CS65: Introduction to Computer Science

Assignment 4
Final Exam Review
Quiz 6

Md Alimoor Reza
Assistant Professor of Computer Science

Topics

- **Assignment 4**

- **Final Exam Review**

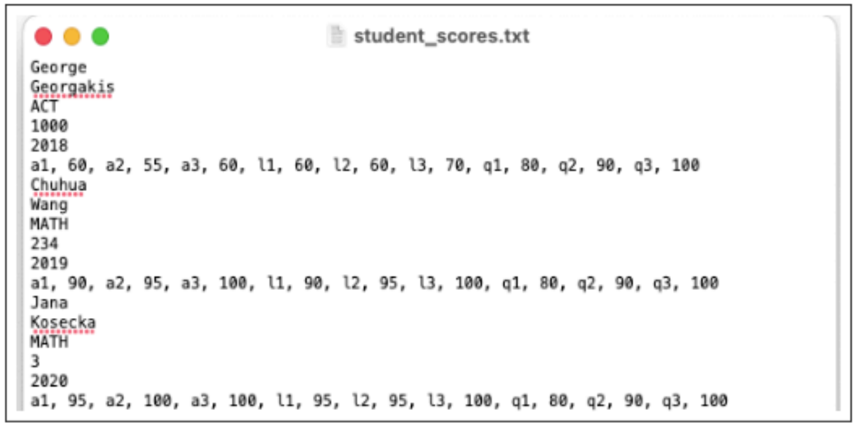
Assignment 4

Grade Calculator

You should use Class, Objects, Files, & String formatting for this task. In this task, you will write two Classes and a few other functions that can enable us to calculate students' letter grades ('A', 'B', 'C', etc) for a course.

Task 1: Read Student Records from an Input Text File

Students and their exam information should come from a text file. The specific text file named **"student_records.txt"** is given to you for this assignment. It contains information for several students enrolled in a course eg, **"CS65"**. Every consecutive six lines in the text file collectively denote a student record. The sequence of six lines denote *first name, last name, major, serial number, year of entrance,* and *9 exam names and their corresponding scores*. For example, the figure below shows the 3 student records consisting of first 18 lines in the text file. Information for the 1st student is as follows: *first name*: "George", *last name*: "Georgakis", *major*: "ACT" (acronym for Accounting), *serial number*: 1000, *year of entrance*: 2018, and *exam names and scores*: "a1, 60, a2, 55, a3, 60, l1, 60, l2, 60, l3, 70, q1, 80, q2, 90, q3, 100".



```
George
Georgakis
ACT
1000
2018
a1, 60, a2, 55, a3, 60, l1, 60, l2, 60, l3, 70, q1, 80, q2, 90, q3, 100
Chuhua
Wang
MATH
234
2019
a1, 90, a2, 95, a3, 100, l1, 90, l2, 95, l3, 100, q1, 80, q2, 90, q3, 100
Jana
Kosecka
MATH
3
2020
a1, 95, a2, 100, a3, 100, l1, 95, l2, 95, l3, 100, q1, 80, q2, 90, q3, 100
```

Assignment 4

Given this input text file of student records, you should write a function that reads the input file and prepare the individual student information for further processing. You should do the following steps:

Step 1: Open the text file using the appropriate function.

Step 2: Read all the lines using the appropriate function and save them in a variable *all_lines*.

Step 3: Process six items from *all_lines* iteratively and save them in variables *first_name*, *last_name*, *std_major*, *std_serial*, *std_year*, *scores_str*. These will be used to make a student record.

Step 4: Remove an extra newline character from the end of the string (if required).

