import csv

instruments = [] # List to store instrument dictionaries

def load\_instruments():

global instruments

try:

with open("instruments.csv", "r", newline="") as csvfile:

reader = csv.DictReader(csvfile)

for row in reader:

instruments.append(dict(row))

except FileNotFoundError:

print("No instruments file found. Creating a new one.")

def save\_instruments():

global instruments

with open("instruments.csv", "w", newline="") as csvfile:

writer = csv.DictWriter(csvfile, fieldnames=instruments[0].keys())

writer.writeheader()

writer.writerows(instruments)

def add\_instrument():

"""Prompts user for instrument details and adds it to the list."""

new\_instrument = {} # Empty dictionary to store instrument details

# Get instrument details from user

new\_instrument["id"] = int(input("Enter instrument ID (unique): "))

new\_instrument["name"] = input("Enter instrument name: ")

new\_instrument["brand"] = input("Enter instrument brand: ")

new\_instrument["model"] = input("Enter instrument model (optional): ")

new\_instrument["category"] = input("Enter instrument category: ")

new\_instrument["quantity"] = int(input("Enter quantity: "))

new\_instrument["price"] = float(input("Enter price: "))

rental\_price = input("Enter rental price (optional, leave blank if not for rent): ")

if rental\_price:

new\_instrument["rental\_price"] = float(rental\_price)

else:

new\_instrument["rental\_price"] = None

new\_instrument["condition"] = input("Enter instrument condition: ")

new\_instrument["notes"] = input("Enter any additional notes (optional): ")

def view\_instruments():

"""Displays all instruments or allows filtering based on user input."""

print("\nInstruments:")

print("{:<5} {:<20} {:<15} {:<15} {:<10} {:<10} {:<10}".format(

"ID", "Name", "Brand", "Category", "Quantity", "Price", "Rental Price"

))

print("-" \* 100)

# Display all instruments (default behavior)

if True: # Change to if True for full list or implement filtering logic here

for instrument in instruments:

rental\_price = instrument.get("rental\_price") # Handle missing rental price

print("{:<5} {:<20} {:<15} {:<15} {:<10} {:<10.2f} {:<10.2f}".format(

instrument["id"],

instrument["name"],

instrument["brand"],

instrument["category"],

instrument["quantity"],

instrument["price"],

rental\_price if rental\_price else "-"

))

else:

# Implement filtering logic based on user input (category, price range, etc.)

pass

print("\n")

def update\_instrument():

"""Searches for an instrument by ID and allows updates."""

instrument\_id = int(input("Enter the ID of the instrument to update: "))

# Find the instrument with the matching ID

found\_instrument = None

for instrument in instruments:

if instrument["id"] == instrument\_id:

found\_instrument = instrument

break

if not found\_instrument:

print("Instrument not found. Please try again with a valid ID.")

return

print("\nCurrent details:")

for key, value in found\_instrument.items():

print(f"{key.capitalize()}: {value}")

# Prompt user for fields to update

update\_fields = input("\nEnter comma-separated list of fields to update (or 'all' for all): ")

if update\_fields.lower() == "all":

fields\_to\_update = found\_instrument.keys() # Update all fields

else:

fields\_to\_update = [field.strip().lower() for field in update\_fields.split(",")]

# Get user input for specific updates

updates = {}

for field in fields\_to\_update:

if field in found\_instrument:

new\_value = input(f"Enter new value for {field.capitalize()}: ")

# Handle numerical conversions (optional)

if field in ("quantity", "price", "rental\_price"):

new\_value = float(new\_value)

updates[field] = new\_value

else:

print(f"Invalid field: {field}")

# Update the instrument dictionary with confirmed changes

found\_instrument.update(updates)

# Save changes to persistence layer (replace with your specific method)

save\_instruments() # Assuming you have a save\_instruments function

print(f"Instrument with ID {instrument\_id} updated successfully!")