# Advanced Python

You've been hired by Stevens Institute of Technology to create a data repository of courses, students, and instructors. The system will be used to help students track their required courses, the courses they have successfully completed, their grades, GPA, etc. The system will also be used by faculty advisors to help students to create study plans. We will continue to add new features to this solution over the coming weeks so you'll want to think carefully about a good design and implementation.

Your assignment this week is to begin to build the framework for your project and summarize student and instructor data. You'll need to download three ascii, tab separated, data files:

1. students.txt (available from Canvas)

- Provides information about each student
- Each line has the format: CWID\tName\tMajor (where \t is the <tab> character)

2. instructors.txt (available from Canvas)

- Provides information about each instructor
- Each line has the format: CWID\tName\tDepartment (where \t is the <tab> character)

3. grades.txt (available from Canvas)

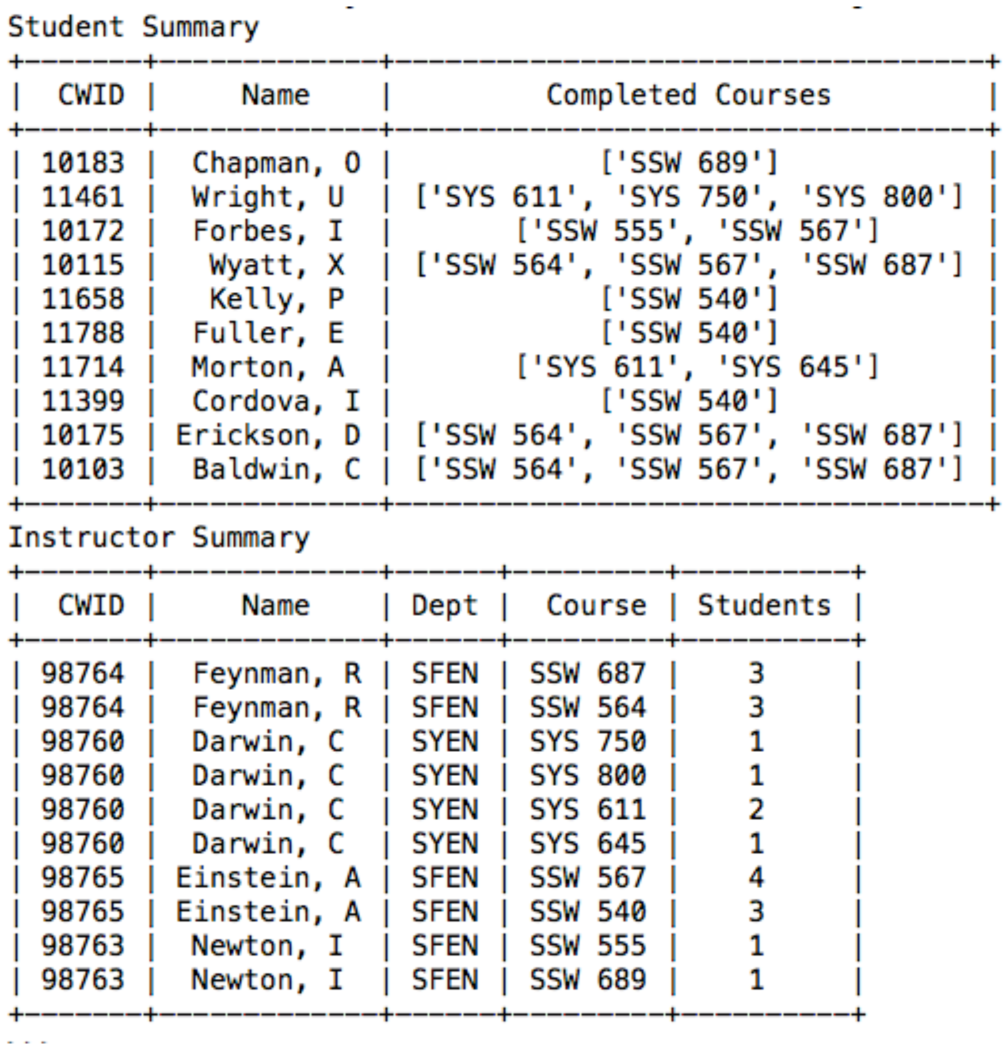
- Specifies the student CWID, course, and grade for that course, and the instructor CWID
- Each line has the format: Student CWID\tCourse\tLetterGrade\tInstructor CWID

Your assignment is to read the data from each of the three files and store it in a data structure that is easy to process to meet the following requirements:

- Your solution should allow a single program to create repositories for different sets of data files, e.g. one set of files for Stevens, another for Columbia University, and a third for NYU. Each repository will have different directories of data files.
- You may hardcode the paths of each of the required students.txt, instructors.txt, and grades.txt files. Your solution should accept the name of a directory path where the three data files exist so you can easily create multiple sets of input files for testing.

