Introduction to Programming
1st Project

Programming Group
guido.salvaneschi@unisg.ch

University of St. Gallen – October 17th, 2024

Declaration of Authorship

“I hereby declare

- that I have solved this project without any help from others and without the use of documents and aids other than explicitly stated;
- that I have mentioned all the sources used and that I have cited them correctly according to established academic citation rules;
- that I have acquired any immaterial rights to materials I may have used such as images or graphs, or that I have produced such materials myself;
- that I will not pass on copies of this work to third parties or publish them without the University’s written consent;
- that I am aware that my work can be electronically checked for plagiarism and that I hereby grant the University of St. Gallen copyright in accordance with the Examination Regulations in so far as this is required for administrative action;
- that I am aware that the University will prosecute any infringement of this declaration of authorship and, in particular, the employment of a ghostwriter, and that any such infringement may result in disciplinary and criminal consequences which may result in my expulsion from the University or my being stripped of my degree”

By submitting a solution to this project, you confirm through conclusive action that you are submitting the Declaration of Authorship, that you have read and understood it, and that it is true.

Warning: You must work individually and without the help of any other person on this project. All submissions will be checked for plagiarism. If we suspect plagiarism, we may ask you for a clarification meeting and a personal code presentation. Found plagiarism leads to failing the entire exam.

Please follow HSG regulations on written work, including DDoS Section 5 `https://erlasse.unisg.ch/lexoverview-home/lex-II_B_1_20`.

Submission deadline: October 31, 2024, 16:00.

Task 1:

Implement the program specified below in a module file `tsv_filter.py` and submit it before the deadline to codePost. Start from the provided Implementation Scaffold and implement the specified functionality in it. Add further functions to structure your code, but do not remove or rename the ones given in the scaffold. Make sure your program, i.e., its `main()` function, is only executed when the module is directly executed, not when it is imported in another Python program. Only submit your `tsv_filter.py` file.

Make sure your code is of high quality, i.e., it uses Python adequately, is well-structured and formatted, uses self-explaining identifier names, and contains reasonable comments that concisely document all non-trivial design decisions. According to the published grading scheme, your implementation will be assessed for functionality (50%), compliance (20%), and style (30%).

Info: Every time you upload, the submission platform runs a basic check on your submission to verify that it is compatible. For this, it calls each function of the scaffold with simple values and performs a basic check on the returned value. For your personal understanding, we provide these basic tests in `test_tsv_filter_scaffold.py`. All of these tests are successful if you submit the scaffold as we provide it to you, and all of them should still be successful after you completed the implementation. Make sure these tests work on codePost to ensure compatibility. We recommend uploading and testing your submission every now and then. You can update your submission until the deadline. Only the final submission will be considered for grading.

Note that these tests only cover the *basic* scaffold for your early feedback. They are *not* indicative of the fact that the application fully implements the functionalities required for the project. To grade your submission, we will run additional tests and further verify that the application implements all the requirements presented in this document.

Warning: Submissions that are not compatible with the submission platform are treated as missing submissions. If you should face technical issues with the submission platform, contact the TAs as early as possible. The TAs are not obliged to respond shortly before the deadline nor to fix non-systemic technical issues, i.e., issues that are not caused by the platform or its configuration.

Employee Management System

In a busy company with many employees, the HR team has a problem. They are managing hundreds of employees in a Tab-separated values (TSV) file, and it is becoming too hard to keep up. They have to search through long rows and columns just to find basic information. The company now turns to you, to help the company out of this situation. Your mission is to develop a new Employee Management System that simplifies handling employee records and makes data filtering fast and easy.

TSV is a common format to store structured, tabular datasets in files. This program implements a tool that allows users to generate an output TSV file that contains only a subset of an input TSV file's columns and only records with specific values (at the end of this document, we provide Example Executions and Example Execution: Empty Dataset that depict possible expected interactions of users with the program).

1. The program asks the user for the filenames of the input and the output TSV files. If the input file does not exist, a `FileNotFoundError` is raised. If the output file exists or is an existing directory, a `FileExistsError` is raised. Invalid files cause a `ValueError` that clearly states at least one of the format violations. The expected TSV File Format is described below.
2. If the input file has columns, the program prints the column names and asks which of these columns shall be excluded in the output file. No columns is valid, too. If the input file has records, the column types are displayed in the overview. If the user's answer is invalid, the input prompt is repeated until an answer in the correct format is given.
3. If the input file has records, the program shows the column names and types and asks for which of these columns the records shall be filtered. No columns is valid, too. If the user's answer is invalid, the input prompt is repeated until an answer in the correct format is given.
4. For each of the selected filter columns:
   (a) The program shows all distinct column values in the input file (sorted alphabetically or numerically increasing, `false` before `true`).
   (b) The program asks for which column values a record should be excluded in the output. At least one value has to be selected. If the user's answer is invalid, the input prompt is repeated until an answer in the correct format is given.
5. The program displays the chosen configuration and generates the output file.

