

[Mark as done](#)[Description](#)[Submission view](#)**Available from:** Monday, 10 November 2025, 9:15 AM**Due date:** Monday, 10 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\_p2a.csv, student\_data\_p2b.csv, p1.py, p2.py ([Download](#))**Type of work:** Individual work

## Problem 1 - CSV Tool: Get Value

### Description:

Your task is to write a Python function named `get_value` in the file `csvtool_p1.py`. This function will be the core logic for a tool that retrieves a single value from a CSV file. You must write the entire function from scratch based on the exact specifications provided below.

### Function Specification: `get_value`

You must implement the function with this **exact** signature. Note that `col` and `row` are passed as strings, and your function is responsible for converting them to integers.

```
def get_value(file: str, col: str, row: str) -> str:
```

#### Parameters:

- `file`: The path to the CSV file (a string).
- `col`: The column index, provided as a string.
- `row`: The row index, provided as a string.

#### Returns:

- The value at the specified cell, as a string. **Important: Your function must return this value. It should not use the `print()` function. The testing script is responsible for printing the value that your function returns.**

#### Required Exceptions:

Your function must not print errors. Instead, it must `raise` the following exceptions with precisely formatted messages. **When checking for out-of-range errors, you must validate the row index before you validate the column index.**

- **Invalid Type for Indices:** Raise a `ValueError`. The message must be the exact string '`Column and row must be integers`'.
- **File Not Found:** Raise a `FileNotFoundException`. The message must be "File ", followed by the filename, followed by " not found". For example, if `missing.csv` is not found, the message string should be '`File missing.csv not found`'.
- **Empty CSV File:** Raise an `IndexError`. The message must be the exact string '`CSV file is empty`'.
- **Row Out of Range:** Raise an `IndexError`. The message must be "Row index ", followed by the invalid row number, followed by " out of range". For example, if row `10` is out of range, the message should be '`Row index 10 out of range`'.
- **Column Out of Range:** Raise an `IndexError`. The message must be "Column index ", followed by the invalid column number, followed by " out of range". For example, if column `5` is out of range, the message should be '`Column index 5 out of range`'.

### Examples

For the following examples, assume `student_data_p1.csv` contains its initial content:

```
HeaderA,HeaderB,HeaderC
Data1A,Data1B,Data1C
Data2A,Data2B,Data2C
```

#### Example 1:

**Test Command:** `python3 csvtool_p1.py -get student_data_p1.csv 1 0`

**Your Function's Behavior:** Must `return "HeaderB"`. (which is the element in 1st column and 0th row in `student_data_p1.csv` file)

**Final Output on Screen (printed by the provided tool):**

?

```
HeaderB
```

**Example 2:**

*Test Command: `python3 csvtool_p1.py -get student_data_p1.csv 2 1`*

*Your Function's Behavior: Must `return "Data1C"`. (which is the element in 2nd column and 1st row in `student_data_p1.csv` file)*

*Final Output on Screen (printed by the provided tool):*

```
Data1C
```

**How to Test Your Code Locally**

The "Run" button executes the tests defined in `p1_input.txt`. You could create your own tests by modifying these files.

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

1. **Objective:** Test what happens when you request the last item in a smaller, 2x2 CSV file.

2. **Step 1: Modify** `student_data_p1.csv`. Change its content to:

```
Name,Age
Alice,30
```

3. **Step 2: Modify** `p1_input.txt`. Change its content to a single test command that targets the last cell:

```
1
python3 csvtool_p1.py -get student_data_p1.csv 1 1
```

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

```
30
```

**Notes:**

- You will write your code in `csvtool_p1.py`. Do not modify any other files.
- You **CANNOT** use Python's built-in `csv` module.
- **You can assume that all input CSV files are well-formed. This means every row in a given file will have the same number of columns as its header.**

**Problem 2 - CSV Tool: Merge Files****Description:**

In the file `csvtool_p2.py`, your task is to write a Python function named `merge_csv`. This function will be the core logic for a tool that merges two CSV files into a new destination file. You must write the entire function from scratch based on the exact specifications below.

**Function Specification: `merge_csv`**

You must implement the function with this **exact** signature, including parameter names and order.

```
def merge_csv(file1: str, file2: str, dest: str) -> None:
```

**Parameters:**

- `file1`: The path to the first source CSV file.
- `file2`: The path to the second source CSV file.
- `dest`: The path for the destination file.

**Returns / Output:**

- Your function should **not print anything** and should **not return any value**. Its only job is to perform the file operation or raise an exception if an error occurs. A successful run should produce no output.

**Required Exceptions:**

If an error occurs, your function must `raise` one of the following exceptions with a precisely formatted message.