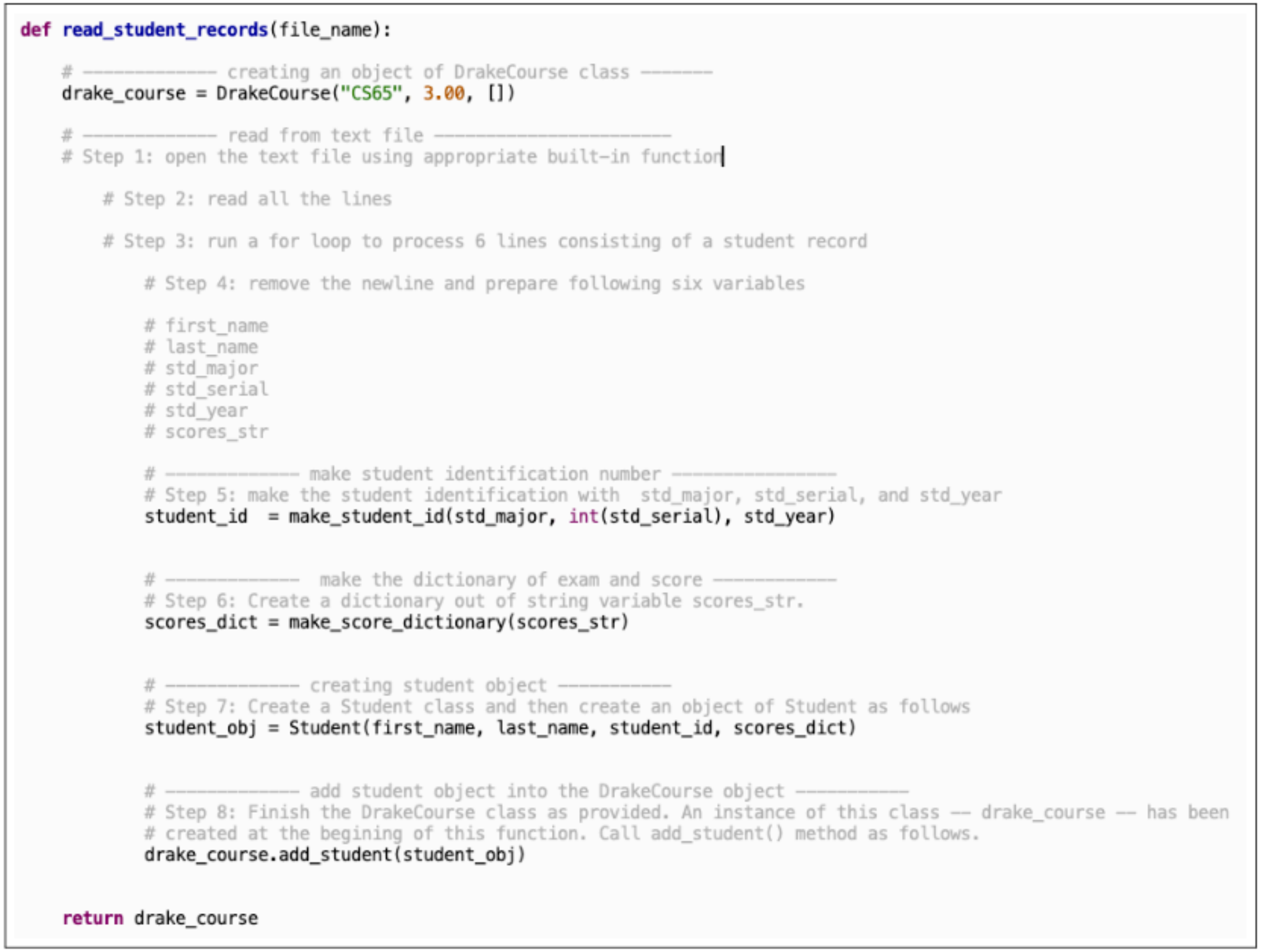
Step 5: Make the student identification number using *std_major*, *std_serial*, and *std_year*. You should call a function *make_student_id(major, serial_num, year)* to accomplish this goal, which you will be implementing in **Task 2**.

Step 6: Create a dictionary out of string variable *scores_str*. The keys of the dictionary should be exam-names and values should be exam-scores. You should call a function *make_score_dictionary(scores_str)* to accomplish this goal. **Task 3** asks you to implement this function.

Step 7: Create a Student object using variables *first_name*, *last_name*, *student_id*, *scores_dict*. In order to do so, first, you need to create a Student class, which you will be doing in **Task 4**.

