

Mark as done

Description

Submission view

Available from: Monday, 17 November 2025, 9:15 AM

Due date: Monday, 17 November 2025, 10:45 AM

Requested files: csvtool_p1.py, csvtool_p2.py, p1_input.txt, p2_input.txt, student_data_p1.csv, student_data_p2_left.csv, student_data_p2_right.csv, p1.py, p2.py, visible_data_p1.csv, visible_emp.csv, visible_salary.csv ([Download](#))

Type of work: Individual work

Problem 1 - CSV Tool: Sanitize Dangerous Cells

Description:

Your task is to implement a function named `sanitize_csv` in the file `csvtool_p1.py`. This function will read a CSV file and create a new, "sanitized" version of it by handling potentially dangerous cell values according to a given policy.

A cell is considered **dangerous** if, after removing any leading spaces or tabs, its first character is one of the following: `+`, `-`, `@`, `$`, or `=`.

Function Specification: `sanitize_csv`

You must implement the function with this **exact** signature.

```
def sanitize_csv(file: str, out: str, policy: str) -> None:
```

Parameters:

- file:** The path to the source CSV file.
- out:** The path for the destination (sanitized) CSV file.
- policy:** A string specifying the sanitization method. It must be one of:
 - "escape":** If a cell is dangerous, its entire content should be replaced with a single hyphen (-).
 - "reject":** If any cell in a row is dangerous, the entire row must be omitted from the output file.
 - "encode":** If a cell is dangerous, the prefix **ENC** should be added to its original value (e.g., `$hello` becomes **ENC\$hello**). This modification should happen on the original value, including any leading whitespace.

Returns / Output:

- Your function should **not print anything** and should **not return any value**. Its purpose is to create the output file or raise an exception.
- The header row from the input file should always be written to the output file, even if all data rows are rejected.
- If the input file is empty (contains no lines at all), an empty output file (with no header) should be created.

Required Exceptions:

- File Not Found:** Raise a `FileNotFoundError` if the input `file` does not exist. Message: `"File <filename> not found"`.
- Permission Denied:** Raise a `PermissionError` if writing to the `out` file is not permitted. Message: `"Permission denied: cannot write to <out_filename>"`.

Examples

Assume `formulas.csv` contains:

```
Product,Formula,Notes
OfficeSuite,=SUM(A1:B5),Approved
WebBrowser, -c 'rm -rf',Malicious
TextEditor,@User,Reference
```

Example 1: Escape Policy

Test Command: `python3 csvtool_p1.py -sanitize formulas.csv sanitized_escape.csv escape`

Content of `sanitized_escape.csv`:

?

```
Product,Formula,Notes
OfficeSuite,-,Approved
WebBrowser,-,Malicious
TextEditor,-,Reference
```

Example 2: Reject Policy

Test Command: `python3 csvtool_p1.py -sanitize formulas.csv sanitized_reject.csv reject`

Content of `sanitized_reject.csv`:

```
Product,Formula,Notes
```

Example 3: Encode Policy

Test Command: `python3 csvtool_p1.py -sanitize formulas.csv sanitized_encode.csv encode`

Content of `sanitized_encode.csv`:

```
Product,Formula,Notes
OfficeSuite,ENC=SUM(A1:B5),Approved
WebBrowser,ENC -c 'rm -rf',Malicious
TextEditor,ENC@User,Reference
```

How to Test Your Code Locally

When you click the "Run" button, the `p1.py` script executes the tests from `p1_input.txt`. Since a successful operation produces no direct output, the script helps by showing a **preview** of the file that was created. This is how you verify your sanitization worked correctly.

Note: To help you understand the visible tests and debug your code, the data file `visible_data_p1.csv` (used by the auto-grader) is provided to you as a read-only file in the "Requested Files" section. You can view its content to better understand the test scenarios.

A step-by-step example of creating a custom test:

1. **Objective:** Test the "reject" policy on a small, custom CSV file.
2. **Step 1: Modify `student_data_p1.csv`.** Change its content to include one safe and one dangerous row:

```
User,Comment
Alice,Safe comment
Bob,@mention
```

3. **Step 2: Modify `p1_input.txt`.** Set up a single command to run the "reject" policy and save to a new file called `safe_comments.csv`:

```
1
python3 csvtool_p1.py -sanitize student_data_p1.csv safe_comments.csv reject
```

4. **Step 3: Click "Run".** The runner script (`p1.py`) will execute your command. The expected final output on the screen should be:

```
Executing: python3 csvtool_p1.py -sanitize student_data_p1.csv safe_comments.csv reject

--- Output from your program ---
[No output produced]
-----

===== PREVIEW of created file: 'safe_comments.csv' =====
User,Comment
Alice,Safe comment
===== END OF PREVIEW =====
```

Notes:

- You will write your code in `csvtool_p1.py`.
- You **CANNOT** use Python's built-in `csv` module.
- Policy string will always be a valid one.

