Mark as done

Description Submission view

Available from: Tuesday, 11 November 2025, 9:15 AM
Due date: Tuesday, 11 November 2025, 11:05 AM
Requested files: csvtool_p1.py, p1_input.txt, p2_input.txt, data.csv, users.csv, p1.py, p2.py, csvtool_p2.py (Download)
Maximum number of files: 9
Type of work: Individual work

Problem 1 — CSV Tool: Print Subsection

Description

You must implement a Python function named print_subsection in `csvtool_p1.py.`
This function prints a rectangular subsection of a CSV file based on given row and column index ranges.

A subsection is the rectangular block of cells defined by the intersection of rows [r1, r2) and columns [c1, c2).
That is, all entries from row r1 (inclusive) up to but not including r2, and from column c1 (inclusive) up to but not including c2.

Function Specification

```
def print_subsection(file: str, r1: int, r2: int, c1: int, c2: int) -> None:
```

Parameters

file — Path to the CSV file (string).
r1 — Start row index (inclusive, integer, non-negative).
r2 — End row index (exclusive, integer, non-negative).
c1 — Start column index (inclusive, integer, non-negative).
c2 — End column index (exclusive, integer, non-negative).

Returns

Nothing. The function must print the requested subsection directly to standard output.

Validation and Exception Rules

All validation checks must be executed in the exact order specified below.
Within each category, the parameters must be checked sequentially in this order: r1, r2, c1, c2.

1. **File Existence**
   If the specified file does not exist:
   raise FileNotFoundError("File " + file + " not found")

2. **Type Validation**
   If any of r1, r2, c1, or c2 is not an integer:
   raise ValueError("All indices must be integers")

3. **Non-Negative Validation**
   If any of r1, r2, c1, or c2 is negative:
   raise ValueError("All indices must be non-negative")

4. **Range Validation**
   If r1 > r2 or c1 > c2:
   raise ValueError("Invalid range: r1 must be <= r2 and c1 must be <= c2")

5. **Row Range Validation**
   Let max_row_index = total number of rows - 1.
   If the file is empty, treat it as having range (0--1).

Check in order:
If r1 > max_row_index:
raise IndexError("Row index " + str(r1) + " out of range (0-" + str(max_row_index) + ")")

Else if r2 > total number of rows:
raise IndexError("Row index " + str(r2) + " out of range (0-" + str(max_row_index) + ")")

Note: Since r2 is exclusive, r2 == total number of rows is valid and must not raise an error. The exception applies only when r2 exceeds the total number of rows.

6. **Column Range Validation**
   Let max_col_index = total number of columns - 1 (from the header line).

Check in order:
If c1 > max_col_index:
raise IndexError("Column index " + str(c1) + " out of range (0-" + str(max_col_index) + ")")
Else if c2 > total number of columns:
raise IndexError("Column index " + str(c2) + " out of range (0-" + str(max_col_index) + ")")

Note: Since c2 is exclusive, c2 == total number of columns is valid and must not raise an error. The exception applies only when c2 exceeds the total number of columns.

7. **Subsection Output**
   After all validations pass, print each line of the subsection.
   For every row i in [r1, r2) and every column j in [c1, c2), print the corresponding cell values joined by commas.
   Each row of output must appear on a separate line.

Behavior Requirements

- The function must not modify the CSV file.
- Output must maintain CSV formatting (comma-separated, no extra spaces).
- Start indices are inclusive, end indices are exclusive.
- Indexing is 0-based for both rows and columns.
- The header row (row 0) counts toward the row indices.
- Use only standard file reading and string methods — no external libraries (including csv).

Examples

Given data.csv:

```
Name,Age,City,Salary
John,25,London,50000
Alice,30,Paris,60000
Bob,35,New York,70000
Charlie,28,Tokyo,55000
Diana,32,Berlin,65000
```

Example 1
Command:

```
python3 csvtool_p1.py -sub data.csv 0 3 0 2
```

Output:

```
Name,Age
John,25
Alice,30
```

Example 2
Command:

```
python3 csvtool_p1.py -sub data.csv 1 4 1 3
```

Output:

```
25,London
30,Paris
35,New York
```

Testing Instructions
To test your code locally:

1. Create a simple CSV file data.csv:
   ```
   Name,Age
   Alice,30
   Bob,35
   ```
2. Edit p1_input.txt to include:
   ```
   1
   python3 csvtool.py -sub data.csv 0 2 0 2
   ```
3. Run your code. Expected output:
   ```
   Name,Age
   Alice,30
   ```

Notes

- The specification defines both which conditions to check and the exact order.
- Any deviation from the prescribed validation order or message wording will be treated as incorrect.
- For out-of-range messages:
  <max_row_index> = total rows − 1
  <max_col_index> = total columns − 1.

Problem 2 — CSV Tool: Remove Duplicates

Description
You must implement a Python function named dedupe in the file `csvtool_p2.py.`
This function removes duplicate rows from a CSV file based on one or more key columns, keeping either the first or last occurrence as specified.
You must write the function exactly as described in the specifications below.

