컴퓨터의 개념 및 실습

Practice 2

서울대학교 산업공학과

남유리 Nam, Youlee

과제 관련 유의사항

- 뼈대코드 내에서 주석 사이 pass를 지우고 코드 구현

```
##############################

pass

##############################
```

- 과제와 관련된 질문은 **eTL Q&A 게시판**을 통해서만 받음
  - eTL > 게시판 > Q&A 게시판
  - 개인 이메일 질문에는 답변 X
- 어떠한 사유로든 **지각 제출은 받지 않음 (Due: 11/11(월) 오후 11시 59분)**
  - 과제는 여러 번 제출 가능하며 채점은 마지막 제출물로 평가
  - 마감 전 여유를 두고 과제 제출할 것을 권장
  - eTL 제출 에러 발생시 에러 증빙을 위한 스크린샷과 과제 제출물을 마감시간 이전까지 email로 전송한 경우에만 인정
- **제출 파일 형식** 준수
  - 제출 파일 : 작성한 2개의 파일(stock_trading_fp.py, stock_trading_oop.py)
  - 총 2개의 파일을 "dccp_이름_practice2" 폴더에 저장한 뒤, 해당 폴더를 압축하여 하나의 zip파일로 제출
  - eTL 과제 > 수시평가 > Practice2에 반드시 하나의 압축파일로 업로드
- Copy / 정보공유 / 표절 주의 (예외없이 0점 처리)
  - 참고한 코드를 그대로 사용한 경우 표절로 간주되니 주의할 것

(개요) 주식 트레이딩 시뮬레이션

- 문제 정의
  - 여러 사용자가 각자의 포트폴리오를 가지고 서로 다른 전략으로 주식 트레이딩 시뮬레이션을 수행하여 그 결과를 비교하는 문제를 해결하고자 한다. 시뮬레이션은 특정 기간의 실제 시장 데이터를 기반으로 하며, 각 사용자는 서로 다른 거래 전략을 사용해 주식을 매수하거나 매도한다. 각 사용자는 자신의 포트폴리오(현금 및 보유 주식)를 관리하며, 시뮬레이션 종료 후 사용자의 최종 포트폴리오 가치를 비교하여 어떤 전략이 가장 효과적이었는지 평가한다.
  - 주요 요구사항
    - 여러 사용자가 개별 포트폴리오를 가지고 있으며, 각 사용자는 서로 다른 매수/매도 전략을 사용한다.
    - 매수 및 매도 조건은 각 전략에 따라 다르게 설정된다. 조건에는 주가 변화 비율과 최대 매수/매도 수량이 포함된다.
    - 시뮬레이션이 진행되는 동안 각 사용자는 자신의 포트폴리오를 업데이트하며 거래를 수행한다.
    - 최종 포트폴리오 가치를 기준으로 각 사용자의 전략 성과를 비교한다.
  - 주어진 문제를 함수형 프로그래밍과 객체 지향 프로그래밍 두가지 방식으로 각각 구현해본다.

뼈대코드 및 파일 개요

- stock_data.txt : 주가 데이터 파일
  - 시뮬레이션에 사용되는 주식 시장 데이터는 텍스트 파일 형식으로 주어진다.
  - 파일의 각 줄은 날짜와 주식 종목의 가격(종가) 정보를 포함한다.
  - 각 줄에는 "날짜, 주식1의 가격, 주식2의 가격"이 쉼표로 구분되어 있다.
    - 첫 번째 값: 날짜 (형식: YYYY-MM-DD)
    - 두 번째 값: AAPL (애플 주식)의 가격
    - 세 번째 값: GOOGL (구글 주식)의 가격
- stock_trading_fp.py : 주식 트레이딩을 함수 기반으로 구현
  - 함수형 프로그래밍 방식으로 각 사용자에 대해 개별 포트폴리오를 관리하고, 서로 다른 거래 전략을 적용해 주식 투자 시뮬레이션을 실행한다.
- stock_trading_oop.py : 주식 트레이딩을 객체 지향 프로그래밍 방식으로 구현
  - 객체 지향 프로그래밍 방식으로 필요한 클래스를 정의하고, 이를 기반으로 사용자가 개별 포트폴리오와 거래 전략을 관리하며 주식 투자 시뮬레이션을 실행한다.
- 두 모듈의 실행 결과는 동일하며, 14 페이지 출력 예시를 참고하여 출력 형식을 지정할 것