Problem 2 - CSV Tool: Outer Join

Description:

For this problem, you will implement a powerful data manipulation feature: an outer join. In the file `csvtool_p2.py`, you will write a function named `join_csv` that combines two CSV files into a single file based on a shared "key" column.

An **outer join** merges two datasets, keeping all rows from *both* files. When a key from one file exists in the other, their data is combined into a single row. If a key from one file does not have a matching key in the other, its data is still included, and the columns from the other file are filled with empty values. The following image illustrates this concept:

Employees table		Departments table		
ID	Name		Dept	Manager
101	Alice		Sales	Eve
102	Bob		HR	Frank
104	Charlie		Marketing	Grace
Outer Join of Employees table and Departments table				
ID	Name	Dept	Manager	
101	Alice	Sales	Eve	
102	Bob	Marketing	Grace	
103	(Empty)	HR	Frank	
104	Charlie	(Empty)	(Empty)	

Function Specification: `join_csv`

You must implement the function with this **exact** signature.

```
def join_csv(left: str, right: str, key: str, out: str) -> None:
```

Parameters:

- **left**: The path to the "left" source CSV file.
- **right**: The path to the "right" source CSV file.
- **key**: The name of the column that is present in both files and will be used to match rows.
- **out**: The path for the destination (joined) CSV file.

Core Logic and Output Format:

- The output file's header should be constructed in a specific order: first the key column, then all other columns from the **left** file (in their original order), and finally all other columns from the **right** file (in their original order).
- For a given key, if a matching row does not exist in one of the files, the corresponding cells in the output row should be empty.
- **Handling Special Files:**
 - **Header-Only Files:** If an input file contains only a header, your function must still validate that the **key** column exists in that header. If it does, the join proceeds normally, treating the file as having no data rows.
 - **Empty Files:** If an input file is completely empty (has no lines), it has no header. Therefore, the check for the **key** column will fail, and your function **must** raise a `KeyError` as required by the exception order.
- The order of data rows in the output file **does not matter**.
- You can assume that the key column contains unique values within each source file (no duplicate keys in `left.csv`, no duplicate keys in `right.csv`).
- You can assume that there are no **common column names** between **left** and **right** files except the **key** column.

Required Exceptions:

You must check for errors and raise exceptions in a specific order:

1. First, check if the **left** file exists. If not, raise `FileNotFoundError` with the message: "File <left_filename> not found".
2. Second, check if the **right** file exists. If not, raise `FileNotFoundError` with the message: "File <right_filename> not found".
3. Third, check if the **key** column is in the **left** file's header. If not, raise `KeyError` with the message: "Column <key> not found in file <left_filename>".
4. Fourth, check if the **key** column is in the **right** file's header. If not, raise `KeyError` with the message: "Column <key> not found in file <right_filename>".
5. Fifth, check if writing to the **out** file is permitted. If not, raise `PermissionError` with the message: "Permission denied: cannot write to <out_filename>".

Example

(Data corresponds to the image example above)

Assume `employees.csv` (left file) contains:

```
ID,Name
101,Alice
102,Bob
104,Charlie
```

And `departments.csv` (right file) contains:

```
Dept,Manager,ID
Sales,Eve,101
HR, Frank,103
Marketing,Grace,102
```

Test Command: `python3 csvtool_p2.py -join employees.csv departments.csv ID joined_data.csv`

One possible correct content for `joined_data.csv` (note the empty cells for non-matches; your row order may be different):

```
ID,Name,Dept,Manager
104,Charlie,,
101,Alice,Sales,Eve
103,,HR, Frank
102,Bob,Marketing,Grace
```

How to Test Your Code Locally

The "Run" button executes the tests defined in `p2_input.txt`. The runner script (`p2.py`) will execute the command and then show you a **preview** of the file created by your function.

Note: To help you understand the visible tests and debug your code, the data files `visible_emp.csv` and `visible_salary.csv` (used by the auto-grader) are provided to you as read-only files in the "Requested Files" section. You can view their content to better understand the test scenarios.

