

## SF 거래 시스템 아이템 추가

```
메뉴
-> 구매/판매, 리스트, 프로그램 종료
-구매/판매
아이템 코드 입력
-----
물건/재료 가치함 = 0
-> 물건 이라면?
-----
투입된 재료 코드 입력
입력된 코드에 대응되는 재료 수량 입력
계산 = 물건_가치함
물건_가치함 변수에 합산
stop 이라고 말할 때 까지 반복
-----
-> 재료 라면?
재료 코드 입력
입력된 코드에 대응되는 재료 수량 입력
-----
물건_가치함 or 재료_가치함 변수 총_가치함 변수에 더함
stop 이라고 말할 때 까지 반복
-----
이게 내가 생각한 프로세스인데 잘못된 부분이 있을까?
```

◆ 제시해주신 프로세스는 전체적인 흐름이 매우 논리적이고 체계적입니다. 특히 '물건'은 재료들의 합으로 가치가 결정되고, '재료'는 그 자체의 고유 가치로 계산된다는 구분이 명확해서 프로그래밍하기 아주 좋은 구조입니다.

다만, 실제 코드로 구현할 때 더 완벽해질 수 있도록 몇 가지 디테일한 보정 사항을 짚어드릴게요.

### 🔍 프로세스 리뷰 및 보완점

#### 1. 재료 단독 거래 시의 처리:

- 현재 프로세스에서 '재료라면?' 부분에 수량 입력은 있지만 **가치 계산 식이 빠져 있습니다.** `재료 가치 * 수량` 을 `총_가치함` 에 더하는 과정이 명시되어야 합니다.

#### 2. 물건 가치 계산 시 가산 요소:

- 이전 대화에서 언급하신 **작업량(Workload), 품질, 내구성, 마진** 등의 변수를 어디서 입력받을지 결정해야 합니다. 보통 '물건'의 재료 투입이 모두 끝난 직후(stop 직후)에 입력받는 것이 자연스럽습니다.

#### 3. 데이터 무결성 확인:

- 코드 입력 시 리스트에 없는 코드를 입력했을 경우에 대한 예외 처리(에러 메시지 출력 후 재입력)가 추가되면 훨씬 견고한 프로그램이 됩니다.

🔗 보완된 프로세스 로직 (흐름도)

리스트를 효율적으로(여러 데이터들을 보관 할 수 있도록)사용하는 방법을 알려줘!

◆ 데이터를 단순히 한 줄로 나열하는 것을 넘어, '구조화'해서 관리하면 프로그램이 훨씬 강력해집니다. 우리가 만든 거래 시스템처럼 여러 속성(이름, 가격, 코드 등)을 가진 데이터를 다룰 때 유용한 3가지 레벨의 리스트 활용법을 알려드릴게요.

### 1. 2차원 리스트 (가장 기본적인 장부 형태)

리스트 안에 또 다른 리스트를 넣는 방식입니다. 엑셀 시트처럼 행(Row)과 열(Column)로 데이터를 이해하면 쉽습니다.

```
Python ↕ 📄  
  
# [이름, 가격, 코드] 순서로 저장된 2차원 리스트  
registry = [  
    ["철광석", 10.0, "M1"],  
    ["실리콘", 25.0, "M2"],  
    ["갈륨프러임", -1, "I1"]  
]  
  
# 같은 방법  
print(registry[0]) # 첫 번째 항목 전체: ["철광석", 10.0, "M1"]  
print(registry[0][0]) # 첫 번째 항목의 이름: "철광석"
```

- **장점:** 구현이 매우 쉽고 직관적입니다.
- **단점:** `registry[0][2]` 가 무엇을 의미하는지 코드를 판 본인만 알 수 있습니다 (가독성이 떨어짐).

### 2. 리스트 속의 딕셔너리 (가장 추천하는 방식)

리스트 안에 키(Key)와 값(Value)으로 이루어진 딕셔너리를 넣는 방식입니다. 현재 실무에서 가장 많이 쓰이는 형태입니다.

```
Python ↕ 📄  
  
# 각 데이터를 '이름'표를 붙여 저장  
registry = [  
    {"name": "철광석", "price": 10.0, "code": "M1"},  
    {"name": "실리콘", "price": 25.0, "code": "M2"},  
    {"name": "갈륨프러임", "price": -1, "code": "I1"}  
]  
  
# 같은 방법  
for item in registry:
```

🗨 Gemini 프롬프트 입력

+ 🗨 도구

빠른 모델 ▾