A **"duplicate"** means: all values in the given key columns are identical between two or more rows.
Rows are only considered duplicates when all specified key column values match exactly (case-sensitive).
If even one key column differs, the rows are treated as distinct.

Function Specification
```
def dedupe(file: str, keycols: list[str], out: str, keep: str = "last") -> None:
```

Parameters
file — Path to the input CSV file.
keycols — List of column names used together as keys for deduplication.
out — Path to the output CSV file.
keep — Which duplicate to keep: "first" or "last" (default is "last").

Returns / Output
The function must not print anything and must not return any value.
Its only responsibility is to perform the file operation or raise an appropriate exception.
A successful execution produces no terminal output.

Required Exceptions and Order of Validation
All validations must be performed in the following strict order, exactly as implemented in the program.

1. **keep Parameter Validation**
   If keep is not "first" or "last", raise
   ValueError("keep must be 'first' or 'last'")
2. **File Existence Check**
   If the input file does not exist, raise
   FileNotFoundError("File " + file + " not found")
3. **Empty File Check**
   If the input file exists but contains no lines, raise
   ValueError("File " + file + " is empty")
4. **Key Column Validation**
   If any column listed in keycols is not present in the header row, raise
   ValueError("Column '" + k + "' not found in CSV")

Error checks must follow this exact order. Once an error is raised, no further validations or operations must be performed

Behavior

• Deduplication is performed using all columns in keycols collectively.
Two rows are duplicates only if all key column values match exactly.
• When keep="first", the first occurrence of each key combination is preserved.
• When keep="last", the last occurrence of each key combination is preserved.
• Header and valid CSV structure must always be preserved.
• The order of remaining rows must follow the logical preservation rule ("first" or "last").
• Use only standard Python file and string operations — no external dependencies or csv module.
• Matching of column names is case-sensitive.

Examples

Given users.csv:

```
ID,Name,Email,Age
1,John,john@email.com,25
2,Alice,alice@email.com,30
3,Bob,bob@email.com,35
4,John,john@email.com,28
5,Charlie,charlie@email.com,40
6,Diana,diana@email.com,32
7,John,john@email.com,26
8,Eve,eve@email.com,29
```

Example 1 — Keep last occurrence by Email
Command:

```
python3 csvtool_p2.py -dedupe users.csv Email clean_users.csv last
```

Behavior: Must create clean_users.csv keeping only the last duplicate for each unique Email.

Output file (clean_users.csv):

```
ID,Name,Email,Age
2,Alice,alice@email.com,30
3,Bob,bob@email.com,35
5,Charlie,charlie@email.com,40
6,Diana,diana@email.com,32
7,John,john@email.com,26
8,Eve,eve@email.com,29
```

Example 2 — Keep first occurrence by Name and Email (multi-column deduplication)
Command:

```
python3 csvtool_p2.py -dedupe users.csv Name Email unique_combined.csv first
```

Behavior: Deduplicate only when both Name and Email match exactly.
Rows that share the same Name but have different Emails are not treated as duplicates.

Output file (unique_combined.csv):

```
ID,Name,Email,Age
1,John,john@email.com,25
2,Alice,alice@email.com,30
3,Bob,bob@email.com,35
5,Charlie,charlie@email.com,40
6,Diana,diana@email.com,32
8,Eve,eve@email.com,29
```

Testing Instructions

1. Create users.csv:
   ID,Email,Name
   1,john@email.com,John
   2,alice@email.com,Alice
   3,john@email.com,John

2. Create p2_input.txt:
   ```
   1
   python3 csvtool.py -dedupe users.csv Email dedup_test.csv last
   ```

3. Run the test. The function should produce no printed output.

Expected preview of dedup_test.csv:

```
ID,Email,Name
2,alice@email.com,Alice
3,john@email.com,John
```

Notes

• keycols is passed as a Python list of column names (not space-separated).
• Deduplication occurs only when all key column values match exactly.
• keep parameter can be "first" or "last".
• Preserve header and correct CSV formatting.
• No printing or returning of results.
• The written output file must contain the exact deduplicated content.

Requested files

csvtool_p1.py

```
#!/usr/bin/env python3

# You may only add code inside the print_subsection function.
# You are NOT allowed to use the csv module for this exercise.

from lab_utils import run_csvtool_p1_main

# ================================================================================
# ================== WRITE YOUR FUNCTION IN THE SPACE BELOW =======================
# ================================================================================


# ================================================================================
# =================== DO NOT MODIFY THE CODE BELOW THIS LINE ======================
# ================================================================================

if __name__ == '__main__':
    run_csvtool_p1_main(print_subsection_func=print_subsection)

```

p1_input.txt

```
2
python3 csvtool_p1.py -sub data.csv 0 2 0 1
python3 csvtool_p1.py -sub data.csv 1 3 1 2

