python #5

▼ 문제

한 스타트업이 온라인 음료 주문 플랫폼을 개발 중이다.
사용자의 주문 이력을 기반으로 머신러닝 없이도 간단한 추천 시스템을 포함하려 한다.

이 시스템은 다음 기능을 포함해야 한다.
- 다양한 음료 메뉴 정의 (이름, 가격, 태그: 예: '커피', '차', '콜드', '뜨거운')
- 사용자의 주문 내역 저장 (사용자는 임의로 설정 후 진행)
- 최근 주문된 음료를 바탕으로 유사한 태그의 음료 추천
- 총 주문 금액 계산 (총합, 평균 등 포함)

```
# 메뉴 예시
menu = [
  Beverage("아이스 아메리카노", 3000, ["커피", "콜드"]),
  Beverage("카페라떼", 3500, ["커피", "밀크"]),
  Beverage("녹차", 2800, ["차", "뜨거운"]),
  Beverage("허브티", 3000, ["차", "차가운"]),
]
```

프로그램 코드와 실행 결과를 화면 Capture 하셔서 댓글로 달아주세요.
(10시 50분까지 진행해주세요.)

▼ 코드

▼ 전체 코드

```
from dataclasses import dataclass
from typing import List

# Beverage 클래스: 음료 데이터 정의
@dataclass
class Beverage:
    name: str # 음료 이름 속성
    price: float # 가격 속성
    tags: List[str] # 분류 태그 속성

# Order 클래스: 주문 내역 저장
class Order:
    __slots__ = ("beverage", "quantity") # 인스턴스 속성을 beverage와 quantity로 고정하여 메모리 절약
    def __init__(self, beverage: Beverage, quantity: int) -> None: # beverage 타입 명시(Beverage), quantit
y 타입 명시 (int)
        self.beverage = beverage # self 객체에 beverage 객체 저장
        self.quantity = quantity # 주문 수량 quantity 저장
    @property
    def total_price(self) -> float: # total_price 프로퍼티: 주문금액 = 음료 가격 × 수량 자동 계산
        return self.beverage.price * self.quantity

# User 클래스: 사용자 정보와 주문 내역 관리
class User:
    def __init__(self, name: str) -> None: # name 타입 명시(str)
        self.name = name # self 객체에 주문자 이름 name 저장
        self.orders: List[Order] = [] # 비어있는 주문 목록 orders 저장
```

```
    def add_order(self, order: Order) -> None: # order 타입 명시(Order)
        self.orders.append(order) # 주문 목록(self.orders)에 새로운 Order 객체를 추가
    def get_total_spent(self) -> float: # 반환 타입 명시 (float여야함)
        return sum(order.total_price for order in self.orders) # 총 지출 금액 계산
    def get_recent_tags(self, n: int = 3) -> List[str]: # 기본값이 3, 미지정시 최근 3개의 주문을 기준으로 태그 생
성
        tags: List[str] = []
        for order in self.orders[-n:]: # 최근 n개의 주문을 확인
            tags.extend(order.beverage.tags) # 그 음료의 태그를 전부 모음
        return list(set(tags)) # 중복 제거 후 리스트로 변환

# RecommendationEngine 클래스: 태그 기반 음료 추천 기능
class RecommendationEngine:
    def __init__(self, menu: List[Beverage]) -> None: # menu 타입 명시 (Beverage 객체들이 들어 있는 리스트)
        self.menu = menu # menu 리스트 저장
    def recommend(self, user: User) -> List[Beverage]: # user 타입 명시(User), 반환 타입 명시 (Beverage
객체들이 들어 있는 리스트)
        recent_tags = user.get_recent_tags() # 사용자의 최근 태그 목록
        recommendations: List[Beverage] = [] # 추천 음료를 저장할 리스트 recommendations 초기화
        for beverage in self.menu:
            if any(tag in beverage.tags for tag in recent_tags): # 태그가 겹치는 음료의 경우
                recommendations.append(beverage) # 추천 목록 리스트에 추가
        return recommendations
```

```
# 음료 메뉴 정의
menu = [
    Beverage("아이스 아메리카노", 3000, ["커피", "콜드"]),
    Beverage("카페라떼", 3500, ["커피", "밀크", "웜"]),
    Beverage("녹차", 2800, ["차", "핫", "머그"]),
    Beverage("헤이즐넛티", 3000, ["차", "머그"]),
    Beverage("콜드브루", 4000, ["커피", "콜드", "진한"]),
    Beverage("바닐라 라떼", 3800, ["커피", "밀크", "달콤"]),
    Beverage("카라멜 마끼아또", 4200, ["커피", "밀크", "달콤"]),
    Beverage("홍차", 2700, ["차", "핫"]),
    Beverage("페퍼민트티", 2900, ["차", "허브"]),
    Beverage("레몬티", 3200, ["차", "과일"]),
    Beverage("망고 스무디", 4500, ["스무디", "과일", "차가운"]),
    Beverage("딸기 스무디", 4500, ["스무디", "과일", "차가운"]),
    Beverage("초코 프라푸치노", 5000, ["커피", "초코", "차가운"]),
    Beverage("그린티 프라푸치노", 5000, ["차", "차가운", "밀크"]),
    Beverage("핫초코", 3500, ["초코", "핫", "밀크"]),
    Beverage("얼그레이 라떼", 3700,["차", "밀크", "웜"]),
    Beverage("허니 자몽티", 3900, ["차", "과일", "달콤"]),
    Beverage("아포가토", 4800, ["커피", "아이스크림", "디저트"]),
    Beverage("플랫화이트", 3600, ["커피", "밀크"]),
    Beverage("마키아토", 3300, ["커피", "진한"]),
]

# 사용자의 주문 내역
user = User("철수")
user.add_order(Order(menu[0], 1)) # 아메리카노 1잔 주문
user.add_order(Order(menu[1], 2)) # 카페라떼 2잔 주문

# 최근 주문된 음료를 바탕으로 유사한 태그의 음료 추천
```

```
recommender = RecommendationEngine(menu) # 추천 엔진 생성
recommended = recommender.recommend(user) # 최근 태그를 기반으로 음료 추천 목록 생성

# 추천 음료 출력
print("추천 음료:")
for b in recommended:
    print(f"- {b.name} (tags: {b.tags})") # 추천 음료 목록 출력

# 총 주문 금액 계산
print()
print(f"총 주문 금액: {user.get_total_spent():,.0f}원") # 총 주문 금액 출력
```

