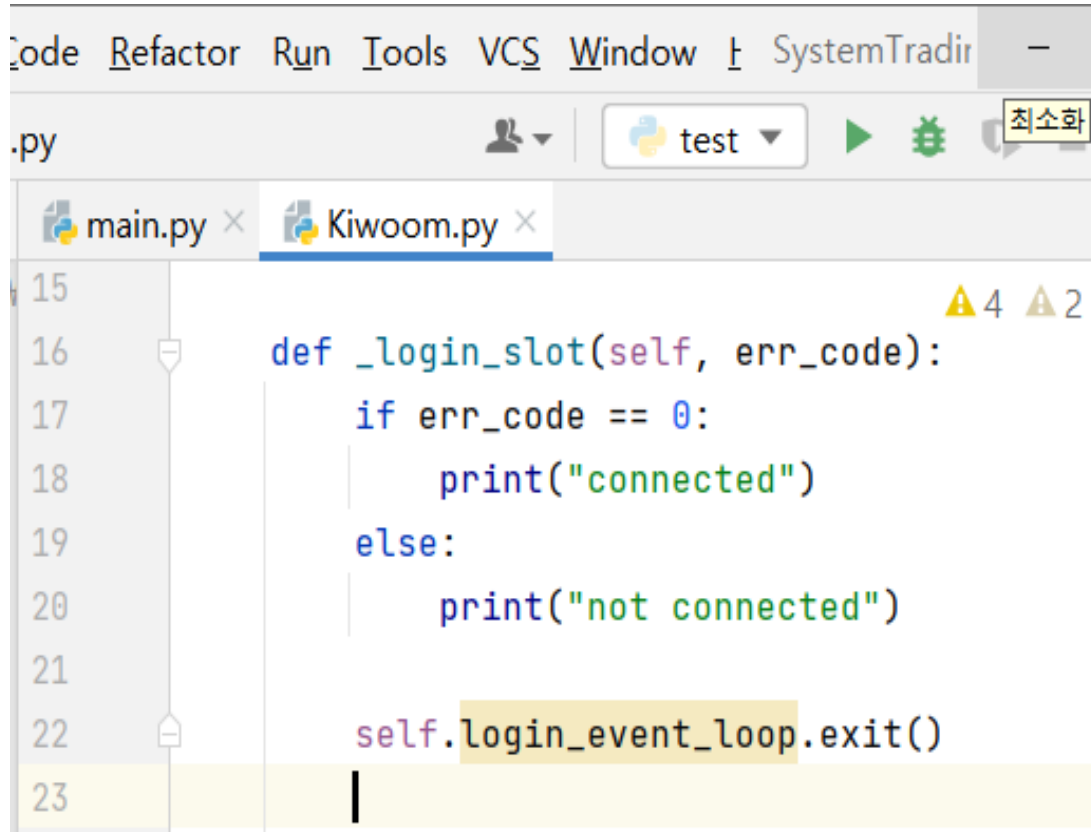
- 로그인 처리에 대한 응답을 받을 스롯(slot)함수 생성 및 등록
- 로그인요청
- a에서 만든 slot함수를 사용하여 응답 확인



```
1 from PyQt5.QAxContainer import *
2 from PyQt5.QtWidgets import *
3 from PyQt5.QtCore import *
4
5 class Kiwoom(QAxWidget):
6     def __init__(self):
7         super().__init__()
8         self._make_kiwoom_instance()
9
10    def _make_kiwoom_instance(self):
11        self.setControl("KHOPENAPI.KHOpenAPICtrl.1")
12
13    def _set_signal_slots(self):
14        self.OnEventConnet.connect(self._login_slot)
```

Api로 보내는 요청들을 받아올
slot등록 함수

self._login_slot 함수 만들기



The screenshot shows a Python IDE with two tabs: 'main.py' and 'Kiwoom.py'. The 'Kiwoom.py' tab is active, displaying the following code:

```
15
16 def _login_slot(self, err_code):
17     if err_code == 0:
18         print("connected")
19     else:
20         print("not connected")
21
22     self.login_event_loop.exit()
23
```

Line numbers 15 through 23 are visible on the left. The code defines a method `_login_slot` that takes `self` and `err_code` as arguments. It uses an `if` statement to check if `err_code` is 0. If true, it prints "connected". Otherwise, it prints "not connected". Finally, it calls `self.login_event_loop.exit()`. There are yellow warning icons (triangles) next to lines 16 and 22, with numbers 4 and 2 respectively. A '최소화' (Minimize) button is visible in the top right corner of the IDE window.

■ 매개변수 : `err_code`

- 로그인 처리에 대한 성공/실패구분값을 전달 받는다.
- 값이 0이면 로그인 성공(`connected`)
아니면 로그인 실패 (`not connected`)

```

class Kiwoom(QAxWidget):
    def __init__(self):
        super().__init__()
        self._make_kiwoom_instance()
        self._set_signal_slots()

    def _make_kiwoom_instance(self):
        self.setControl("KHOPENAPI.KHOpenAPICtrl.1")

    def _set_signal_slots(self):
        self.OnEventConnet.connect(self._login_slot)

    def _login_slot(self, err_code):
        if err_code == 0:
            print("connected")
        else:
            print("not connected")

        self.login_event_loop.exit()