TSV File Format

The program supports only TSV files adhering to the following format. Example Executions and Example Execution: Empty Dataset below provide examples for valid TSV files.

1. The first line of the file is the header. The header is followed by zero or more records.
2. In both header and records, columns are separated by a tab character.
3. The header and all records must have the same number of columns.
4. The header contains the columns' names. Column names contain one or more characters except for tab characters and linebreaks.
5. Column names must be distinct.
6. Each record starts with a new line.
7. Records contain a value for each column. Column values must be of one of these types:

   boolean `true` or `false`.

integer Positive and negative values may start with a minus character, are followed by one or more digits.

string String values start and end with a double quote character. In between, they contain zero or more characters, but double quotes and backslashes must be escaped with a backslash. For example, the string `"Hi", I said \\/` is encoded as `"\"Hi\", I said \\\\/"`.

8. The columns' types must be the same for every record.

> **Info:** When opening a file in text mode, Python represents linebreaks by the line feed (LF) character '`\n`' by default. Tab characters are represented by '`\t`'.

Implementation Constraints

The program has to adhere to the following constraints:

1. In the successful case, the terminal interaction looks exactly like the examples below.
2. In the failing cases, a suitable error (as described in Employee Management System) is raised as early as possible.
3. The input file is only read once.
4. The program iterates at most twice over all records in the input dataset.
5. The program only uses Python features that were already discussed in the lecture or are functions in the standard library that do not require additional imports. This explicitly *excludes* dictionaries, sets, and classes, which will be introduced later in this course. If you should be uncertain whether a specific Python feature violates this constraint, please ask the TAs for clarification.
6. The only allowed import is `import os`.

Implementation Scaffold

Your implementation of the `tsv_filter.py` module must implement the following functions. Other modules have to be able to import them from your module file and use them as described below. You may add additional functions to your module to structure your code and all functions below definitely may call other functions to implement their functionality. As reference and starting point, use the provided `tsv_filter_scaffold.py` file, which defines all functions as required with a boilerplate implementation.

1. `main()` is the entry point into the program and runs its full implementation. Returns `None`.
2. `read_input_file(input_file)` reads the TSV file with the name `input_file`. If successful, it returns the a tuple of four lists: (1) column names, (2) column types, (3) distinct column values per column, and (4) the records. On any error or invalid format (cf. TSV File Format), it raises an exception (cf. Employee Management System). Example return value on success:
   ```
   ['EmployeeID', 'Name', 'Department', 'Role', 'Salary', 'RemoteWorker'], ['integer', 'string', 'string',
   'string', 'integer', 'boolean'], [['1001', '1002', '1003'], ['"Jane Smith"', '"John Doe"', '"Michael Brown"'],
   ['"Engineering"', '"Marketing"', '"Sales"'], ['"Marketing Specialist"', '"Sales Executive"', '"Software
   Engineer"'], ['62000', '72000', '85000'], ['false', 'true']], [['1001', '"John Doe"', '"Engineering"',
   '"Software Engineer"', '85000', 'true'], ['1002', '"Jane Smith"', '"Marketing"', '"Marketing Specialist"',
   '62000', 'false'], ['1003', '"Michael Brown"', '"Sales"', '"Sales Executive"', '72000', 'true']]
   ```
3. `get_unique_integers_from_user(description, max_value, min_length)` asks the user to input a comma-seperated list of unique integers between 1 and the given maximum value `max_value` with the given minimum length `min_length`. The input prompt should display the string given in `description`. The function

returns the list of integers and repeatedly asks the user upon invalid input. For example, calling the function with `get_unique_integers_from_user("Numbers of the columns to exclude separated by commas: ", 5, 4)` returns `[5, 4, 1, 3]` after the following interaction:

```
Command Line
Numbers of the columns to exclude separated by commas: 5,4
Invalid answer. Please retry.
Numbers of the columns to exclude separated by commas:
Invalid answer. Please retry.
Numbers of the columns to exclude separated by commas: 5,4,1,3,4
Invalid answer. Please retry.
Numbers of the columns to exclude separated by commas: 5,4, 1,3
Invalid answer. Please retry.
Numbers of the columns to exclude separated by commas: 6,4,1,3
Invalid answer. Please retry.
Numbers of the columns to exclude separated by commas: 5,4,1,3
```

4. `check_output_file(output_file)` checks that a string provided as `output_file` is a valid output file name. If this is not the case, it raises an exception (cf. Employee Management System). Returns `None`.