▼ Beverage 클래스

```
@dataclass
class Beverage: # Beverage 클래스: 음료 데이터 정의
    name: str # 음료 이름 속성
    price: float # 가격 속성
    tags: List[str] # 분류 태그 속성
```

- Beverage 클래스
  - @dataclass로 음료 데이터 정의 (__init__, __repr__, __eq__ 등이 자동 생성)
- 속성 정의
  - name: 음료 이름(str), price: 가격(float), tags: 분류 태그(str list)

▼ Order 클래스

```
class Order: # Order 클래스: 주문 내역 저장
    __slots__ = ("beverage", "quantity") # 인스턴스 속성을 beverage와 quantity로 고정하여 메모리 절약
    def __init__(self, beverage: Beverage, quantity: int) -> None: # beverage 타입 명시(Beverage 타입 객체여야 함), quantity 타입 명시 (int여야함)
        self.beverage = beverage # self 객체에 beverage 객체 저장
        self.quantity = quantity # 주문 수량 quantity 저장
    @property
    def total_price(self) -> float: # total_price 프로퍼티: 주문금액 = 음료 가격 × 수량 자동 계산
        return self.beverage.price * self.quantity
```

- Order 클래스
  - 주문 내역 저장
- __slots__
  - 인스턴스 속성을 고정해 메모리 절약: "beverage"와 "quantity" 두 개만 가질 수 있음.
- __init__
  - __init__(self, beverage: Beverage, quantity: int)
    - self → 만들어질 객체 자신
    - beverage: Beverage → beverage라는 매개변수를 받는데, Beverage 타입 객체여야 한다는 타입 명시
    - quantity: int → 주문 수량을 나타내는 매개변수인데 정수(int) 여야 함을 명시
  - self.name = name: self 객체에 음료이름 beverage 저장
  - self.orders = []: 비어있는 주문 목록 quantity 저장
- total_price

- 주문금액 = 음료 가격 × 수량 자동 계산

▼ User 클래스

```
class User: # User 클래스: 사용자 정보와 주문 내역 관리
    def __init__(self, name: str) -> None: # name 타입 명시(str)
        self.name = name # self 객체에 주문자 이름 name 저장
        self.orders: List[Order] = [] # 비어있는 주문 목록 orders 저장
    def add_order(self, order: Order) -> None: # order 타입 명시(Order)
        self.orders.append(order) # 주문 목록(self.orders)에 새로운 Order 객체 추가
    def get_total_spent(self) -> float: # 반환 타입 명시 (float)
        return sum(order.total_price for order in self.orders) # 총 지출 금액 계산
    def get_recent_tags(self, n: int = 3) -> List[str]: # 기본값이 3, 미지정시 최근 3개의 주문을 기준으로 태그 생
성
        tags: List[str] = []
        for order in self.orders[-n:]: # 최근 n개의 주문을 확인
            tags.extend(order.beverage.tags) # 그 음료의 태그를 전부 모음
        return list(set(tags)) # 중복 제거 후 리스트로 변환
```

- __init__
  - self, name: str →name 타입 명시(str)
  - self.name →주문자 이름 name 저장
  - self.orders →비어있는 주문 목록 orders 저장
- add_order
  - order: Order → order 타입 명시(Order)
  - 주문 목록(self.orders)에 새로운 Order 객체 추가
- get_total_spent
  - (self) → float: 반환 타입 명시 (float)
  - order.total_price로 지출 금액 계산
- get_recent_tags
  - (self, n: int = 3): 기본값이 3, 미지정시 최근 3개의 주문을 기준으로 태그 생성
  - tags: List[str] = []: 빈 태그 리스트 생성
  - order in self.orders[-n:]: 최근 n개의 주문을 확인
  - tags.extend(order.beverage.tags) → list(set(tags)): 그 음료의 태그를 전부 모으고 중복 제거 후 리스트로 변환