```

생성자 추가

Self._comm_connect()

```
class Kiwoom(QAxWidget):
    def __init__(self):
        super().__init__()
        self._make_kiwoom_instance()
        self._set_signal_slots()
        self._comm_connect()

    def _make_kiwoom_instance(self):
        self.setControl("KHOPENAPI.KHOpenAPICtrl.1")

    def _set_signal_slots(self):
        self.OnEventConnect.connect(self._login_slot)

    def _login_slot(self, err_code):
        if err_code == 0:
            print("connected")
        else:
            print("not connected")

        self.login_event_loop.exit()

    def _comm_connect(self):
        self.dynamicCall("CommConnect()")

        self.login_event_loop = QEventLoop()
        self.login_event_loop.exec_()
```

로그인을 요청하는 함수

로그인완성

```
class Kiwoom(QAxWidget):  
    def __init__(self):  
        super().__init__()  
        self._make_kiwoom_instance()  
        self._set_signal_slots()  
        self._comm_connect()
```

생성자

```
def _make_kiwoom_instance(self):  
    self.setControl("KHOPENAPI.KHOpenAPICtrl.1")
```

1. 설치한 api사용할수 있도록 설정

```
def _set_signal_slots(self):  
    self.OnEventConnect.connect(self._login_slot)
```

4. 로그인 요청에 대한 응답을 이 함수를 사용하여 등록한 슬롯(2번)에서 받아 옴

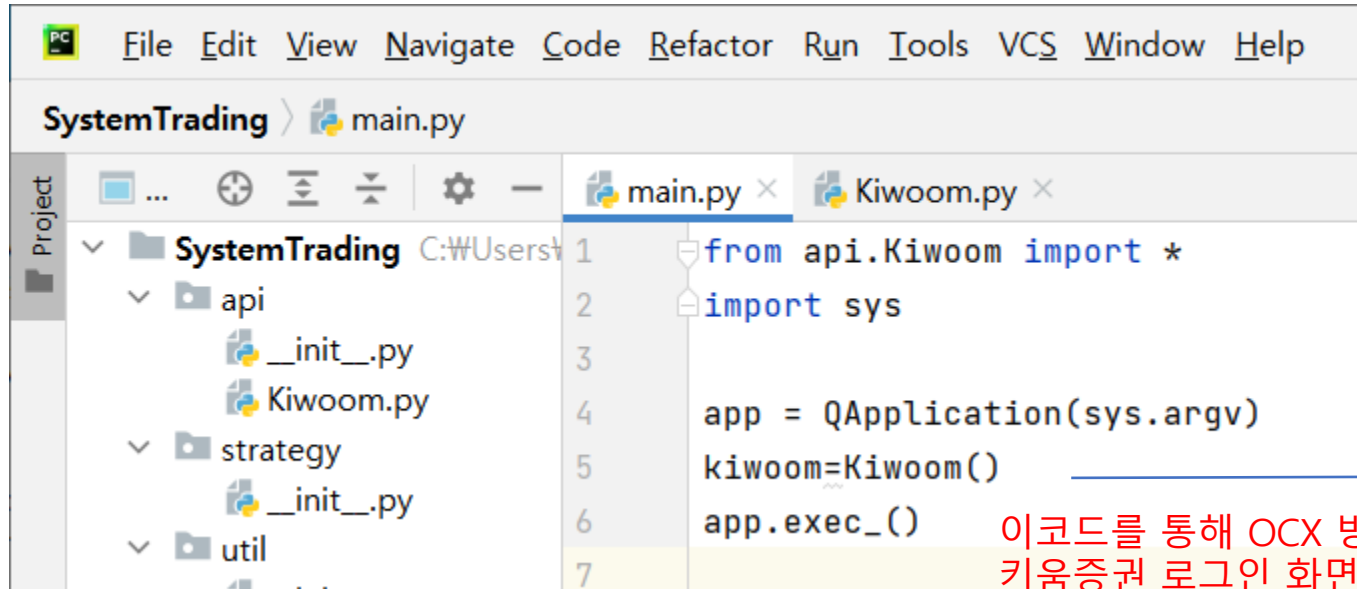
```
def _login_slot(self, err_code):  
    if err_code == 0:  
        print("connected")  
    else:  
        print("not connected")  
  
    self.login_event_loop.exit()
```

2. 로그인, 실시간 정보, 기타 제공받을 수 있는 데이터에 대한 응답 받을 수있는 slot등록

```
def _comm_connect(self):  
    self.dynamicCall("CommConnect()")  
  
    self.login_event_loop = QEventLoop()  
    self.login_event_loop.exec_()
```

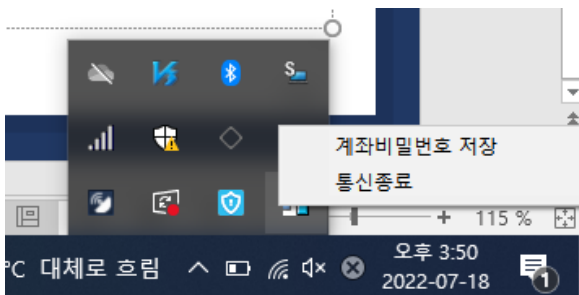
3. 로그인 요청 보내기

로그인(실행결과)



```
1 from api.Kiwoom import *
2 import sys
3
4 app = QApplication(sys.argv)
5 kiwoom=Kiwoom()
6 app.exec_()
7
```

이코드를 통해 OCX 방식 api 사용하여
키움증권 로그인 화면 실행가능



자동로그인을 위해 계좌비밀번호 입력 후
등록하여 사용하면 편리

2. 계좌 정보

■ 로그인 이후 본인 계좌 정보 얻어오는 방법

```
① def get_account_number(self, tag="ACCNO"):
②     account_list = self.dynamicCall("GetLoginInfo(QString)", tag)
③     account_number = account_list.split(';')[0]
④     print(account_number, account_list)
⑤     return account_number
```