1. stock_trading_fp.py

- **1-1. initialize_portfolio(initial_cash)** *(뼈대코드에 이미 구현되어 있음)*
  - 초기 포트폴리오를 설정. 현금(cash)과 보유 주식(holdings) 정보를 초기화
  - 파라미터
    - initial_cash: 사용자가 보유하고 있는 초기 현금 자산 (float)
  - 구현할 기능
    - 포트폴리오 딕셔너리(key는 cash, holdings이며 value는 대응되는 초기값)를 생성
    - cash: 초기 현금으로 설정
    - holdings: AAPL과 GOOGL 보유 주식을 각각 0주로 설정. holdings는 주식 종목을 key, 보유 수량을 value로 갖는 딕셔너리
  - 반환값
    - 포트폴리오 초기 상태 (dict)
- **1-2. read_market_data(file_name)**
  - 텍스트 파일에서 주식 시장 데이터를 읽어옴
  - 날짜와 주식 종목별 가격이 포함된 딕셔너리의 리스트 형태로 데이터를 저장하고 반환
  - 파라미터
    - file_name: 읽어들일 텍스트 파일 경로 또는 이름 (str)
  - 구현할 기능
    - 파일을 읽고 각 줄을 쉼표(,)로 분리하여 날짜, AAPL 주가, GOOGL 주가를 추출
    - 리스트: 주가 데이터 리스트는 파일의 각 줄을 하나의 데이터 포인트로 저장. 각 날짜에 해당하는 데이터를 딕셔너리로 구성하여 리스트에 추가
    - 딕셔너리: 리스트의 각 아이템은 딕셔너리로 key는 date, AAPL, GOOGL로 구성되며, value는 대응되는 날짜(str), AAPL 주식 가격(float), GOOGL 주식 가격(float). 이때 주식 가격은 소수점 둘째 자리까지의 실수값으로 저장
  - 반환값
    - 일자별 주가 정보 딕셔너리의 리스트

1. stock_trading_fp.py

- **1-3. buy_stock(portfolio, stock, price, quantity)**
  - 특정 주식을 매수하여 포트폴리오 상태를 업데이트
  - 파라미터
    - portfolio: 현재 포트폴리오 상태 (dict)
    - stock: 매수할 주식의 종목 이름 (str) 예: 'AAPL'
    - price: 매수 가격 (float)
    - quantity: 매수할 주식 수량 (int)
  - 구현할 기능
    - 잔고가 충분하면, 해당 주식을 매수하고 현금 잔고를 차감하며 보유 주식을 증가시킴
    - 매수 후 포트폴리오의 새로운 상태와 결과 메시지를 반환
    - 잔고가 부족할 경우, 매수하지 않고 포트폴리오 상태와 매수 불가 메시지 반환
  - 반환값
    - 매수 후 업데이트된 포트폴리오 (dict), 매수 결과 메시지 (str)
- **1-4. sell_stock(portfolio, stock, price, quantity)**
  - 특정 주식을 매도하여 포트폴리오 상태를 업데이트
  - 파라미터
    - portfolio: 현재 포트폴리오 상태 (dict)
    - stock: 매도할 주식의 종목 이름 (str) 예: 'AAPL'
    - price: 매도 가격 (float)
    - quantity: 매도할 주식 수량 (int)
  - 구현할 기능
    - 보유 주식 수량이 충분하면, 해당 주식을 매도하고 현금 잔고를 증가시키며 보유 주식 수량을 감소시킴
    - 매도 후 포트폴리오의 새로운 상태와 결과 메시지를 반환
    - 보유 주식이 부족할 경우, 매도하지 않고 포트폴리오 상태와 매도 불가 메시지 반환
  - 반환값
    - 매도 후 업데이트된 포트폴리오 (dict), 매도 결과 메시지 (str)

