Grocery App

A console-based grocery shopping application built in Python, running in Google Colab, using JSON for data storage. The app manages a grocery cart with products priced in Indian Rupees (₹), supports save-for-later functionality, generates invoices, collects feedback, and displays data.

Features and Functionalities

- **Product Management**: Predefined list of 10 products (e.g., potato, tomato, bread) with prices in ₹ (e.g., potato: ₹2.0, Sweet-kalakand: ₹190.0).
- **Cart Operations**:
  - Add items to the cart (e.g., `add 1,12` for 12 potatoes).
  - Update item quantities in the cart (e.g., `update 2,6` for 6 tomatoes).
  - Remove items from the cart (e.g., `remove 1`).
- **Save for Later**:
  - Save items from cart to a saved-for-later list (e.g., `save 3,34` for 34 bread).
  - Move items from saved-for-later to cart (e.g., `move 3,30` for 30 bread).
  - View saved-for-later items in a formatted table.
- **Payment and Invoice**: Calculate total cost and generate a detailed invoice with item names, quantities, prices, and total.
- **Feedback Collection**: Optionally collect feedback after transactions.
- **Data Display**: Show contents of all JSON files (products, pricing, transactions, feedback) in formatted tables after transaction completion.
- **Error Handling**: Validate product IDs, quantities, and availability; handle invalid inputs.

Code Flow

1. **Initialization**:
   a. Creates JSON files (`products.json`, `pricing.json`, `transactions.json`, `feedback.json`) with initial data if they don't exist.
   b. Initializes a `Cart` object and a unique transaction ID.
2. **Main Loop**:
   a. Displays products and prices.

    b. Prompts for actions: `add`, `update`, `remove`, `save`, `move`, `view`, or done.
3. **Action Handling**:
    a. Processes user inputs for cart operations, save-for-later, and viewing saved items.
    b. On done, saves transaction, generates invoice, collects feedback, and displays JSON contents.
4. **Data Persistence**:
    a. Stores transactions and feedback in JSON files; products and pricing remain static.

Data Structures

- **Dictionaries**:
    - **Used For**: Products (`{1: "potato", 2: "tomato"}`), pricing (`{1: 2.0, 2: 1.5}`), cart items (`{1: 12, 2: 6}`), and saved-for-later items.
    - **Why**: Enable fast O(1) lookups by product ID, align with JSON's structure, and represent key-value relationships (e.g., ID → name).
- **Lists**:
    - **Used For**: Transactions (`[{transaction_id, items, total, timestamp}]`) and feedback in JSON files.
    - **Why**: Store sequential records for history, support appending new entries, and map to JSON arrays.

Why JSON?

- **Structured Data**: Handles nested data (e.g., transaction `items: {1: 12, 2: 6}`) better than CSV's semicolon-separated strings (`1:12;2:6`).
- **Python Compatibility**: Maps directly to dictionaries, simplifying data handling.
- **Extensibility**: Supports adding fields (e.g., discounts) without format changes.
- **Error Reduction**: Avoids CSV's delimiter-related errors, ensuring robust storage.

Dependencies

- Python standard library (`json`, `os`, `uuid`, `datetime`).
- No external packages required.