- `account_list = self.dynamicCall("GetLoginInfo(QString)", tag)`
- `dynamicCall()`을 사용하여 로그인에 성공한 사용자 정보를 얻어오는 API 함수인 `GetLoginInfo`를 호출
- `tag` : `account_numbergkatndm1` 매개변수이지만, 함수선언때 계좌번호를 의미하는 `ACCNO`로 고정
- `USER_ID` : 사용자id, `USER_NAME`: 사용자이름,
`GetServerGubun`: 접속 서버 구분(1:모의투자, 그외: 실거래서버)

계좌 정보(실행결과)

- 함수로 얻어 온 계좌정보는
account_number 변수에 저장됨

```
Kiwoom.py x main.py x
22     print("connected")
23 else:
24     print("not connected")
25
26     self.login_event_loop.exit()
27
28 def _comm_connect(self):
29     self.dynamicCall("CommConnect()")
30
31     self.login_event_loop = QEventLoop()
32     self.login_event_loop.exec_()
33
34 #계좌 정보 얻기
35 def get_account_number(self, tag="ACCN0"):
36     account_list = self.dynamicCall("GetLoginInfo(QString)", tag)
37     account_number = account_list.split(';')[0]
38     print(account_number, account_list)
39     return account_number

Kiwoom > get_account_number()

main x
C:\Users\user\anaconda3\envs\system_trading_py38_32\python.exe C:/Users/us
connected
8027945811 8027945811;
```

Tag로 전달한 요청에 대한 응답을 받아옴

얻어온 계좌 정보(결과)

3. 종목 정보

■ GetCodeListByMarket()

GetCodeListByMarket(BSTR sMarket) # BSTR sMarket은 시장구분값
[시장구분]

0: 코스피, 10: 코스닥 3: ELW 8: ETF 50: KONEX

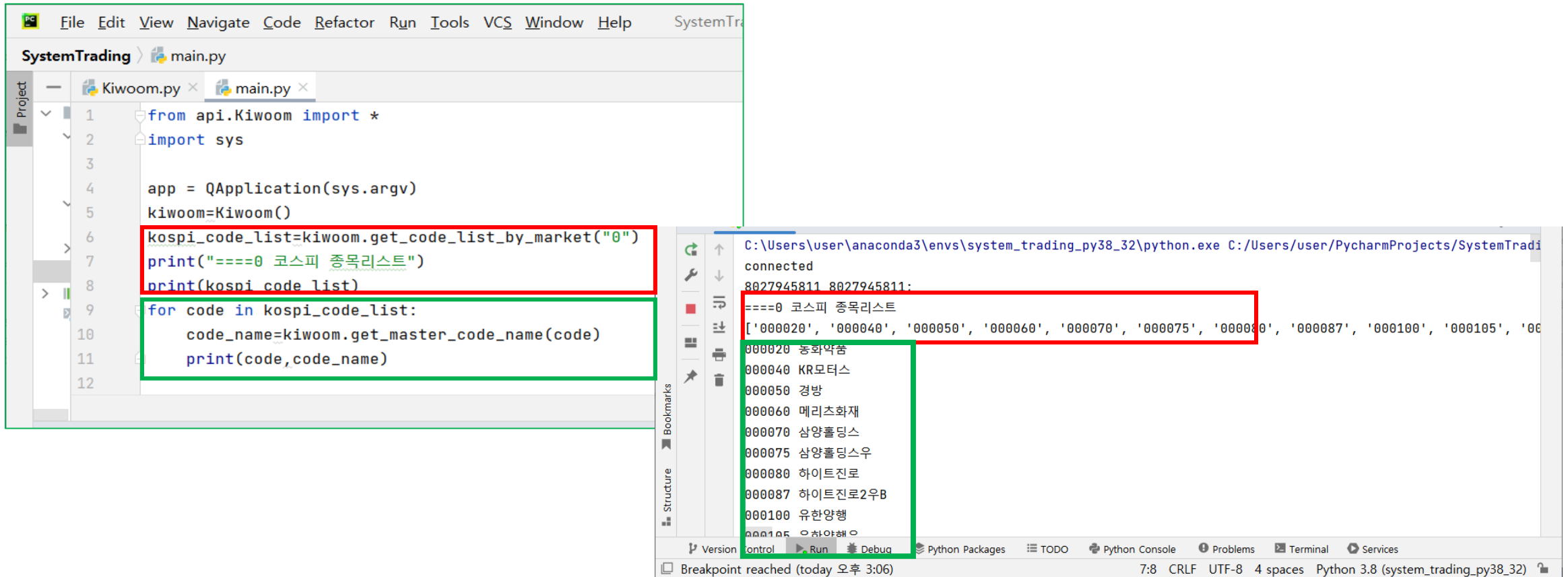
4: 뮤추얼펀드 5: 신주인수권 6: 리츠 9 하이얼펀드 30: K-OTC

```
def get_code_list_by_market(self, market_type):  
    code_list = self.dynamicCall("GetCodeListByMarket(QString)", market_type)  
    code_list = code_list.split(';')[:-1]  
    return code_list
```

■ 종목이름

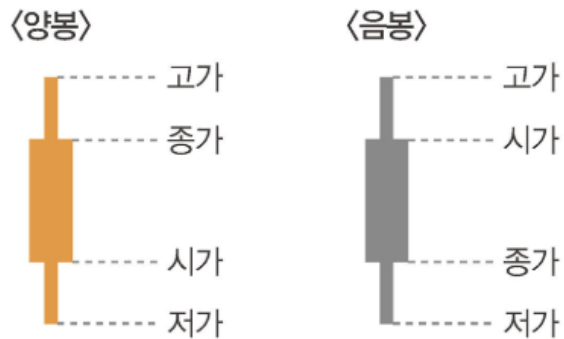
```
def get_master_code_name(self, code):  
    code_name =  
    self.dynamicCall("GetMasterCodeName(QString)", code)  
    return code_name
```

