Mark as done

Description Submission view

Available from: Friday, 14 November 2025, 9:15 AM
Due date: Friday, 14 November 2025, 10:45 AM
Requested files: csvtool_p1.py, csvtool_p2.py, p1_input.txt, p2_input.txt, products.csv, tickets.csv, p1.py, p2.py (Download)
Maximum number of files: 9
Type of work: Individual work

Problem 1 — CSV Tool: Replace Text in CSV

Description
You must implement a Python function named replace_text in csvtool_p1.py.
This function searches for occurrences of a specified text string within the cells of a CSV file and replaces them with another string.
It can optionally perform case-insensitive replacements or restrict replacements to specific columns.

The modified data must be written to a new output file while preserving the original CSV structure and formatting.

Function Specification

```
def replace_text(src: str, dest: str, old: str, new: str,ignore_case: bool = False, cols: Optional[List[str]] = None) -> None:
```

Parameters
src — Path to the input CSV file (string).
dest — Path to the output CSV file (string).
old — The substring to find and replace (string).
new — The replacement string (string).
ignore_case — Boolean flag; if True, perform case-insensitive matching. Default is False.
cols — Optional list of column names to restrict replacements. If None, replacements apply to all columns.

Returns
Nothing. The function writes the modified data to the output file directly.

Validation and Exception Rules

All validations must be executed in the exact order specified below.
Once an exception is raised, no further checks or operations must be performed.

1. **File Existence Check**
   If the input file `src` does not exist:
   raise FileNotFoundError("File " + src + " not found")

2. **Empty File Check**
   If the file exists but is empty (no lines):
   raise ValueError("File " + src + " is empty")

3. **Column Restriction Validation**
   If `cols` is provided (not None), validate that each column listed exists in the header row.
   For any missing column name `c`:
   raise KeyError("Column " + c + " not found in CSV")

4. **Output Write Validation**
   If the program fails to write to the destination file `dest` (e.g., due to permissions):
   raise PermissionError("Cannot write to file " + dest)

?

Behavior Requirements

• The function must open and read the CSV file, without using the `csv` module or any external libraries.
• Text replacement must occur only within cell contents, not across commas or line boundaries.
• When `ignore_case` is True, the function must perform replacements in a case-insensitive manner, preserving all other content.
• When `cols` is specified, only the listed columns are subject to replacement; other columns remain unchanged.
• When `cols` is None, all cells are processed.
• The header row (row 0) must always be preserved exactly, regardless of replacements.The 0th row will always be header.
• The CSV output must maintain comma-separated formatting with no extra spaces or trailing commas.
• The function must not print or return anything.
• The function must never modify the original input file.

Examples

Given products.csv:

```
Product,Brand,Description
Pen,Alpha Ltd.,Red Ink
Notebook,Beta LTD.,Blue Cover
Red Eraser,Gamma Co.,White Rubber
```

Example 1 — Simple Replacement

Command:

```
python3 csvtool_p1.py -replace products.csv out.csv "Ltd." "Limited"
```

Output (out.csv):

```
Product,Brand,Description
Pen,Alpha Limited,Red Ink
Notebook,Beta LTD.,Blue Cover
Red Eraser,Gamma Co.,White Rubber
```

Example 2 — Case-Insensitive Replacement

Command:

```
python3 csvtool_p1.py -replace products.csv out.csv "ltd." "Limited" --ignore_case
```

Output (out.csv):

```
Product,Brand,Description
Pen,Alpha Limited,Red Ink
Notebook,Beta Limited,Blue Cover
Red Eraser,Gamma Co.,White Rubber
```

Example 3 — Column-Restricted Replacement

Command:

```
python3 csvtool_p1.py -replace products.csv out.csv "Red" "Crimson" --ignore_case --cols Product,Description
```

Output (out.csv):

```
Product,Brand,Description
Pen,Alpha Ltd.,Crimson Ink
Notebook,Beta Ltd.,Blue Cover
Crimson Eraser,Gamma Co.,White Rubber
```