A step-by-step example of creating a custom test:

1. **Objective:** Test joining two small, custom CSV files.
2. **Step 1: Modify `student_data_p2_left.csv`.** Change its content to:

```
UserID,Name
1,Alice
2,Bob
```

3. **Step 2: Modify `student_data_p2_right.csv`.** Change its content to include a matching and non-matching user:

```
Role,UserID
Admin,1
Guest,3
```

4. **Step 3: Modify `p2_input.txt`.** Set up a single command to join these files on the `UserID` column into `joined_roles.csv`:

```
1
python3 csvtool_p2.py -join student_data_p2_left.csv student_data_p2_right.csv UserID joined_roles.csv
```

5. **Step 4: Click "Run".** The complete output you see on screen should be:

```
Executing: python3 csvtool_p2.py -join student_data_p2_left.csv student_data_p2_right.csv UserID joined_roles.csv

--- Output from your program ---
[No output produced]
-----

===== PREVIEW of created file: 'joined_roles.csv' =====
UserID,Name,Role
1,Alice,Admin
2,Bob,
3,,Guest
===== END OF PREVIEW =====
```

Notes:

- You will write your code in `csvtool_p2.py`.
- You **CANNOT** use Python's built-in `csv` module.

Requested files

csvtool_p1.py

```

1  #!/usr/bin/env python3
2
3  from lab_utils import run_csvtool_p1_main
4  from typing import List, Dict, Any, Tuple
5
6  # =====
7  # ===== WRITE YOUR FUNCTION IN THE SPACE BELOW =====
8  #
9  # Refer to the Problem 1 description for the EXACT specifications.
10 #
11 # =====
12
13
14
15 # =====
16 # ===== DO NOT MODIFY THE CODE BELOW THIS LINE =====
17 # =====
18
19 if __name__ == '__main__':
20     run_csvtool p1 main(sanitize csv func=sanitize csv)

```

csvtool_p2.py

```

1  #!/usr/bin/env python3
2
3  from lab_utils import run_csvtool_p2_main
4  from typing import List, Dict, Any, Tuple
5
6  # =====
7  # ===== WRITE YOUR FUNCTION IN THE SPACE BELOW =====
8  #
9  # Refer to the Problem 2 description for the EXACT specifications.
10 #
11 # =====
12
13
14
15 # =====
16 # ===== DO NOT MODIFY THE CODE BELOW THIS LINE =====
17 # =====
18
19 if __name__ == '__main__':
20     run_csvtool p2 main(join csv func=join csv)

```

p1_input.txt

```

1  3
2  python3 csvtool_p1.py -sanitize student_data_p1.csv sanitized_escape.csv escape
3  python3 csvtool_p1.py -sanitize student_data_p1.csv sanitized_reject.csv reject
4  python3 csvtool p1.py -sanitize student data p1.csv sanitized_encode.csv encode

```

p2_input.txt

```

1  1
2  python3 csvtool p2.py -join student data p2 left.csv student data p2 right.csv ID joined data.csv

```

student_data_p1.csv

```

1  Product,Formula,Notes
2  OfficeSuite,=SUM(A1:B5),Approved
3  WebBrowser, -c 'rm -rf',Malicious
4  TextEditor,@User,Reference

```

student_data_p2_left.csv

```

1  ID,Name
2  101,Alice
3  102,Bob
4  104,Charlie

```

student_data_p2_right.csv

```

1  Dept,Manager,ID
2  Sales,Eve,101
3  HR,Frank,103
4  Marketing,Grace,102