종목 정보(실행결과)



4. 가격 정보 (일봉)캠

- 일봉 : 거래일 동안의 주가변동을 캔들 차트로 표현한것
 - 시가, 고가, 저가, 종가
 - 양봉 : 시작 가격보다 상승하는 것 (빨간색)
 - 음봉 : 시작 가격보다 하락하는 것 (파란색)



TR별로 데이터 오기

- 키움증권 API서버에 전달하는 요청단위: TR
 - CommRqData함수를 사용하여 TR호출

[CommRqData() 함수]

```
CommRqData(  
    BSTR sRQName,    // 사용자 구분명(임의로 지정, 한글 지원)  
    BSTR sTrCode,     // 조회하려는 TR 이름  
    long nPrevNext,   // 연속 조회 여부  
    BSTR sScreenNo    // 화면 번호(4자리 숫자 임의로 지정)  
)
```

리턴값이 0이면 조회요청 정상

- 호출 함수 : self.dynamicCall()

매개변수 1
self.dynamicCall("CommRqData(QString, QString, int, QString)", "opt10081_req", "opt10081", 0, "0001")
API에서 제공하는 함수

매개변수 2 매개변수 3 매개변수 4 매개변수 5
↓ ↓ ↓ ↓
사용자 구분명 TR 이름 연속 조회 여부 화면 번호

TR요청에 대한 응답을 받는 역할 : slot

■ Kiwoom.py파일에

```
def _set_signal_slots(self):  
    self.OnEventConnect.connect(self._login_slot)
```

```
self.OnReceiveTrData.connect(self._on_receive_tr_data)
```

TR의 응답 결과를 _on_receive_tr_data로 받도록 설정

[OnReceiveTrData() 이벤트]

```
void OnReceiveTrData(  
    BSTR sScrNo,      // 화면 번호  
    BSTR sRQName,     // 사용자 구분명  
    BSTR sTrCode,     // TR 이름  
    BSTR sRecordName, // 레코드 이름  
    BSTR sPrevNext,   // 연속 조회 유무를 판단하는 값(0: 연속(추가 조회) 데이터 없음, 2: 연속(추가 조회) 데이터 있음)  
    LONG nDataLength, // 사용 안 함.  
    BSTR sErrorCode,  // 사용 안 함.  
    BSTR sMessage,    // 사용 안 함.  
    BSTR sSplmMsg     // 사용 안 함.  
)
```

일봉(실행결과: 삼성전자)

The screenshot shows a PyCharm IDE window with a Python script named 'Kiwoom.py'. The script is running, and the output is displayed in the 'Run' console. The output shows a series of 'connected' messages and a table of stock data. The table has columns for 'open', 'high', 'low', 'close', and 'volume'. The data is organized into two groups, one for '198501' and one for '202207'. The 'Run' button is highlighted, and a 'Generate' button is visible in the output area.

	open	high	low	close	volume
19850104	130	130	129	129	111765
19850105	129	129	128	128	108497
19850107	129	130	128	129	771895
19850108	129	129	127	127	845098
19850109	126	126	122	123	324837
...
20220712	58600	58700	58100	58100	9336061
20220713	58300	58600	58000	58000	10841315
20220714	57500	58200	57400	57500	15067012
20220715	58400	60000	58100	60000	18685583
20220718	60600	62000	60500	61900	20545439

5. 예수금 갖고 오기

```
116
117     self.tr_data = ohlcv
118
119     elif rqname == "opw00001_req":
120         deposit = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname,
121         self.tr_data = int(deposit)
122         print(self.tr_data)
123
124     self.tr_event_loop.exit()
125     time.sleep(0.5)
126
127     #예수금 갖고 오기
128     def get_deposit(self):
129         self.dynamicCall("SetInputValue(QString, QString)", "계좌번호", self.account_number)
130         self.dynamicCall("SetInputValue(QString, QString)", "비밀번호입력매체구분", "00")
131         self.dynamicCall("SetInputValue(QString, QString)", "조회구분", "2")
132         self.dynamicCall("CommRqData(QString, QString, int, QString)", "opw00001_req", "opw00001",
133         self.tr_event_loop.exec_()
134         return self.tr_data
```

Run: main

```
C:\Users\user\anaconda3\envs\system_trading_py38_32\python.exe C:/Users/user/PycharmProjects/SystemTradi
connected
8027945811 8027945811;
[Kiwoom] _on_receive_tr_data is called 0002 / opw00001_req / opw00001
10000000
```

모의투자 신청때 설정한 예수금 : 1000만
(다르게 했을 경우 설정한 금액 출력)

6. 주문 접수확인

- 주문 접수 > 주문 체결 > 잔고 이동

- 시장가 주문 vs 지정가 주문

- 주문 처리 단계

- 주문발생(SendOrder) → 주문응답(OnReceiveTrData)

사용자가 호출
리턴값이 0인 경우 정상

주문 발생시 첫번째 서버응답
주문 번호취득(주문번호없음 주문거부)

→ 주문 메시지 수신(OnReceiveMsg) → 주문 접수/체결(OnReceiveChejan)