1. stock_trading_fp.py

- **1-5. calculate_portfolio_value(portfolio, current_price)**
  - 현재 포트폴리오의 총 가치를 계산
  - 파라미터
    - portfolio: 현재 포트폴리오 상태 (dict)
    - current_price: 현재 주식 가격 (dict) 예: {'AAPL': 180.00, 'GOOGL': 140.00}
  - 구현할 기능
    - 현금 잔고와 보유 주식의 현재 가치를 모두 합산하여 포트폴리오의 총 가치를 계산
  - 반환값
    - 총 포트폴리오 가치 (float)
- **1-6. apply_trading_strategy(user, current_price, price_history)**
  - 사용자의 거래 전략을 현재 시장 상황에 적용
  - 파라미터
    - user: 사용자 정보 (dict)
    - current_price: 현재 주가 (dict)
    - price_history: 주가 히스토리 (dict) 예: {'AAPL': [170.03, 168.84, 169.65, 168.82, 169.58], 'GOOGL': [156.5, 155.87, 156.37, 151.94, 153.94]}
  - 구현할 기능
    - 과거 5일간의 주가 히스토리를 바탕으로 평균 가격을 계산하고, 이를 현재 가격과 비교하여 매수/매도 조건에 맞을 경우 거래를 수행
      - 5일간 주가 평균 대비 현재 가격이 매수 임계값 비율 이하로 감소하면 매수
      - 5일간 주가 평균 대비 현재 가격이 매도 임계값 비율 이상으로 증가하면 매도
    - 최대 구매 가능 수량과 보유 수량에 따라 매수/매도 수량을 결정하여 거래 수행
      - 매수할 수 있는 최대 수량은 사용자의 포트폴리오에 있는 현금 잔액을 현재 주가로 나눈 몫으로 계산하며, 실제로 매수할 수량은 매수 가능한 최대 수량과 전략에 정의된 최대 매수 수량(max_buy_quantity) 중 더 작은 값으로 결정
      - 매도할 최대 수량은 현재 보유하고 있는 주식 수량과 전략에서 정의한 최대 매도 수량(max_sell_quantity) 중 더 작은 값으로 결정
    - 만약 과거 5일치의 가격 데이터가 없을 경우 거래를 수행하지 않음
    - 포트폴리오 상태를 업데이트하고, 거래 결과 메시지를 출력
  - 반환값
    - 전략 적용 후 포트폴리오가 업데이트된 사용자 정보 (dict)

1. stock_trading_fp.py

- **1-7. update_price_history(price_history, current_price)**
  - 주가 히스토리 업데이트
  - 파라미터
    - price_history: 주가 히스토리 (dict)
    - current_price: 현재 주가 (dict)
  - 구현할 기능
    - 각 주식의 가격 히스토리를 업데이트하며, 최대 5일의 최근 가격 히스토리 유지
  - 반환값
    - 업데이트된 주가 히스토리 (dict)
- **1-8. run_simulation(file_path, users)**
  - 사용자들의 거래 전략으로 주식 투자 시뮬레이션을 실행하고 결과를 출력
  - 파라미터
    - file_path: 주식 시장 데이터 파일 경로 (str)
    - users: 사용자 리스트 (각 사용자는 딕셔너리 형태)
  - 구현할 기능
    - 주식 시장 데이터를 읽어와 각 날짜별로 사용자 거래 전략 실행
    - 주가 히스토리는 공통으로 업데이트하고 이를 각 사용자가 참조
    - 트레이딩 결과 각 사용자의 포트폴리오 상태를 업데이트하고, 최종 포트폴리오 가치를 계산하여 출력
  - 반환값
    - 없음