```

p2_input.txt

```
2
python3 csvtool_p2.py -dedupe users.csv Email clean_users.csv last
python3 csvtool_p2.py -dedupe users.csv Name unique_names.csv first
```

data.csv

```
Name,Age,City,Salary
John,25,London,50000
Alice,30,Paris,60000
Bob,35,New York,70000
Charlie,28,Tokyo,55000
Diana,32,Berlin,65000
```

users.csv

```
ID,Name,Email,Age
1,John,john@email.com,25
2,Alice,alice@email.com,30
3,Bob,bob@email.com,35
4,John,john@email.com,28
5,Charlie,charlie@email.com,40
6,Diana,diana@email.com,32
7,John,john@email.com,26
8,Eve,eve@email.com,29
```

p1.py

```python
import subprocess
import shlex

############################ Do Not Change This File #################################
# This file implements a caller function for the csvtool utility.
# It is used by the exercises and must not be modified.

def csvtool_caller(command: str) -> str:
    """Execute csvtool command."""
    try:
        args = shlex.split(command)
        result = subprocess.run(
            args,
            capture_output=True,
            text=True,
            timeout=2
        )
        if result.returncode != 0:
            # Return the error exactly as the program would raise it
            return result.stderr.strip()
        return result.stdout.strip()
    except subprocess.TimeoutExpired:
        return "Error: Command timed out"
    except Exception as e:
        return f"{type(e).__name__}: {str(e)}"


def solution(inp: str) -> str:
    return csvtool_caller(inp)


def process_input(filename: str):
    """Reads commands from input file."""
    with open(filename, 'r') as f:
        lines = [line.strip() for line in f if line.strip()]
    num_tests = int(lines[0])
    return lines[1:num_tests + 1]


if __name__ == "__main__":
    try:
        inputs = process_input('p1_input.txt')
        for command in inputs:
            print(f"Executing: {command}")
            output = solution(command)
            print("--- Tool Output ---")
            print(output if output else "[No output produced]")
            print("-------------------\n")
    except FileNotFoundError:
        print("p1_input.txt not found.")

```

p2.py

```python
import subprocess
import shlex
import os

############################ Do Not Change This File ##################################
# This file implements a caller function for the csvtool utility.
# It is used by the exercises and must not be modified,
# except for the local testing preview section at the end.
#######################################################################################


def csvtool_caller(command: str) -> str:
    """Execute csvtool command."""
    try:
        args = shlex.split(command)
        result = subprocess.run(
            args,
            capture_output=True,
            text=True,
            timeout=2
        )
        if result.returncode != 0:
            return result.stderr.strip()
        return result.stdout.strip()
    except subprocess.TimeoutExpired:
        return "Error: Command timed out"
    except Exception as e:
        return f"Error: Failed to execute command - {str(e)}"


def solution(inp):
    return csvtool_caller(inp)


def process_input(filename):
    lines = open(filename, 'r').readlines()
    lines = [line.strip() for line in lines if line.strip()]
    num_tests = int(lines[0])
    input_tests = [lines[t].strip() for t in range(1, num_tests + 1)]
    return input_tests


# --- MODIFIED SECTION FOR LOCAL TESTING AND FILE PREVIEW ---
if __name__ == "__main__":
    try:
        Input = process_input("p2_input.txt")
        for command in Input:
            print(f"Executing: {command}")

            # Execute student's csvtool command
            program_output = solution(command)

            # Show stdout clearly (dedupe doesn't usually print anything)
            if program_output.strip():
                print(f"\n--- Output from your program ---\n{program_output}\n------------------------------\n")
            else:
                print("\n--- Output from your program ---\n[No output produced]\n------------------------------\n")

            # --- File Preview Logic (for dedupe) ---
            try:
                parts = shlex.split(command)
                if "-dedupe" in parts:
                    # According to: -dedupe <input> <keycols> <output> <keep>
                    idx = parts.index("-dedupe")
                    if len(parts) >= idx + 5:
                        output_file = parts[idx + 3]
                        print(f"======= PREVIEW of created file: '{output_file}' =======")
                        if os.path.exists(output_file):
                            with open(output_file, "r") as f:
                                content = f.read()
                            if content.strip():
                                print(content.strip())
                            else:
                                print("[ The file was created but is empty. ]")
                        else:
                            print(f"[ File '{output_file}' was not found. It may not have been created due to an error. ]")
                        print("================ END OF PREVIEW ================\n")

            except Exception as e:
                print(f"[ Could not generate file preview due to an error: {e} ]\n")

    except FileNotFoundError:
        print("p2_input.txt not found. Cannot run local tests.")
    except Exception as e:
        print(f"An unexpected error occurred in the runner script: {e}")

```

csvtool_p2.py

```python
#!/usr/bin/env python3

# You may only add code inside the dedupe function.
# You are NOT allowed to use the csv module for this exercise.

from lab_utils import run_csvtool_p2_main

# =================================================================================
# ================== WRITE YOUR FUNCTION IN THE SPACE BELOW =======================
#
# Refer to the Problem 2 description for the EXACT specifications.
#
# =================================================================================



# =================================================================================
# ================== DO NOT MODIFY THE CODE BELOW THIS LINE =======================
# =================================================================================

if __name__ == '__main__':
    # This will fail until you have defined the dedupe function.
    run_csvtool_p2_main(dedupe_func=dedupe)

```

VPL