주문거부 사유를 포함한 서버 메시지수신

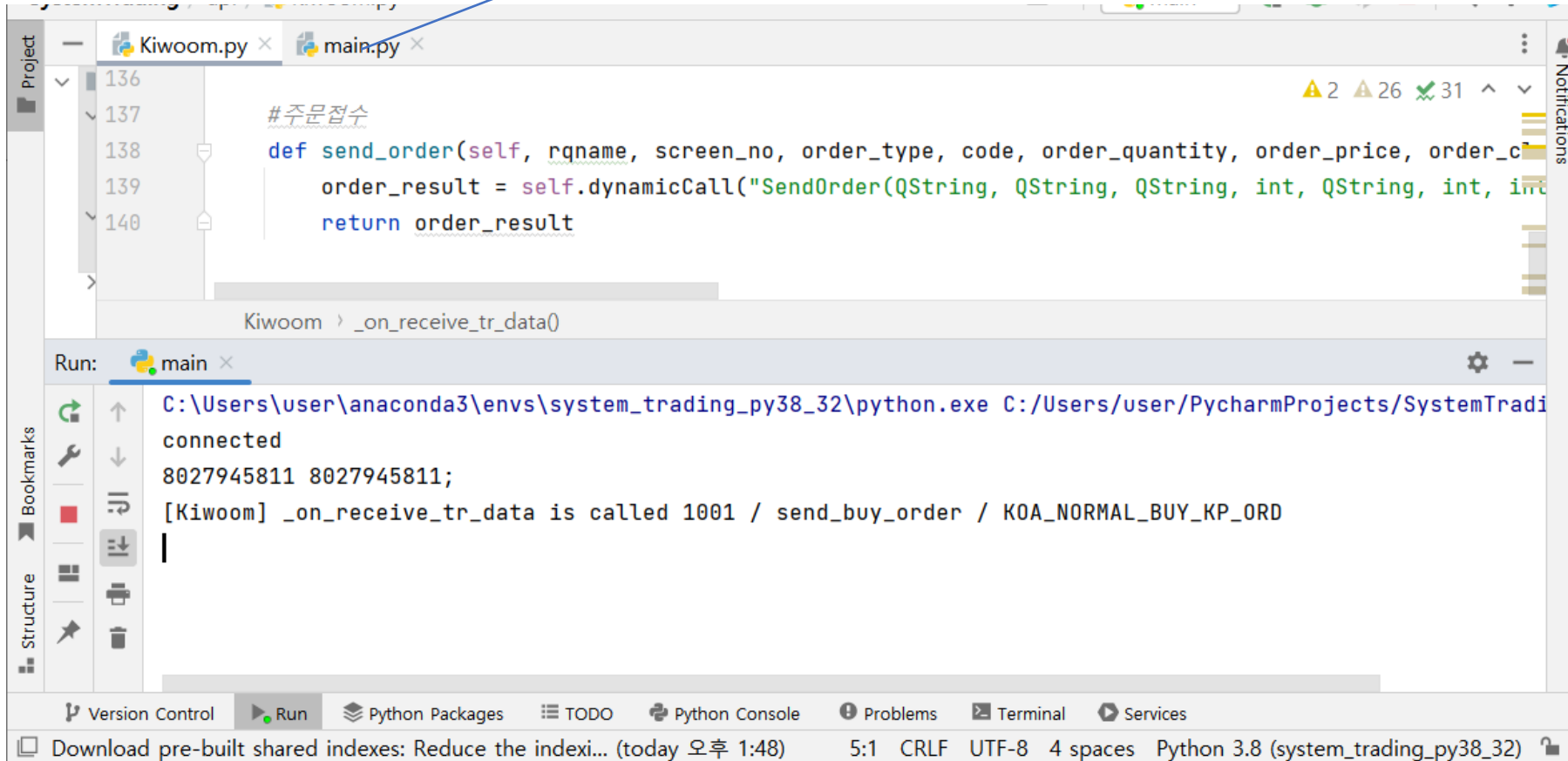
주문 상태에 따른 실시간 수신
(주문접수-체결-잔고 변동)

- 주문응답(OnReceiveTrData)이벤트

- 주문시에도 발생(정상인 경우 이벤트 내부에서 주문번호 획득, 비정상인 경우 " ")

주문접수 확인

order_result=kiwoom.send_order('send_buy_order','1001',1,'007700',1,18800,'00')



The screenshot shows the PyCharm IDE with the `Kiwoom.py` file open. The `send_order` function is defined as follows:

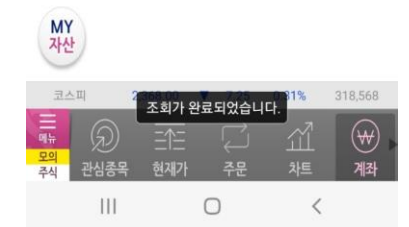
```
#주문접수
def send_order(self, rqname, screen_no, order_type, code, order_quantity, order_price, order_c
order_result = self.dynamicCall("SendOrder(QString, QString, QString, int, QString, int, int
return order_result
```

The Run console shows the following output:

```
C:\Users\user\anaconda3\envs\system_trading_py38_32\python.exe C:/Users/user/PycharmProjects/SystemTradi
connected
8027945811 8027945811;
[Kiwoom] _on_receive_tr_data is called 1001 / send_buy_order / KOA_NORMAL_BUY_KP_ORD
```



잔고	미체결	예수금	주문내역
안내	8027-9458 [위탁]	조회	
총매입	37,450	총평가	37,300
총손익	-494	총수익률	-1.32%
실현손익		용자별	용자합
종목명	매입가	평가손익	수익률
F&F홀딩스	18,725	-494	-1.32%