2. stock_trading_oop.py

- 명시되지 않은 세부 구현 내용은 1번에서 제시한 것과 동일하므로 1번 문제 참조
- **2-1. StockMarket 클래스**
  - StockMarket 클래스는 주식 시장 데이터를 관리하고 매 거래일의 가격 정보 제공
  - attributes

attribute 이름	타입	default 값	설명
market_data	list		주식의 날짜별 가격 정보를 담고 있는 딕셔너리의 리스트. 각 딕셔너리는 날짜와 주식 종목별 가격으로 구성
current_day	int	0	현재 시뮬레이션이 진행된 일수. 시장 데이터에서 현재 위치를 나타냄
price_history	dict	{'AAPL':[], 'GOOGL': []}	주가 히스토리를 관리. Key는 주식 종목명 value는 과거 5일의 가격 히스토리 리스트

  - methods

method이름	파라미터	type	수행 기능	설명
__init__	file_path	str		주어진 파일 경로에서 주식 시장 데이터를 읽어오고 초기화
read_market_data	file_path	list	파일을 읽고 각 줄을 쉼표로 구분하여 날짜, AAPL 주가, GOOGL 주가 추출. 일자별 주가 정보 딕셔너리의 리스트로 저장하고 반환	파일을 읽어 주식 시장 데이터를 parsing하여 리스트에 저장
update_price			과거 5일의 주가 히스토리를 유지하며 현재 가격 정보를 반환. 시뮬레이션 일수를 증가시키며, 더 이상 데이터가 없을 경우 None 반환	시뮬레이션의 가격 정보 업데이트

2. stock_trading_oop.py

- **2-2. Portfolio 클래스**
  - Portfolio 클래스는 사용자의 자산 포트폴리오를 관리
  - attributes

attribute 이름	타입	default 값	설명
cash	float		사용자의 보유 현금
holdings	dict	{'AAPL': 0, 'GOOGL': 0}	각 주식 종목에 대한 보유 수량

  - methods

method이름	파라미터	type	수행 기능	설명
__init__	initial_cash	float		초기 현금 자산을 설정하고 보유 주식 수량을 초기화
buy	stock	str	현금이 충분할 경우에 매수를 진행하여 현금 잔고와 보유 주식을 업데이트하고 매수 정보 출력. 그렇지 않은 경우 매수 불가 메시지 출력	주어진 가격, 수량만큼 주식을 매수
	price	float		
	quantity	int		
Sell	stock	str	보유 수량이 충분할 경우에 매도를 진행하여 현금 잔고와 보유 주식을 업데이트하고 매도 정보 출력. 그렇지 않은 경우 매도 불가 메시지 출력	주어진 가격, 수량만큼 주식을 매도
	price	float		
	quantity	int		
calculate_value	current_price	dict	현금 자산과 보유 주식의 현재 가치를 합산하여 반환	포트폴리오 총 가치 계산

2. stock_trading_oop.py

- **2-3. TradingStrategy 클래스**
  - TradingStrategy 클래스는 사용자의 매수/매도 전략을 정의
  - attributes

attribute 이름	타입	default 값	설명
buy_threshold	float		매수 임계값 비율. 과거 5일 주가 평균 대비 현재 가격의 비율
sell_threshold	float		매도 임계값 비율. 과거 5일 주가 평균 대비 현재 가격의 비율
max_buy_quantity	int		최대 매수 가능한 수량
max_sell_quantity	int		최대 매도 가능한 수량

