CapIA.py - A Transaction Data Processor

This program takes financial transaction data from a CSV file, cleans it up, converts it to a standard format, and saves it as a JSON file. Let's examine it section by section:

1. Importing Libraries

```
import csv
import re
import os
import json
from datetime import datetime
from decimal import Decimal
```

These first lines bring in tools the program needs:

- csv: Helps read and work with spreadsheet files
- re: Lets the program search and modify text using patterns
- os: Provides ways to interact with the computer's file system
- json: Used to create and read JSON data (a common format for storing information)
- datetime: Helps work with dates and times
- Decimal: Handles money amounts precisely (unlike regular numbers which can have rounding errors)

2. Detecting the CSV File's Delimiter

```
def detect_delimiter(file_path):
    with open(file_path, 'r', newline='', encoding='utf-8') as f:
        sample = f.read(1024)  # Read a small portion of the file
        sniffer = csv.Sniffer()
        return sniffer.sniff(sample).delimiter
```

- The program reads a small part of the file (1024 characters).
- It uses csv.Sniffer() to automatically detect which symbol (,, ;, |, etc.) separates the values.

3. Standardizing Column Names

```python
def normalize_column_name(column_name):
    return re.sub(r'[\s\-]+', '_', column_name.strip().lower())
```

- Converts column names to snake_case (e.g., "Transaction Date" → "transaction_date").
- Spaces and hyphens are replaced with underscores (_).
- Makes everything lowercase.

4. Cleaning and Converting Currency Values

```python
def clean_amount(amount_str):
    if not amount_str:
        return Decimal('0.00')

    if re.search(r'\d+,\d{2}$', amount_str):
        amount_str = amount_str.replace(".", "").replace(",", ".")
    else:
        amount_str = amount_str.replace(",", "")

    return Decimal(amount_str)
```

- Removes thousands separators (1,234.56 → 1234.56)
- Converts different decimal formats (e.g 1.234,56 → 1234.56)
- If the amount is empty, it defaults to 0.00

5. Formatting Status Values

```python
def normalize_status(status):
    return status.strip().lower()
```

-Removing any extra spaces.

-Converting everything to lowercase. So "COMPLETED", "Completed", and "completed " all become "completed"

6. Converting Dates to a Standard Format

```
def normalize_date(date_str):
    for fmt in ("%Y-%m-%d", "%d-%m-%Y", "%m/%d/%Y"):
        try:
            return datetime.strptime(date_str.strip(),
fmt).date()
        except ValueError:
            continue
    raise ValueError(f"Unrecognized date format: {date_str}")
```

- It tries three common formats (year-month-day, day-month-year, month/day/year)
- Converts them all to YYYY-MM-DD format.
- If the format is unrecognized, it raises an error.

7. Checking if the File Has Headers

```
def has_headers(file_path):
    with open(file_path, 'r', newline='', encoding='utf-8') as f:
        sample = f.read(1024)
        return csv.Sniffer().has_header(sample)
```

- Reads the first few lines to check if the file has column headers.

8. Saving the Processed Data as JSON

```
def save_json(data, output_path):
    os.makedirs("JSON", exist_ok=True)
    with open(output_path, "w", encoding="utf-8") as json_file:
        json.dump(data, json_file, indent=4, default=str)
```

- Creates a JSON folder if it doesn't exist.
- Saves the processed data in a nicely formatted JSON file.

9. Processing the CSV File

```python
def process_csv(file_path):
    delimiter = detect_delimiter(file_path)
    column_mapping = {
        0: 'transaction_date',
        1: 'description',
        2: 'amount',
        3: 'currency',
        4: 'status'
    }

    with open(file_path, 'r', newline='', encoding='utf-8') as f:
        reader = csv.reader(f, delimiter=delimiter, quotechar='"')
        headers = next(reader)

        if has_headers(file_path):
            headers = [normalize_column_name(col) for col in headers]
        else:
            headers = [column_mapping[i] for i in range(len(column_mapping))]

        normalized_data = []
        for row in reader:
            if len(row) != len(headers):
                print(f"Skipping row due to column mismatch: {row}")
                continue  # Skip invalid rows

            data_dict = dict(zip(headers, row))

            data_dict['transaction_date'] = normalize_date(data_dict.get('transaction_date', ''))
            data_dict['amount'] = clean_amount(data_dict.get('amount', ''))
            data_dict['status'] = normalize_status(data_dict.get('status', ''))

            normalized_data.append(data_dict)

    return normalized_data
```

- Detects the delimiter and reads the file.
- If headers are present, it standardizes them.
- If no headers are present, it assigns default column names.
- Cleans and formats each row.
- Skips rows if they have missing or extra values.
- Returns a list of processed transactions.

Running the Program

1. Sets the input file as say "no_header.csv" (in the same directory)
2. Sets the output file as "JSON/output.json"
3. Saves the results as JSON and Prints each processed row to the screen