7. 체결 확인

- 슬롯에 주문메시지수신, 주문 접수/체결에 사용할 슬롯 함수 추가

```
# TR/주문 메시지를 _on_receive_msg를 통해 받도록 설정
self.OnReceiveMsg.connect(self._on_receive_msg)

# 주문 접수/체결 결과를 _on_chejan_slot을 통해 받도록 설정
self.OnReceiveChejanData.connect(self._on_chejan_slot)
```

[OnReceiveMsg() 이벤트]

```
OnReceiveMsg(
    BSTR sScrNo,    // 화면 번호
    BSTR sRQName,   // 사용자 구분명
    BSTR sTrCode,   // TR 이름
    BSTR sMsg       // 서버에서 전달하는 메시지
)
```

- OnReceiveMsg함수를 사용하여 등록한 _on_receive_msg함수는 TR 조회 응답 및 주문에 대한 메시지를 수신할 때 사용

```
def _on_receive_msg(self, screen_no, rqname, trcode, msg):
    print("[Kiwoom] _on_receive_msg is called {} / {} / {} / {}".format(screen_no, rqname, trcode, msg))
```

const.py: FID값 및 연결된 항목저장

- 그후 OnReceiveChejanData를 사용하여 등록한 _on_chejan_slot은 주문 접수/체결에 대한 응답을 받아 오도록 설정됨

[OnReceiveChejanData() 이벤트]

```
OnReceiveChejanData(  
    BSTR sGubun, // 체결 구분. 접수와 체결 시 '0' 값, 국내 주식 잔고 변경은 '1' 값, 파생 잔고 변경은 '4'  
    LONG nItemCnt,  
    BSTR sFidList  
)
```

주문 전송 후 주문 접수, 체결 통보, 잔고 통보를 수신할 때마다 발생합니다.
GetChejanData() 함수를 이용해서 FID 항목별 값을 얻을 수 있습니다.

- Util/onst.py 파일생성후

<https://bit.ly/376qoRJ> 접속후 내용 복사해서 붙이기

```
SystemTrading/  
├── api  
├── __init__.py  
├── Kiwoom.py  
├── util  
├── __init__.py  
├── const.py  
├── strategy  
├── __init__.py  
└── main.py
```

■ Kiwoom.py

```
from PyQt5.QAxContainer import *
....
from util.const import *

class Kiwoom(QAxWidget):
    def __init__(self):
        super().__init__()
        ...
        self.account_number = self.get_account_number()

        self.tr_event_loop = QEventLoop()

        self.order = {}
        self.balance = {}
```

```
def _on_chejan_slot(self, s_gubun, n_item_cnt, s_fid_list):
    print("[Kiwoom] _on_chejan_slot is called {} / {} / {}".format(s_gubun, n_item_cnt,
s_fid_list))

    for fid in s_fid_list.split(";"):
        if fid in FID_CODES:
            code = self.dynamicCall("GetChejanData(int)", '9001')[1:]
            data = self.dynamicCall("GetChejanData(int)", fid)
            data = data.strip().lstrip('+').lstrip('-')

            if data.isdigit():
                data = int(data)
            item_name = FID_CODES[fid]
            print("{}: {}".format(item_name, data))

            if int(s_gubun) == 0:
                if code not in self.order.keys():
                    self.order[code] = {}
                self.order[code].update({item_name: data})
            elif int(s_gubun) == 1:
                if code not in self.balance.keys():
                    self.balance[code] = {}
                self.balance[code].update({item_name: data})

    if int(s_gubun) == 0:
        print("* 주문 출력(self.order)")
        print(self.order)
    elif int(s_gubun) == 1:
        print("* 잔고 출력(self.balance)")
        print(self.balance)
```

주문완료결과

8. 주문 정보 가져오기

- 주문을 접수하거나 체결이 수행되면 order딕셔너리에 저장하여 주문 정보와 체결 상태 확인 가능(_on_chejan_slot함수가 동작하여 order에 저장)
- 프로그램 재실행시 이미 접수/체결된 주문 정보가 order에 저장되지 않음

① 프로그램 실행 > ② 주문 접수 > ③ _on_chejan_slot 함수 실행 > ④ order 데이터 저장 > ⑤ 프로그램 종료 상황 발생 > ⑥ 프로그램 재실행 > ⑦ 재주문 > ⑧ _on_chejan_slot 함수 실행 > ⑨ order 데이터 저장

→접수한 주문 자체를 인지하지 못함

- 다시 주문하는 문제 해결: 실행과 주문접수 사이에 '주문조회' 추가

① 프로그램 실행 > ② 주문 정보 조회(접수 주문 확인) > ③ 주문 접수 > ④ _on_chejan_slot 함수 실행 > ⑤ order 데이터 저장 > ⑥ 프로그램 종료 상황 발생 > ⑦ 프로그램 재실행 > ⑧ 재주문 > ⑨ _on_chejan_slot 함수 실행 > ⑩ order 데이터 저장

① 프로그램 실행 > ② 주문 정보 조회(접수 주문 확인) > ③ 주문 접수 > ④ _on_chejan_slot 함수 실행 > ⑤ order 데이터 저장 > ⑥ 프로그램 종료 상황 발생 > ⑦ 프로그램 재실행 > ⑧ 주문 정보 조회(접수 주문 확인) > ⑨ (주문 정보 확인하고 재주문하지 않음)

미체결요청

[opt10075 : 미체결 요청]

1. Open API 조회 함수 입력 값을 설정합니다.

계좌번호 = 전문 조회할 보유 계좌번호

SetInputValue("계좌번호", "입력값 1");

