In the first part of this practice project, you implemented methods that read and write cancer-risk data stored in a CSV file. In the second part of this project, you will implement two methods that are useful in processing this data. You are encouraged to complete this **optional** practice project before you attempt the required project for this week. Remember that, if you are struggling, you are welcome to examine our solution for the practice project provided at the end of the description.

Practice Project: Processing Cancer-Risk Data

The data that we will process in the practice project was generated in 2005 by the Environmental Protection Agency as part of an effort to understand the affect of air toxics on human health. The specific county-level data on cancer-risk from air toxics is stored in an .xls file located here. As part of our initial processing of this data, we have downloaded and manually removed some of the extra text from the data set. This processed CSV file which will be a critical component of our remaining practice projects is available on Google Storage here.

The structure of the cancer-risk CSV file

The data file provided above for this project contains a 2D table of data whose rows are indexed by the states and counties of the United States and whose corresponding columns contain the following information:

- Column A - State name
- Column B - County name
- Column C - Unique five-digit FIPS county county code
- Column E - Population
- Column L - Lifetime cancer-risk

To familiarize yourself further with this data set, we suggest that you download the CSV file and examine its contents using an tool like Excel.

Provided functions

For this week's part of the practice project, we have provided a starting template that includes a solution to the previous week's practice project. In particular, this template includes our implementation of the following two functions:

- **`read_csv_file(file_name)`** - Given a file path specified as the string, **`file_name`** , load the associated CSV file and return a nested list whose entries are the fields in the CSV file. Each entry in the returned table should be of type **`str`**
- **`write_csv_file(csv_table, file_name)`** - Given a nested list **`csv_table`** and a file path specified as the string, **`file_name`**, write entries in the nested list as the fields of a comma-separated CSV file with the specified path.

Required functions

Your task in part 2 of this practice project is to implement two functions that manipulate 2D tables of data (represented as nested lists). These functions are

- `select_columns(my_table, col_indices)` - Given a nested list `my_table` and a list of integers `col_indices`, return a new 2D table (as a nested list) consisting of those items in the specified columns.
- `sort_by_column(my_table, col_idx)` - Given a nested list `my_table` and an integer `col_idx` , **mutate** `my_table` by sorting its rows such that items in the column specified by `col_idx` are in **descending** order when interpreted as numbers. (You may assume that this function is only applied to columns whose entries correspond to numbers.)

Testing your functions

As always, you should test your functions to make sure that they process the 2D tables correctly. To help you with this task, we have provided part of a function `test_part2_code()`. This function reads a small CSV file test_case.csv and then uses `print_table()` to display the results in the console. The expected output from the first several tests appear in the comments at the end of the template.

To apply your code to the cancer-risk data, the second part of `test_part2_code()` loads the cancer-risk data, selects the relevant data in columns A, B, C, E, and L, and writes this data to the file `"cancer_risk_trimmed.csv"` that you will use in the practice projects for the last course in this specialization.

To test your implementation of the required functions further, we suggest that you download our version of the trimmed csv file (cancer_risk_trimmed_solution.csv) and add code to `test_part2_code()` that compares your version with our version. Once you are satisfied with the correctness of your code, you are welcome to compare your solution to our solution that is available here.

Hints for the project

- Remember `select_columns()` returns a new table while `sort_by_column()` mutates its input table.
- Your implementation of `sort_by_column()` should use the `sort()` method with an appropriate `key` function. This key function can be defined using `lambda` in a single line of code.

Mark as completed