  - methods

method이름	파라미터	type	수행 기능	설명
__init__	buy_threshold	float		매수/매도 임계값 비율과 최대 매수/매도 수량을 기반으로 거래 전략 초기화
	sell_threshold	float		
	max_buy_quantity	int		
	max_sell_quantity	int		
apply_strategy	portfolio	Portfolio	- 과거 5일간 주가 평균 대비 현재 가격이 매수 임계값 비율 이하로 감소하면 매수 - 과거 5일간 주가 평균 대비 현재 가격이 매도 임계값 비율 이상으로 증가하면 매도 - 최대 구매 가능 수량과 보유 수량에 따라 매수/매도 수량을 결정하여 거래 수행 - 만약 과거 5일치 가격 데이터가 없을 경우 거래를 수행하지 않음 - 포트폴리오의 buy/sell 메서드 사용	현재 주가와 주가 히스토리가 주어졌을 때, 임계값 비율을 기준으로 최대 거래 수량을 고려하여 거래 전략 수행
	stock	str		
	current_price	float		
	price_history	list		

2. stock_trading_oop.py

- **2-4. User 클래스**
  - User 클래스는 사용자 정보를 관리하고 거래 전략을 수행
  - Attributes

attribute 이름	타입	default 값	설명
name	str		사용자 이름
portfolio	Portfolio		사용자의 포트폴리오 객체
strategy	TradingStrategy		사용자가 사용하는 거래 전략 객체

  - methods

method이름	파라미터	Type	수행 기능	설명
__init__	name	str		사용자를 초기화하고 포트폴리오와 거래 전략 설정
	initial_cash	float		
	strategy	TradingStrategy		
run_strategy	current_price	dict	사용자 거래 전략의 apply_strategy 사용	현재 주가와 과거 히스토리를 바탕으로 거래 전략 수행
	price_history	dict		
calculate_portfolio_value	current_price	dict	사용자 포트폴리오의 calculate_value 사용	현재 주가 정보를 이용해 사용자 포트폴리오의 총 가치를 계산하여 반환

2. stock_trading_oop.py

- **2-5. Simulation 클래스**
  - Simulation 클래스는 여러 사용자들의 거래 전략을 수행하고 최종 포트폴리오 가치를 비교
  - attributes

attribute 이름	타입	default 값	설명
users	list of User		시뮬레이션에 참여하는 사용자 객체 목록
market	StockMarket		주식 시장 데이터를 담고있는 객체

  - methods

method이름	파라미터	type	수행 기능	설명
__init__	users	list of User		시뮬레이션 초기화
	market	StockMarket		
run			매 거래일마다 각 사용자에 대해 거래 전략을 실행하고 트레이딩 결과 각 사용자의 포트폴리오 상태를 업데이트하고, 최종 포트폴리오 가치를 계산하여 출력	시뮬레이션을 실행

실행결과 출력 예시

- 일자별로 각 사용자의 거래 수행 결과 및 포트폴리오 정보 출력

```
Date: 2024-03-05
User Alice:
Bought 10 shares of AAPL at $170.12
Portfolio Value = $9830.50, Cash = $5453.70, Holdings: AAPL = 10 shares, GOOGL = 20 shares
User Bob:
Portfolio Value = $9348.85, Cash = $108.05, Holdings: AAPL = 15 shares, GOOGL = 50 shares
User Charlie:
Bought 5 shares of AAPL at $170.12
Portfolio Value = $9915.25, Cash = $7726.85, Holdings: AAPL = 5 shares, GOOGL = 10 shares
```

```
Date: 2024-03-18
User Alice:
Sold 5 shares of GOOGL at $148.48
Portfolio Value = $10160.50, Cash = $6196.10, Holdings: AAPL = 10 shares, GOOGL = 15 shares
User Bob:
Portfolio Value = $10137.85, Cash = $108.05, Holdings: AAPL = 15 shares, GOOGL = 50 shares
User Charlie:
Portfolio Value = $10080.25, Cash = $7726.85, Holdings: AAPL = 5 shares, GOOGL = 10 shares
```

- 최종 포트폴리오 가치 비교

```
Final Results:
User Alice: Final Portfolio Value = $10222.70
User Bob: Final Portfolio Value = $10441.80
User Charlie: Final Portfolio Value = $10130.50
```