Testing Instructions

Create sample products.csv:

```
Product,Color,Stock
Pen,Red,100
Pencil,Red,200
Red,White,150
```

Create p1_input.txt:

```
1
python3 csvtool_p1.py -replace products.csv out.csv Red Crimson --cols Color
```

Run your code.
Expected output file (out.csv):

```
Product,Color,Stock
Pen,Crimson,100
Pencil,Crimson,200
Red,White,150
```

Notes

• Order of validation must be strictly followed.
• No printing or console output is allowed.
• Matching is exact unless `ignore_case=True`.
• If replacement text does not occur in the file, write the output unchanged.
• The output file must always be generated unless an exception halts execution.

Problem 2 — CSV Tool: Split CSV by Column Value

Description
You must implement a Python function named split_csv in csvtool_p2.py.
This function divides a single CSV file into multiple smaller CSV files based on unique values in a specified column.
Each output file contains all rows that share the same value in that column, including the header.

Prefix is an external string provided by the caller. The function concatenates that string, an underscore, and the column value to form each output file name. For example, if the column "Category" has values "A", "B", and "C", and if the caller supplies "base" as the prefix, the function should create three files:
base_A.csv, base_B.csv, base_C.csv.

Function Specification
```
def split_csv(file: str, col: str, prefix: str) -> None:
```

Parameters
file — Path to the input CSV file (string).
col — Name of the column used to group rows (string).
prefix — string used as the base for output filenames. Each output file is named as f"{prefix}_{uniquevalue}.csv".

Returns
Nothing. The function writes multiple output files.

Validation and Exception Rules

All validations must be executed in the exact order specified below.
Once an exception is raised, no further validations or writes should occur.

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

2. **Empty File Check**
   If the file exists but contains no lines:
   raise ValueError("File " + file + " is empty")

3. **Column Validation**
   If the specified column `col` does not exist in the header row:
   raise KeyError("Column " + col + " not found in CSV")

4. **Output Write Validation**
   If writing to any output file fails:
   raise PermissionError("Cannot write file for group " + value)

Behavior Requirements

• The function must preserve the original header row in every output file.
• Each output file must include all rows sharing the same value in the specified column.
• The output filename format must be: `{prefix}_{value}.csv`, where value is a unique value from the specified column.
Example: tickets_Medium.csv or tickets_High.csv
• Values must be taken exactly as they appear in the CSV (case-sensitive).
• The function must not modify the input file.
• No extra spaces or formatting changes are allowed.
• The function must not print or return anything.
• The order of rows within each output file must match their order in the source file.
• Use only standard file and string operations, not the csv module or external libraries.

Examples

Given tickets.csv:

```
ID,Priority,Status
1,High,Open
2,Low,Closed
3,High,Resolved
4,Medium,Open
5,Low,Open
```

Example 1 — Split by Priority

Command:

```
python3 csvtool_p2.py -split tickets.csv Priority tickets
```

Output files generated:

```
tickets_High.csv
tickets_Low.csv
tickets_Medium.csv
```

Contents:

tickets_High.csv:

```
ID,Priority,Status
1,High,Open
3,High,Resolved
```

tickets_Low.csv:

```
ID,Priority,Status
2,Low,Closed
5,Low,Open
```

tickets_Medium.csv:

```
ID,Priority,Status
4,Medium,Open
```

Testing Instructions

Create tickets.csv:

```
ID,Department,Status
1,Sales,Open
2,Sales,Closed
3,HR,Open
4,IT,Pending
5,IT,Open
```

Create p2_input.txt:

```
1
python3 csvtool_p2.py -split tickets.csv Department dept
```

Run your code.

Expected files:

```
dept_Sales.csv, dept_HR.csv, dept_IT.csv
```

Expected dept_Sales.csv:

```
ID,Department,Status
1,Sales,Open
2,Sales,Closed
```

Expected dept_HR.csv:

```
ID,Department,Status
3,HR,Open
```

Expected dept_IT.csv:

```
ID,Department,Status
4,IT,Pending
5,IT,Open
```

Notes

- Validation order must match exactly as listed above.
- All output files must be valid CSVs, containing the header and relevant subset of rows.
- Column name matching is case-sensitive.
- No console output or logging is permitted.