전체 종목 구분 = 0: 전체, 1: 종목

SetInputValue("전체종목구분", "입력값 2");

매매 구분 = 0: 전체, 1: 매도, 2: 매수

SetInputValue("매매구분", "입력값 3");

종목 코드 = 전문 조회할 종목 코드

SetInputValue("종목코드", "입력값 4");

체결 구분 = 0: 전체, 2: 체결, 1: 미체결

SetInputValue("체결구분", "입력값 5");

2. Open API 조회 함수를 호출해서 전문을 서버로 전송합니다.

CommRqData("RQName", "opt10075", "0", "화면 번호");

주문정보 (Kiwoom.py) 추가

...

def get_order(self):

self.dynamicCall("SetInputValue(QString, QString)", "계좌번호", self.account_number)

self.dynamicCall("SetInputValue(QString, QString)", "전체종목구분", "0")

0:전체, 1:미체결, 2:체결

self.dynamicCall("SetInputValue(QString, QString)", "체결구분", "0")

0:전체, 1:매도, 2:매수

self.dynamicCall("SetInputValue(QString, QString)", "매매구분", "0")

self.dynamicCall("CommRqData(QString, QString, int, QString)",
"opt10075_req", "opt10075", 0, "0002")

self.tr_event_loop.exec_()

return self.tr_data

```

...
elif rqname == "opt10075_req":
    for i in range(tr_data_cnt):
        code = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "종목코드")
        code_name = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "종목명")
        order_number = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "주문번호")
        order_status = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "주문상태")
        order_quantity = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "주문수량")
        order_price = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "주문가격")
        current_price = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "현재가")
        order_type = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "주문구분")
        left_quantity = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "미체결수량")
        executed_quantity = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "체결량")
        ordered_at = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "시간")
        fee = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "당일매매수수료")
        tax = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "당일매매세금")

        # 데이터 형변환 및 가공
        code = code.strip()
        code_name = code_name.strip()
        order_number = str(int(order_number.strip()))
        order_status = order_status.strip()
        order_quantity = int(order_quantity.strip())
        order_price = int(order_price.strip())

        current_price = int(current_price.strip().lstrip('+').lstrip('-'))
        order_type = order_type.strip().lstrip('+').lstrip('-') # +매수,-매도처럼 +,- 제거
        left_quantity = int(left_quantity.strip())
        executed_quantity = int(executed_quantity.strip())
        ordered_at = ordered_at.strip()
        fee = int(fee)
        tax = int(tax)

```



```
# code를 key값으로 한 딕셔너리 변환
self.order[code] = {
    '종목코드': code,
    '종목명': code_name,
    '주문번호': order_number,
    '주문상태': order_status,
    '주문수량': order_quantity,
    '주문가격': order_price,
    '현재가': current_price,
    '주문구분': order_type,
    '미체결수량': left_quantity,
    '체결량': executed_quantity,
    '주문시간': ordered_at,
    '당일매매수수료': fee,
    '당일매매세금': tax
}

self.tr_data = self.order
```

#main.py 파일

```
orders=kiwoom.get_order()
Print(orders)
```

결과

9. 잔고

- 현재 보유중인 종목
- '주문접수 및 체결'에서 주문 체결 이후 주식이 '잔고'로 이동
- TR(opw00018:계좌 평가잔고내역요청)조회 사용

[opw00018: 계좌 평가 잔고 내역 요청]

[주의]

"수익률%" 데이터는 모의투자에서는 소수점 표현, 실거래 서버에서는 소숫점으로 변환 필요합니다.

1. Open API 조회 함수 입력 값을 설정합니다.

계좌번호 = 전문 조회할 보유 계좌번호

SetInputValue("계좌번호", "입력값 1");

비밀번호 = 사용 안 함(공백)

SetInputValue("비밀번호", "입력값 2");

비밀번호 입력 매체 구분 = 00

SetInputValue("비밀번호입력매체구분", "입력값 3");

조회 구분 = 1: 합산, 2: 개별

SetInputValue("조회구분", "입력값 4");

2. Open API 조회 함수를 호출해서 전문을 서버로 전송합니다.

CommRqData("RQName", "opw00018", "0", "화면 번호")

```
def get_balance(self):
    self.dynamicCall("SetInputValue(QString, QString)", "계좌번호", self.account_number)
    self.dynamicCall("SetInputValue(QString, QString)", "비밀번호입력매체구분", "00")
    self.dynamicCall("SetInputValue(QString, QString)", "조회구분", "1")
    self.dynamicCall("CommRqData(QString, QString, int, QString)", "opw00018_req", "opw00018", 0, "0002")

    self.tr_event_loop.exec_()
    return self.tr_data
```

■ 필요한 정보

- 종목번호, 종목명, 보유수량, 매입가, 수익률, 현재가, 매입금액, 매매가능수량

```

elif rqname == "opw00018_req":
    for i in range(tr_data_cnt):
        code = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "종목번호")
        code_name = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "종목명")
        quantity = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "보유수량")
        purchase_price = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "매입가")
        return_rate = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "수익률(%)"")
        current_price = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "현재가")
        total_purchase_price = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "매입금액")
        available_quantity = self.dynamicCall("GetCommData(QString, QString, int, QString", trcode, rqname, i, "매매가능수량")

        # 데이터 형변환 및 가공
        code = code.strip()[1:]
        code_name = code_name.strip()
        quantity = int(quantity)
        purchase_price = int(purchase_price)
        return_rate = float(return_rate)
        current_price = int(current_price)
        total_purchase_price = int(total_purchase_price)
        available_quantity = int(available_quantity)

        # code를 key값으로 한 딕셔너리 변환
        self.balance[code] = {
            '종목명': code_name,
            '보유수량': quantity,
            '매입가': purchase_price,
            '수익률': return_rate,
            '현재가': current_price,
            '매입금액': total_purchase_price,
            '매매가능수량': available_quantity
        }

    self.tr_data = self.balance