```

p1.py

```

1 import subprocess
2 import shlex
3 import os
4 from typing import Tuple, Optional, List
5
6 ##### Do Not Change This File #####
7
8 def csvtool_caller(command: str) -> Tuple[str, Optional[str]]:
9     """Execute csvtool command and return combined output and destination filename."""
10    try:
11        args = shlex.split(command)
12        dest_filename = args[4] if len(args) >= 5 else None # The out file is at index 4
13
14        result = subprocess.run(
15            args, capture_output=True, text=True, timeout=2
16        )
17
18        # MODIFIED LOGIC: Ignore exit code. Return stderr if it has content, otherwise stdout.
19        stderr_content = result.stderr.strip()
20        stdout_content = result.stdout.strip()
21
22        final_output = stderr_content if stderr_content else stdout_content
23        return final_output, dest_filename
24
25    except Exception as e:
26        return f"Error: Failed to execute command - {str(e)}", None
27
28 def solution(inp: str) -> str:
29     """Wrapper function that the evaluator's subprocess calls."""
30     output, _ = csvtool_caller(inp)
31     return output
32
33 def process_input(filename: str) -> List[str]:
34     """Function the evaluator imports to get test case commands."""
35     with open(filename, 'r') as f:
36         lines = [line.strip() for line in f if line.strip()]
37         num_tests = int(lines[0])
38         return lines[1 : num_tests + 1]
39
40 if __name__ == "__main__":
41     """Logic for when a student clicks the 'Run' button."""
42     try:
43         commands = process_input('p1_input.txt')
44         for command in commands:
45             print(f"Executing: {command}")
46
47             program_output, dest_filename = csvtool_caller(command)
48
49             print(f"\n--- Output from your program ---\n{program_output if program_output else '[No output produced]'}\n-----")
50
51             if dest_filename and not program_output:
52                 print(f"===== PREVIEW of created file: '{dest_filename}' =====")
53                 if os.path.exists(dest_filename):
54                     with open(dest_filename, 'r') as f:
55                         content = f.read()
56                         print(content.strip() if content else "[ File is empty ]")
57                 else:
58                     print(f"[ File '{dest_filename}' was not created. ]")
59                 print("===== END OF PREVIEW =====\n")
60
61     except FileNotFoundError:
62         print("p1_input.txt not found. Cannot run local tests.")
63     except Exception as e:
64         print(f"An unexpected error occurred in the runner script: {e}")

```

p2.py

```

1 import subprocess
2 import shlex
3 import os
4 from typing import Tuple, Optional, List
5
6 ##### Do Not Change This File #####
7
8 def csvtool_caller(command: str) -> Tuple[str, Optional[str]]:
9     """Execute csvtool command and return combined output and destination filename."""
10    try:
11        args = shlex.split(command)
12        dest_filename = args[-1] if len(args) > 1 and args[-1].endswith('.csv') else None
13
14        result = subprocess.run(
15            args, capture_output=True, text=True, timeout=2
16        )
17
18        # MODIFIED LOGIC: Ignore exit code. Return stderr if it has content, otherwise stdout.
19        stderr_content = result.stderr.strip()
20        stdout_content = result.stdout.strip()
21
22        final_output = stderr_content if stderr_content else stdout_content
23        return final_output, dest_filename
24
25    except Exception as e:
26        return f"Error: Failed to execute command - {str(e)}", None
27
28 def solution(inp: str) -> str:
29     """Wrapper function that the evaluator's subprocess calls."""
30     output, _ = csvtool_caller(inp)
31     return output
32
33 def process_input(filename: str) -> List[str]:
34     """Function the evaluator imports to get test case commands."""
35     with open(filename, 'r') as f:
36         lines = [line.strip() for line in f if line.strip()]
37         num_tests = int(lines[0])
38         return lines[1 : num_tests + 1]
39
40 if __name__ == "__main__":
41     """Logic for when a student clicks the 'Run' button."""
42     try:
43         commands = process_input('p2_input.txt')
44         for command in commands:
45             print(f"Executing: {command}")
46
47             program_output, dest_filename = csvtool_caller(command)
48
49             print(f"\n--- Output from your program ---\n{program_output if program_output else '[No output produced]'}\n-----")
50
51             if dest_filename and not program_output:
52                 print(f"===== PREVIEW of created file: '{dest_filename}' =====")
53                 if os.path.exists(dest_filename):
54                     with open(dest_filename, 'r') as f:
55                         content = f.read()
56                         print(content.strip() if content else "[ File is empty ]")
57                 else:
58                     print(f"[ File '{dest_filename}' was not created. ]")
59                 print("===== END OF PREVIEW =====\n")
60
61     except FileNotFoundError:
62         print("p2_input.txt not found. Cannot run local tests.")
63     except Exception as e:
64         print(f"An unexpected error occurred in the runner script: {e}")

```

visible_data_p1.csv

```

1 User,Comment,Action
2 Alice,This is a great post!,None
3 Bob,=SUM(A1:B10),FormulaInjection
4 Charlie, - Executable script,PotentialAttack
5 Diana,Safe message,None
6 Eve,@mention user,Reference

```

visible_emp.csv

```

1 Name,Department,EmpID
2 Alice,Sales,E1
3 Bob,Marketing,E2
4 Charlie,Sales,E4
5 David,HR,E5

```

visible_salary.csv

```

1 EmpID,Salary
2 E1,70000
3 E3,85000
4 E4,72000
5 E5,90000

```