- Use your file reader generator from HW08 to read the students, instructors, and grades files into appropriate data structures or classes.
  - Generate warning messages for the user if the input file doesn't exist or doesn't meet the expected format
- Handle error conditions gracefully if the file does not exist
- Use PrettyTable to generate a summary table of all of the students with their CWID, name, and a **sorted** list of the courses they've taken (as specified in the grades.txt file).
- Use PrettyTable to generate a summary table of each of the instructors with their CWID, name, department, course they've taught, and the number of students in each class.

Here's an example of the **Student and Instructor summary tables**:



```
Student Summary
+-------+-------------+------------------------------------+
|  CWID |     Name    |          Completed Courses         |
+-------+-------------+------------------------------------+
| 10183 |  Chapman, O |             ['SSW 689']            |
| 11461 |  Wright, U  | ['SYS 611', 'SYS 750', 'SYS 800']  |
| 10172 |  Forbes, I  |       ['SSW 555', 'SSW 567']       |
| 10115 |   Wyatt, X  | ['SSW 564', 'SSW 567', 'SSW 687']  |
| 11658 |   Kelly, P  |             ['SSW 540']            |
| 11788 |  Fuller, E  |             ['SSW 540']            |
| 11714 |  Morton, A  |       ['SYS 611', 'SYS 645']       |
| 11399 |  Cordova, I |             ['SSW 540']            |
| 10175 | Erickson, D | ['SSW 564', 'SSW 567', 'SSW 687']  |
| 10103 |  Baldwin, C | ['SSW 564', 'SSW 567', 'SSW 687']  |
+-------+-------------+------------------------------------+
Instructor Summary
+-------+-------------+------+---------+----------+
|  CWID |     Name    | Dept |  Course | Students |
+-------+-------------+------+---------+----------+
| 98764 |  Feynman, R | SFEN | SSW 687 |    3     |
| 98764 |  Feynman, R | SFEN | SSW 564 |    3     |
| 98760 |  Darwin, C  | SYEN | SYS 750 |    1     |
| 98760 |  Darwin, C  | SYEN | SYS 800 |    1     |
| 98760 |  Darwin, C  | SYEN | SYS 611 |    2     |
| 98760 |  Darwin, C  | SYEN | SYS 645 |    1     |
| 98765 | Einstein, A | SFEN | SSW 567 |    4     |
| 98765 | Einstein, A | SFEN | SSW 540 |    3     |
| 98763 |  Newton, I  | SFEN | SSW 555 |    1     |
| 98763 |  Newton, I  | SFEN | SSW 689 |    1     |
+-------+-------------+------+---------+----------+
...
```

**Hints:**

- Think through the overall design and sketch out the solution using CRC Cards (from lecture 2) with all of the major functionality before you write any code
    - class **Repository** to hold the students, instructors and grades. The class is just a container to store all of the data structures together in a single place.
    - class **Student** to hold all of the details of a student, including a defaultdict(str) to store the classes taken and the grade where the course is the key and the grade is the value.
    - class **Instructor** to hold all of the details of an instructor, including a defaultdict(int) to store the names of the courses taught along with the number of students
    - Read grades file and either store the data in a list, or just process each line as you read it from the file to update the courses taken by each student and to update the courses taught by each instructor and update the number of students
    - a main() routine to run the whole thing
    - a test suite that compares known values for a directory of student, instructor, and grades files to the computed values.
    - For each student, you need to know the student's name and major. We'll be adding other student attributes in the future, e.g. email address, etc. so a flexible solution will help longer term.
    - The key to your dictionary should uniquely identify the item, e.g. using the student's name as the key is not a good choice because two students may have the same name.
    - We aren't using the grade for this assignment but future assignments may require us to consider if the student passed the course to determine if she needs to take the course again
    - We will need to calculate the GPA in the future (not for this assignment) so we need to keep the grade readily available
    - e.g. the main components in my solution include:
    - Think about how you need to access the data when choosing the keys and values for each of the dictionaries. Recall that the value may be a dictionary, set, list, etc.
    - Think about the operations that you need to perform on the data, e.g.
    - The data files have several values on each row, separated by a tab character. string.split('\t') is a convenient way to split lines into the individual fields. Careful though, because the last field in file line will have a trailing newline. You can solve that issue with line.strip().split('\t')

Test your program and upload your program to Canvas when ready. Be sure to handle unexpected cases, e.g. a student from the students.txt file who has no grades yet (she might be a first semester student). You can be sure that your testing group (Prof JR) will have some curious test cases to try against your solution.

We will continue to expand this task in the coming weeks so an elegant design is important.