```

```

main.py
position=kiwoom.get_balance()
print(position)

```

10. 실시간 체결 정보 가져오기

■ 실시간 체결정보란

- 체결가, 매수 호가, 매도 호가 등 체결이 될 때마다 발생하는 정보를 의미
- TR(요청>응답대기>응답방식)을 이용하여 호출하고 응답하는 동안에서 데이터의 변화가 생김

➔ 실시간 시세 사용법 참고(KOA)

[실시간 시세 사용법]

시세 관련 TR 서비스를 요청하는 경우 자동으로 서버에 해당 종목의 실시간 시세가 등록됩니다.

시세 관련 TR 서비스 조회 요청이 불필요한 경우 SetRealReg 함수를 통해 조회 없이 실시간 시세 등록이 가능합니다.

방법1. 조회 서비스 이용

SetInputValue(사용자 호출) → CommRqData(사용자 호출) → OnReceiveTrData(이벤트 발생) → OnReceiveRealData(이벤트 발생)

방법2. SetRealReg 함수 이용

SetRealReg(사용자 호출) → OnReceiveRealData(이벤트 발생)

[SetRealReg() 함수]

```
SetRealReg(  
    BSTR strScreenNo, // 화면 번호  
    BSTR strCodeList, // 종목 코드 리스트  
    BSTR strFidList,   // 실시간 FID 리스트  
    BSTR strOptType    // 실시간 등록 타입, 0 또는 1  
)
```

[실시간 시세 등록 예시]

```
OpenAPI.SetRealReg(_T("0150"), _T("039490"), _T("9001;302;10;11;25;12;13"), "0");  
// 039490 종목만 실시간 등록  
OpenAPI.SetRealReg(_T("0150"), _T("000660"), _T("9001;302;10;11;25;12;13"), "1");  
// 000660 종목을 실시간 추가 등록
```

SetRealReg호출하는 함수

```
def set_real_reg(self, str_screen_no, str_code_list, str_fid_list, str_opt_type):  
    self.dynamicCall("SetRealReg(QString, QString, QString, QString)", str_screen_no, str_code_list, str_fid_list, str_opt_type)  
    time.sleep(0.5)
```

```
class Kiwoom(QAxWidget):  
    def __init__(self):  
        ...  
        self.universe_realtime_transaction_info = {}
```

→ 실시간 체결 정보를 저장할 딕셔너리 선언

```

def _on_receive_real_data(self, s_code, real_type, real_data):
    if real_type == "장시작시간":
        pass

    elif real_type == "주식체결":
        signed_at = self.dynamicCall("GetCommRealData(QString, int)", s_code, get_fid("체결시간"))

        close = self.dynamicCall("GetCommRealData(QString, int)", s_code, get_fid("현재가"))
        close = abs(int(close))

        high = self.dynamicCall("GetCommRealData(QString, int)", s_code, get_fid('고가'))
        high = abs(int(high))
        open = self.dynamicCall("GetCommRealData(QString, int)", s_code, get_fid('시가'))
        open = abs(int(open))
        low = self.dynamicCall("GetCommRealData(QString, int)", s_code, get_fid('저가'))
        low = abs(int(low))
        top_priority_ask = self.dynamicCall("GetCommRealData(QString, int)", s_code, get_fid('(최우선)매도호가'))
        top_priority_ask = abs(int(top_priority_ask))
        top_priority_bid = self.dynamicCall("GetCommRealData(QString, int)", s_code, get_fid('(최우선)매수호가'))
        top_priority_bid = abs(int(top_priority_bid))

        ccum_volume = self.dynamicCall("GetCommRealData(QString, int)", s_code, get_fid('누적거래량'))
        accum_volume = abs(int(accum_volume))

        print(s_code, signed_at, close, high, open, low, top_priority_ask, top_priority_bid, accum_volume)
        if s_code not in self.universe_realtime_transaction_info:
            self.universe_realtime_transaction_info.update({s_code: {}})

        # 최초 수신 이후 계속 수신되는 데이터는 update를 이용해서 값 갱신
        self.universe_realtime_transaction_info[s_code].update({
            "체결시간": signed_at,
            "시가": open,
            "고가": high,
            "저가": low,
            "현재가": close,
            "(최우선)매도호가": top_priority_ask,
            "(최우선)매수호가": top_priority_bid,
            "누적거래량": accum_volume
        })

```


const.py에 작성

```
# 찾고자 하는 항목명의 FID를 찾아주는 함수
def get_fid(search_value):
    """
    사용예
    fid = get_fid("고가")
    # fid = 17
    """
    keys = [key for key, value in FID_CODES.items() if value == search_value]
    return keys[0]
```

Kiwoon.py에 슬롯함수에 추가

```
# 실시간 체결 데이터를 _on_receive_real_data을 통해 받도록 설정
self.OnReceiveRealData.connect(self._on_receive_real_data)
```

main.py에 슬롯 함수에 추가

```
kiwoom.set_read_reg("1000","",get_fid("장운영구분"),0)
fids=get_fid("체결시간")
codes='005930;007700;000660;'
Kiwoom.set_real_req("1000",codes,fids,"0")
```

최초등록(0)인지 아닌지(1)를 구분
➔여기서는 최초등록이 되므로"0"전달