- **File Not Found:** Raise a `FileNotFoundException`. The message must be "File ", followed by the name of the missing file, followed by " not found". For example, "File missing.csv not found".
- **Empty File:** Raise a `ValueError`. The message must be "File ", followed by the name of the empty file, followed by " is empty". For example, "File empty\_data.csv is empty".
- **Header Mismatch:** Raise a `ValueError`. The message must be the exact string '`Headers do not match between files`'.
- **Permission Denied:** Raise a `PermissionError`. The message must be "Permission denied: cannot write to ", followed by the destination filename. For example, "Permission denied: cannot write to /root/output.csv".

## Examples

For the following examples, assume `student_data_p2a.csv` and `student_data_p2b.csv` contain their initial content:

`student_data_p2a.csv`:

Name,Role
Alice,Developer
Bob,Manager

`student_data_p2b.csv`:

Name,Role
Charlie,Designer
Diana,Tester

### Example 1: Successful Merge

*Test Command:* `python3 csvtool_p2.py -merge student_data_p2a.csv student_data_p2b.csv student_merged_output.csv`

*Your Function's Behavior:* Must create the `student\_merged\_output.csv` file with the correct content and produce no output.

*Final Output on Screen (printed by the provided tool):*

[No output is produced]
-------------------------

### Example 2: Header Mismatch Error

*Scenario:* You have changed the header in `student_data_p2b.csv` to `Name,JobTitle`.

*Test Command:* `python3 csvtool_p2.py -merge student_data_p2a.csv student_data_p2b.csv out.csv`

*Your Function's Behavior:* Must raise `ValueError("Headers do not match between files")`.

*Final Output on Screen (printed to stderr by the provided tool):*

ValueError: Headers do not match between files
--

## How to Test Your Code Locally

When you click the "Run" button, the `p2.py` script executes the tests from `p2_input.txt`. After your function runs successfully, the script will show you a **preview** of the file that was created. This is how you verify your merge worked correctly. This preview is only for your debugging and is not part of the program's actual output.

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

1. **Objective:** Test if your function can correctly merge two simple files.

2. **Step 1: Modify** `student_data_p2a.csv`. Change its content to:

ID,Item
1,Apple

3. **Step 2: Modify** `student_data_p2b.csv`. Make sure its header matches and add different data:

ID,Item
2,Banana

4. **Step 3: Modify** `p2_input.txt`. Set up a single command to merge these files into a new file called `merged_test.csv`:

1
<code>python3 csvtool_p2.py -merge student_data_p2a.csv student_data_p2b.csv merged_test.csv</code>

5. **Step 4: Click "Run"**. Since a successful merge produces no direct output, the runner script helps you by showing a preview of the resulting file. The complete output you see on screen should be:

Executing: <code>python3 csvtool_p2.py -merge student_data_p2a.csv student_data_p2b.csv merged_test.csv</code>
--- Output from your program ---
[No output is produced]
-----
===== PREVIEW of created file: 'merged_test.csv' =====
ID,Item
1,Apple
2,Banana
===== END OF PREVIEW =====

## Notes:

- You will write your code in `csvtool_p2.py`. Do not modify any other files.

- You **CANNOT** use Python's built-in `csv` module.
- You can assume that all input CSV files are well-formed. This means every row in a given file will have the same number of columns as its header.

## Requested files

### `csvtool_p1.py`

```

1  #!/usr/bin/env python3
2
3  # You may only add code inside the get_value function.
4  # You are NOT allowed to use the csv module for this exercise.
5
6  from lab_utils import run_csvtool_p1_main
7
8  # ===== WRITE YOUR FUNCTION IN THE SPACE BELOW =====
9
10 #
11 # Refer to the Problem 1 description for the EXACT specifications.
12 #
13 # =====
14
15 # TODO: Write your get_value function here.
16
17
18
19
20
21
22
23
24
25 # =====
26 # ===== DO NOT MODIFY THE CODE BELOW THIS LINE =====
27 # =====
28
29 if __name__ == '__main__':
30     # This will fail until you have defined the get_value function.
31     run_csvtool p1 main(get_value func=get_value)

```

### `csvtool_p2.py`

```

1  #!/usr/bin/env python3
2
3  # You may only add code inside the merge_csv function.
4  # You are NOT allowed to use the csv module for this exercise.
5
6  from lab_utils import run_csvtool_p2_main
7
8  # ===== WRITE YOUR FUNCTION IN THE SPACE BELOW =====
9
10 #
11 # Refer to the Problem 2 description for the EXACT specifications.
12 #
13 # =====
14
15 # TODO: Write your merge_csv function here.
16
17
18
19
20
21
22
23
24 # =====
25 # ===== DO NOT MODIFY THE CODE BELOW THIS LINE =====
26 # =====
27
28 if __name__ == '__main__':
29     # This will fail until you have defined the merge_csv function.
30     run_csvtool p2 main(merge_csv func=merge_csv)

```

### `p1_input.txt`

```

1 2
2  python3 csvtool_p1.py -get student_data_p1.csv 0 1
3  python3 csvtool p1.py -get student data p1.csv 2 2