Requested files

csvtool_p1.py

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

# You may only add code inside the replace_text function.
# You are NOT allowed to use the csv module or any external libraries.

from lab_utils import run_csvtool_p1_main
from typing import List, Optional


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



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

if __name__ == '__main__':
    # This lab utility runner handles CLI parsing and execution.
    run_csvtool_p1_main(replace_text_func=replace_text)

```

csvtool_p2.py

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

# You may only add code inside the split_csv function.
# You are NOT allowed to use the csv module or any external libraries.

from lab_utils import run_csvtool_p2_main

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



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

if __name__ == '__main__':
    run_csvtool_p2_main(split_csv_func=split_csv)

```

p1_input.txt

```
3
python3 csvtool_p1.py -replace products.csv out_case.csv "Ltd." "Limited"
python3 csvtool_p1.py -replace products.csv out_color.csv "red" "Crimson" --ignore_case --cols Description
python3 csvtool_p1.py -replace products.csv out_multi.csv "Co." "Company" --cols Brand,Description
```

p2_input.txt

```
2
python3 csvtool_p2.py -split tickets.csv Priority tickets
python3 csvtool_p2.py -split tickets.csv Status report
```

products.csv

```
Product,Brand,Description
Pen,Alpha Ltd.,Red Ink
Notebook,Beta LtD.,Blue Cover
Eraser,Gamma Co.,White Rubber
Marker,Delta Co.,Blue Marker
Glue,Omega LTD.,Transparent Adhesive
Pencil,Zeta Ltd.,Grey Graphite
Stapler,Theta Co.,Metal Body
Ruler,Sigma Ltd.,15cm Plastic
Sharpener,Gamma Co.,Silver Blade
Highlighter,Eta Ltd.,Yellow Ink
```

tickets.csv

```
ID,Priority,Status
1,High,Open
2,Low,Closed
3,High,Resolved
4,Medium,Open
5,Low,Open
```

p1.py

```python
#!/usr/bin/env python3
import subprocess
import shlex
import os

############################ Do Not Change This File #################################
# This file implements a caller function for the csvtool utility (replace_text).
# 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("p1_input.txt")
        for command in Input:
            print(f"Executing: {command}")

            program_output = solution(command)

            # Show stdout clearly
            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 replace_text) ---
            try:
                parts = shlex.split(command)
                if "-replace" in parts:
                    # According to: -replace <src> <dest> <old> <new> [--ignore_case] [--cols ...]
                    idx = parts.index("-replace")
                    if len(parts) >= idx + 3:
                        output_file = parts[idx + 2] if len(parts) >= idx + 3 else None
                        if not output_file:
                            continue
                        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("p1_input.txt not found. Cannot run local tests.")
    except Exception as e:
        print(f"An unexpected error occurred in the runner script: {e}")
```

p2.py

```python
#!/usr/bin/env python3
import subprocess
import shlex
import os

############################ Do Not Change This File #################################
# This file implements a caller function for the csvtool utility (split_csv).
# 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}")

            program_output = solution(command)

            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 split_csv) ---
            try:
                parts = shlex.split(command)
                if "-split" in parts:
                    # According to: -split <file> <col> <prefix>
                    idx = parts.index("-split")
                    if len(parts) >= idx + 4:
                        prefix = parts[idx + 3]
                        # Show preview for all files matching prefix_*.csv
                        print(f"======= PREVIEW of generated files (prefix='{prefix}') =======")
                        found = False
                        for fname in os.listdir("."):
                            if fname.startswith(prefix + "_") and fname.endswith(".csv"):
                                found = True
                                print(f"\n--- {fname} ---")
                                with open(fname, "r") as f:
                                    content = f.read().strip()
                                print(content if content else "[ Empty file ]")
                        if not found:
                            print(f"[ No files found with prefix '{prefix}_'. It may not have created any. ]")
                        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}")
```

VPL