5. `write_output_file(output_file, column_names, records, output_columns, filters)` writes the TSV file with the filename `output_file`. The column names are defined by `column_names`, the records by `records`, and `output_columns` is a list of the column indexes to include. `filters` is a list of filters, each being a pair of a column index and a list of corresponding values to exclude. The function returns `None`. For example, in the first execution demonstrated in Example Executions, the function is called as
`write_output_file("employees-filtered.tsv", ['EmployeeID', 'Name', 'Department', 'Role', 'Salary', 'RemoteWorker'], [['1001', '"John Doe"', '"Engineering"', '"Software Engineer"', '85000', 'true'], ['1002', '"Jane Smith"','"Marketing"', '"Marketing Specialist"', '62000', 'false'], ['1003', '"Michael Brown"', '"Sales"','"Sales Executive"', '72000', 'true']], [3, 4, 5], [(6, ['false']), (3, ['"Engineering"', '"Sales"'])])` and produces the file `employees-filtered.tsv`.

6. `print_operations(column_names, output_columns, filters)` prints a summary of the copying and filtering operations to be applied to generate the TSV file for the given `column_names`, the column indices provided as `output_columns` to be copied and the `filters`, each being a pair of a column index and a list of corresponding values to exclude. The function returns `None`. For example, in the first execution demonstrated in Example Executions, the function is called as
`print_operations(['EmployeeID', 'Name', 'Department', 'Role', 'Salary', 'RemoteWorker'], [3, 4, 5], [(6, ['false']), (3, ['"Engineering"', '"Sales"'])])`
and prints:

```
Command Line
Processing 'Department', 'Role' and 'Salary' where
'RemoteWorker' != false
'Department' != "Engineering" or "Sales"
```

Example Execution: Empty Dataset

A successful interaction with the program on an empty file `empty.tsv` shall result in the following output in the terminal and an empty output TSV file `empty-output.tsv`:

Command Line

```
Please enter the input TSV filename: empty.tsv
Checking input file... Success!

Please enter the output TSV filename: empty-output.tsv

Processing
```

Example Executions

Successful interactions with the program on the provided `employees.tsv` and `employees-2.tsv` files shall result in the following outputs in the terminal and the output TSV files `filtered-employees.tsv` and `filtered-employees-2.tsv`:

Command Line

```
Please enter the input TSV filename: employees.tsv
Checking input file... Success!

Please enter the output TSV filename: filtered-employees.tsv

The TSV file has the following columns:
1. EmployeeID (integer)
2. Name (string)
3. Department (string)
4. Role (string)
5. Salary (integer)
6. RemoteWorker (boolean)
Numbers of the columns to exclude separated by commas: Hello!
Invalid answer. Please retry.
Numbers of the columns to exclude separated by commas: 6,2,1

The TSV file has the following columns:
1. EmployeeID (integer)
2. Name (string)
3. Department (string)
4. Role (string)
5. Salary (integer)
6. RemoteWorker (boolean)
Numbers of the columns to filter for separated by commas: 6,3

'RemoteWorker' contains the following values:
1. false
2. true
Numbers of the values to exclude for separated by commas:
Invalid answer. Please retry.
Numbers of the values to exclude for separated by commas: 1

'Department' contains the following values:
1. "Engineering"
2. "Marketing"
3. "Sales"
Numbers of the values to exclude for separated by commas: 3,1

Processing 'Department', 'Role' and 'Salary' where
'RemoteWorker' != false
'Department' != "Engineering" or "Sales"
```

Command Line

```
Please enter the input TSV filename: employees-2.tsv
Checking input file... Success!

Please enter the output TSV filename: filtered-employees-2.tsv

The TSV file has the following columns:
1. EmployeeID (integer)
2. Name (string)
3. Department (string)
4. Role (string)
5. Salary (integer)
6. RemoteWorker (boolean)
Numbers of the columns to exclude separated by commas: 2,2
Invalid answer. Please retry.
Numbers of the columns to exclude separated by commas: 1,3,4,5,2

The TSV file has the following columns:
1. EmployeeID (integer)
2. Name (string)
3. Department (string)
4. Role (string)
5. Salary (integer)
6. RemoteWorker (boolean)
Numbers of the columns to filter for separated by commas:

Processing 'RemoteWorker'
```