```

### `p2_input.txt`

```

1 1
2  python3 csvtool p2.py -merge student data p2a.csv student data p2b.csv student merged output.csv

```

### `student_data_p1.csv`

```

1 HeaderA,HeaderB,HeaderC
2 Data1A,Data1B,Data1C
3 Data2A,Data2B,Data2C

```

### `student_data_p2a.csv`

```

1 Name,Role
2 Alice,Developer
3 Bob,Manager

```

**student\_data\_p2b.csv**

1 Name,Role
2 Charlie,Designer
3 Diana,Tester

**p1.py**

```

1 import subprocess
2 import shlex
3
4 ##### Do Not Change This File #####
5 # This file implements a caller function for the csvtool utility.
6 # It is used by the exercises and must not be modified.
7
8 def csvtool_caller(command: str) -> str:
9     """Execute csvtool command."""
10    try:
11        # Use shlex.split to handle arguments correctly
12        args = shlex.split(command)
13        result = subprocess.run(
14            args,
15            capture_output=True,
16            text=True,
17            timeout=2 # Add a timeout for safety
18        )
19        if result.returncode != 0:
20            # Errors from csvtool.py are printed to stderr
21            return result.stderr.strip()
22        return result.stdout.strip()
23    except subprocess.TimeoutExpired:
24        return "Error: Command timed out"
25    except Exception as e:
26        return f"Error: Failed to execute command - {str(e)}"
27
28 def solution(inp):
29     return csvtool_caller(inp)
30
31 def process_input(filename):
32     lines = open(filename, 'r').readlines()
33     lines = [line.strip() for line in lines if line.strip()]
34     num_tests = int(lines[0])
35     input_tests = [lines[t].strip() for t in range(1, num_tests + 1)]
36     return input_tests
37
38 if __name__ == "__main__":
39     try:
40         Input = process_input('p1_input.txt')
41         for command in Input:
42             print(f"Executing: {command}")
43             output = solution(command)
44             print(f"--- Tool Output ---")
45             print(output if output else "[No output produced]")
46             print("-----\n")
47     except FileNotFoundError:
48         print("p1_input.txt not found.")

```

**p2.py**

```

1 import subprocess
2 import shlex
3 import os
4
5 ##### Do Not Change This File #####
6 # This file implements a caller function for the csvtool utility.
7 # It is used by the exercises and must not be modified.
8
9 def csvtool_caller(command: str) -> str:
10     """Execute csvtool command."""
11     try:
12         # Use shlex.split to handle arguments correctly
13         args = shlex.split(command)
14         result = subprocess.run(
15             args,
16             capture_output=True,
17             text=True,
18             timeout=2 # Add a timeout for safety
19         )
20         if result.returncode != 0:
21             # Errors from csvtool.py are printed to stderr
22             return result.stderr.strip()
23         return result.stdout.strip()
24     except subprocess.TimeoutExpired:
25         return "Error: Command timed out"
26     except Exception as e:
27         return f"Error: Failed to execute command - {str(e)}"
28
29 def solution(inp):
30     return csvtool_caller(inp)
31
32 def process_input(filename):
33     lines = open(filename, 'r').readlines()
34     lines = [line.strip() for line in lines if line.strip()]
35     num_tests = int(lines[0])
36     input_tests = [lines[t].strip() for t in range(1, num_tests + 1)]
37     return input_tests
38
39 # --- MODIFIED SECTION FOR LOCAL TESTING AND FILE PREVIEW ---
40 if __name__ == "__main__":
41     try:
42         Input = process_input('p2_input.txt')
43         for command in Input:
44             print(f"Executing: {command}")
45
46         # Execute the student's command and get the direct output
47         program_output = solution(command)
48
49         # Print the output from their program clearly
50         print(f"\n--- Output from your program ---\n{program_output}\n-----\n")
51
52         # --- File Preview Logic ---
53         # This part runs AFTER the student's code to show the result
54         try:
55             parts = shlex.split(command)
56             # Check if this was a merge command
57             if '-merge' in parts:
58                 # The destination file is the last argument
59                 dest_filename = parts[-1]
60
61             print(f"===== PREVIEW of created file: '{dest_filename}' =====")
62             if os.path.exists(dest_filename):
63                 with open(dest_filename, 'r') as f:
64                     content = f.read()
65                     if content:
66                         print(content.strip())
67                     else:
68                         print("[ The file was created but is empty. ]")
69             else:
70                 print(f"[ File '{dest_filename}' was not found. It may not have been created due to an error. ]")
71             print("===== END OF PREVIEW =====\n")
72
73         except Exception as e:
74             # This is a fallback in case parsing the command fails
75             print(f"[ Could not generate file preview due to an error: {e} ]\n")
76
77     except FileNotFoundError:
78         print("p2_input.txt not found. Cannot run local tests.")
79     except Exception as e:
80         print(f"An unexpected error occurred in the runner script. {e}\n")

```

[VPL](#)