Homework 7 — Savvy School Counselor

Due Tuesday, October 30th at 11:55pm

(50 points)

Objectives:

- Implement a program that reads formatted data from a file, process it, and write the results to a new file.
- Use what you have learned about designing modular programs (programs that use functions) to design your program's layout.
- Become familiar with lists and list manipulation.

Turn In:

- `hw7_savvycounselor.ipynb`

Turn in this file on moodle under the "Homework 7 - Savvy School Counselor" link.

Provided Files:

- provided example input files are in the `student_files/` directory, example outputs are in the `answers/` directory

You may not use dictionaries on any part of this homework.

You must have an equal number of `open()`s as `close()`s.

Instructions:

You are a counselor working at George Washington High School in Illinois. Over the years you have prided yourself on the quality of education provided to your students, and you have always tried to make sure that your students are met with the best care possible. You are so dedicated, in fact, that you have talked with every one of your students and have determined goals for them to meet during the year.

Every week you check your student's grades to make sure that they are doing well across all of their subjects. This has worked wonderfully over the past few years, and you have seen a remarkable improvement in the performance of your students. However, there is now a problem. Due to budget cuts, the school had to let one of the other councilors go, and the number of students that you are responsible for has nearly doubled! You have tried to keep up with all of their grades, but it has simply been too much.

But there is light in the darkness! You remember that you took a programming course in college, and you set out to build a program that will generate a report to tell you whose parents you need to call!

What is given:
Currently, all of the data on your students is stored in the `students_[size].csv` files (the .csv extension means comma-separated-value file). Each line of data contains the student's last name, first name, and their **current grade** for each of the given subjects.

In the header of the file (the first row in the table), you will find the following fields:
`lastname,firstname,[subj1],[subj2],[subj3],[subj4],...,[subjN]`

Notes: When you read a line in from a file, you read it as a string. You will not want to treat all of the column values as strings though!

You must design your program such that it can accomodate processing a file with scores for just one subject or with scores for many subjects.

Your task:
First, your program will prompt the user for three things using the `input()` function:

1) the input file name
2) the output file name
3) the objective grade

Then, you need to generate a report that tells you two things about the students that are having difficulties achieving the grades they want:

1. Which courses do you need to be concerned about?
   - Courses are concerning if the student's current grade is within 3 percentage points of the given objective grade. (e.g. their current grade is 87 - 85 and the objective is 84)
2. Which courses is the student in trouble in?
   - A student is in trouble if their current grade is at or below the input objective grade for the semester.

These statuses are mutually exclusive, so courses that appear in the "concerning" status should not appear in the "trouble" status and vice versa.

Here is an example of a some students' grades in Math and English:

Input file (students_tiny.csv)	Output file (report.txt)

lastname,firstname,math,english
Marrazzo,Jeremy,81,74
Dupps,Tabitha,88,65
Farness,Christopher,88,94
	Objective grade: 80
	
	Jeremy Marrazzo:
	  Concerning:
	  math -- Current: 81
	  Trouble:
	  english -- Current: 74
	
	Tabitha Dupps:

	Trouble: english -- Current: 65

The report should contain a block for each student who has at least one “concerning” or “trouble” subject. The block should include the student’s full name on the first line, followed by a list of the “concerning” courses, followed by a list of “trouble” courses. For these courses, the report should output the current grade for the subject. Students who have no concerning or trouble courses should not appear in the list.

To get consistent formatting, you will want to use the tab character `"\t"` once before `"Trouble"` and `"Concerning"` and twice before the listed subjects.

While you do have access to the file, it can change on an almost daily basis (budget cuts!), so you must write a program that will work for a file of any length, with any number of courses.

Program Structure:
The structure of your program is up to you. You must use functions and return values, and each function should be in charge of **one and only one** task. Take a look at lab 9 for inspiration!

You may not do any of the actual calculation in your `main()` function. Your functions that do calculation must not also print results.

Output & Testing:
It is your responsibility to test your program.

You can use the difference checking code that we provide in the starter notebook or you can use an online difference checker tool (like https://text-compare.com/) to compare and contrast the contents of the provided output files with the contents of your generated reports.

If your program output does not **exactly** match the files we provide, you will not receive full credit.

We will test your code using a different input file. Those that we have provided cover most possibilities but not necessarily all possibilities.

Consult the provided example files on moodle.

Development:
Your code should always be in a “runnable” state. **Do not attempt to develop your entire program without running it.**
You should complete and test each menu item before moving on to the next one. Start by testing on the file `students_tiny.csv`, then move on to `students_small.csv`, etc.

We suggest starting with either the concerning or the trouble courses, getting them written correctly to your output file, then moving on to the other ones.

As a reference, our solution, including blank lines and comments, is about 81 lines long.

Style & Structure (5 points)
Naming (1 point)

- Your variable names should be meaningful and concise
- Your variable names should be formatted like: user_birthday (words start with lowercase letters and are separated with underscores)
- Function names should be formatted the same way as your variables: get_today() (words start with lowercase letters and are separated with underscores)

Proper Variable Usage (1 point)

- You should not have variables outside of functions.

Functions (3 points)

- You **must** have a `main` function. The only code in your program that is not inside a function should be a single call to `main()`.
  REQUIRED: use if __name__ == "__main__":
- All non-main functions **must** be defined at the zero indentation level, as in the example lecture code. Do not attempt to define a function inside another function.
- Each function must do one and only one task.
- You must use returns. (For example, if you have a function that finds the minimum value in the file, that function should not print the result, instead, you should return it.)

Running without Errors

- Make sure to click "Restart and Run All" in the kernel menu before turning your work in—you will lose points if any errors are produced!

Comments (3 points)
Your code must be commented. You must include a file comment (your name, your section, the homework name, and a brief description of what the program does), inline comments (explaining any complex code), and function comments. An example function comment is shown below:

```
# This function converts a temperature in fahrenheit to celsius
# and prints the equivalence.
# Parameters: fahrenheit - int or float degrees fahrenheit
# Return: float - equivalent degrees celsius
def fahrenheit_to_celsius(fahrenheit):
    celsius = (fahrenheit - 32) * (5/9)
    return celsius
```

Extra Credit (+5 points)

The provided input files list the students in a random order. When you generate your report, alphabetize the students by first by last name, then by first name. You will find the sort and sorted functions (https://docs.python.org/3/howto/sorting.html) helpful. Take a look at the key functions in particular. These examples work with a type called a “tuple”, but they apply to lists as well.

Hint 1: you can read your file’s lines into a list, then alphabetize the students, then use that list to generate your reports.

Hint 2: if you sort a list of lists based on index 1, then based on index 2, the end result will be sorted by 2 then by 1.