▼ RecommendationEngine 클래스

```
class RecommendationEngine: # RecommendationEngine 클래스: 태그 기반 음료 추천 기능
    def __init__(self, menu: List[Beverage]) -> None: # menu 타입 명시 (Beverage 객체들이 들어 있는 리스트)
        self.menu = menu # menu 리스트 저장
    def recommend(self, user: User) -> List[Beverage]: # user 타입 명시(User), 반환 타입 명시 (Beverage
객체들이 들어 있는 리스트)
        recent_tags = user.get_recent_tags() # 사용자의 최근 태그 목록
        recommendations: List[Beverage] = [] # 추천 음료를 저장할 리스트 recommendations 초기화
        for beverage in self.menu:
            if any(tag in beverage.tags for tag in recent_tags): # 태그가 겹치는 음료의 경우
                recommendations.append(beverage) # 추천 목록 리스트에 추가
        return recommendations
```

- __init__
  - menu: List[Beverage]: menu 타입 명시 (Beverage 객체들이 들어 있는 리스트)
- recommend
  - (self, user: User) → List[Beverage] → user 타입 명시(User), 반환 타입 명시 (Beverage 객체들이 들어 있는 리스트)
  - user.get_recent_tags(): 사용자의 최근 태그 목록을 가져옴
  - any(tag in beverage.tags for tag in recent_tags) → recommendations.append(beverage): **태그가 겹치는 음료**를 추천 목록에 추가 (겹치는 태그가 하나라도 있으면 추천)

▼ 실행

```
# 음료 메뉴 정의
menu = [
    Beverage("아이스 아메리카노", 3000, ["커피", "콜드"]),
    Beverage("카페라떼", 3500, ["커피", "밀크", "웜"]),
    Beverage("녹차", 2800, ["차", "핫", "머그"]),
    Beverage("헤이즐넛티", 3000, ["차", "머그"]),
    Beverage("콜드브루", 4000, ["커피", "콜드", "진한"]),
    Beverage("바닐라 라떼", 3800, ["커피", "밀크", "달콤"]),
    Beverage("카라멜 마끼아또", 4200, ["커피", "밀크", "달콤"]),
    Beverage("홍차", 2700, ["차", "핫"]),
    Beverage("페퍼민트티", 2900, ["차", "허브"]),
    Beverage("레몬티", 3200, ["차", "과일"]),
    Beverage("망고 스무디", 4500, ["스무디", "과일", "차가운"]),
    Beverage("딸기 스무디", 4500, ["스무디", "과일", "차가운"]),
    Beverage("초코 프라푸치노", 5000, ["커피", "초코", "차가운"]),
    Beverage("그린티 프라푸치노", 5000, ["차", "차가운", "밀크"]),
    Beverage("핫초코", 3500, ["초코", "핫", "밀크"]),
    Beverage("얼그레이 라떼", 3700,["차", "밀크", "웜"]),
    Beverage("허니 자몽티", 3900, ["차", "과일", "달콤"]),
    Beverage("아포가토", 4800, ["커피", "아이스크림", "디저트"]),
    Beverage("플랫화이트", 3600, ["커피", "밀크"]),
    Beverage("마키아토", 3300, ["커피", "진한"]),
]

# 사용자의 주문 내역
user = User("철수")
user.add_order(Order(menu[0], 1)) # 아메리카노 1잔 주문
user.add_order(Order(menu[1], 2)) # 카페라떼 2잔 주문

# 최근 주문된 음료를 바탕으로 유사한 태그의 음료 추천
recommender = RecommendationEngine(menu) # 추천 엔진 생성
recommended = recommender.recommend(user) # 최근 태그를 기반으로 음료 추천 목록 생성

# 추천 음료 출력
print("추천 음료:")
for b in recommended:
    print(f"- {b.name} (tags: {b.tags})") # 추천 음료 목록 출력

# 총 주문 금액 계산
print()
print(f"총 주문 금액: {user.get_total_spent():,.0f}원") # 총 주문 금액 출력
```

▼ **실행 결과**

```
추천 음료:
- 아이스 아메리카노 (tags: ['커피', '콜드'])
- 카페라떼 (tags: ['커피', '밀크', '웜'])
- 콜드브루 (tags: ['커피', '콜드', '진한'])
- 바닐라 라떼 (tags: ['커피', '밀크', '달콤'])
- 카라멜 마끼아또 (tags: ['커피', '밀크', '달콤'])
- 초코 프라푸치노 (tags: ['커피', '초코', '차가운'])
- 그린티 프라푸치노 (tags: ['차', '차가운', '밀크'])
- 핫초코 (tags: ['초코', '핫', '밀크'])
- 얼그레이 라떼 (tags: ['차', '밀크', '웜'])
- 아포가토 (tags: ['커피', '아이스크림', '디저트'])
- 플랫화이트 (tags: ['커피', '밀크'])
- 마키아토 (tags: ['커피', '진한'])

총 주문 금액: 10,000원
```