Step 8: An instance of another class, DrakeCourse, has been created at the beginning of this function. Call *add_student*() method as follows. Finish the empty body of the DrakeCourse class as described in **Task 5**. Finally, this function returns object *drake_course*. I have provided the basic structure of the above steps to get you up to speed, as shown below. You need to fill in the rest.

Assignment 4

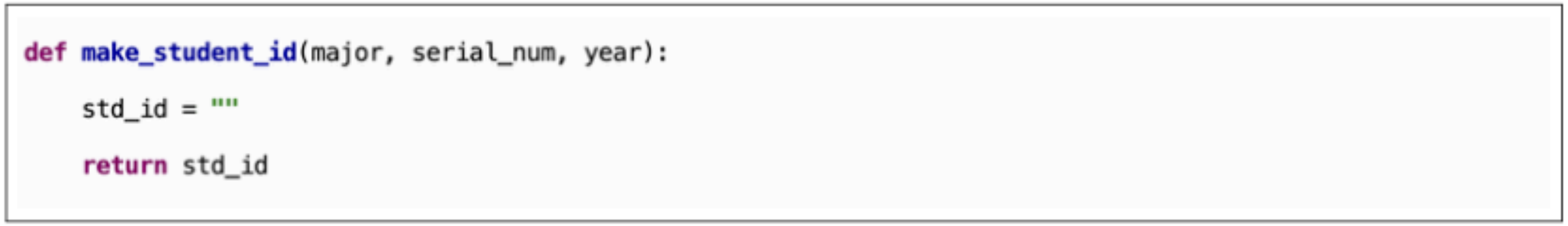


```
def read_student_records(file_name):

    # ------------- creating an object of DrakeCourse class -------
    drake_course = DrakeCourse("CS65", 3.00, [])

    # ------------- read from text file -----------------------
    # Step 1: open the text file using appropriate built-in function

        # Step 2: read all the lines

        # Step 3: run a for loop to process 6 lines consisting of a student record

            # Step 4: remove the newline and prepare following six variables

            # first_name
            # last_name
            # std_major
            # std_serial
            # std_year
            # scores_str

            # ------------- make student identification number ----------------
            # Step 5: make the student identification with  std_major, std_serial, and std_year
            student_id  = make_student_id(std_major, int(std_serial), std_year)


            # -------------  make the dictionary of exam and score ------------
            # Step 6: Create a dictionary out of string variable scores_str.
            scores_dict = make_score_dictionary(scores_str)


            # ------------- creating student object -----------
            # Step 7: Create a Student class and then create an object of Student as follows
            student_obj = Student(first_name, last_name, student_id, scores_dict)


            # ------------- add student object into the DrakeCourse object -----------
            # Step 8: Finish the DrakeCourse class as provided. An instance of this class -- drake_course -- has been
            # created at the begining of this function. Call add_student() method as follows.
            drake_course.add_student(student_obj)


    return drake_course
```

Assignment 4

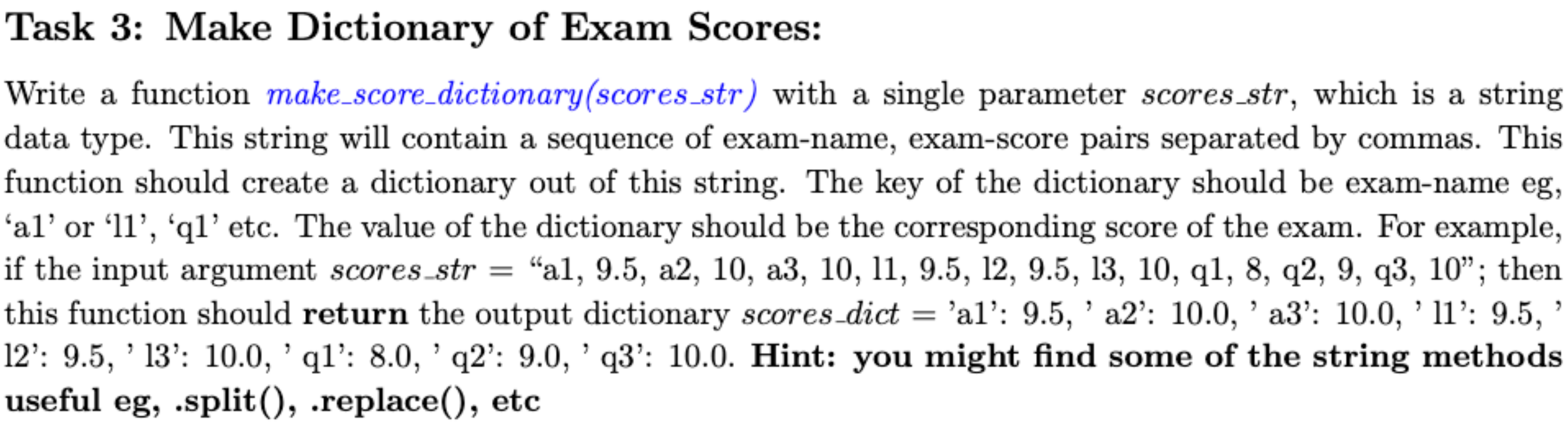
Task 2: Make Student Identification Number:

Write a function *make_student_id(major, serial_num, year)* with three parameters: 1) *major*, 2) *serial_number*, 3) *year*. The major could be "CS" or "MATH" or "ENG", "PSY", "ACC" etc. The second parameter is *serial_number* which is an integer number. Its value should be within 0 to 9999. The third parameter *year* is a string representing the incoming year eg, "2021". This function should make a student unique identification number of the format: *major-serial_number-year*. For example, if the input arguments *major* = "CS", *serial_num* = 12, *year* = "2021". This function should **return** the output of string data type "CS-0012-2021". **You should use string formatting, either % or .format().**



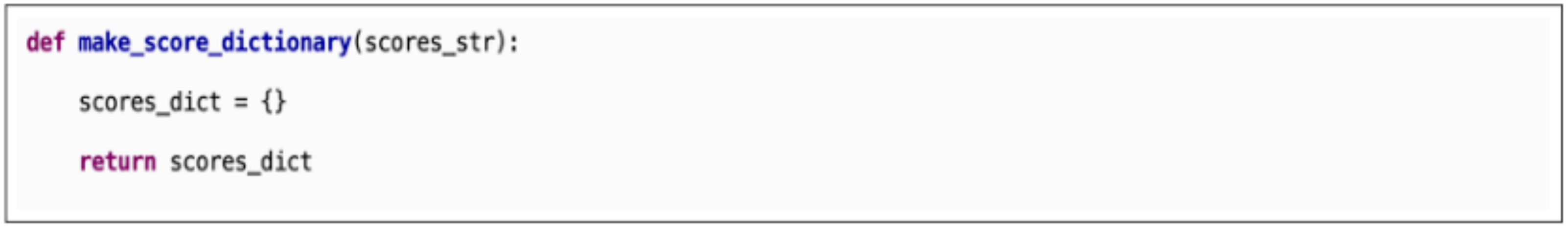
```python
def make_student_id(major, serial_num, year):

    std_id = ""

    return std_id
```

Assignment 4



Task 3: Make Dictionary of Exam Scores:

Write a function *make_score_dictionary(scores_str)* with a single parameter *scores_str*, which is a string data type. This string will contain a sequence of exam-name, exam-score pairs separated by commas. This function should create a dictionary out of this string. The key of the dictionary should be exam-name eg, ‘a1’ or ‘l1’, ‘q1’ etc. The value of the dictionary should be the corresponding score of the exam. For example, if the input argument *scores_str* = “a1, 9.5, a2, 10, a3, 10, l1, 9.5, l2, 9.5, l3, 10, q1, 8, q2, 9, q3, 10”; then this function should **return** the output dictionary *scores_dict* = ’a1’: 9.5, ’ a2’: 10.0, ’ a3’: 10.0, ’ l1’: 9.5, ’ l2’: 9.5, ’ l3’: 10.0, ’ q1’: 8.0, ’ q2’: 9.0, ’ q3’: 10.0. **Hint: you might find some of the string methods useful eg, .split(), .replace(), etc**



```
def make_score_dictionary(scores_str):

    scores_dict = {}

    return scores_dict
```

Assignment 4

Task 4: Student Class

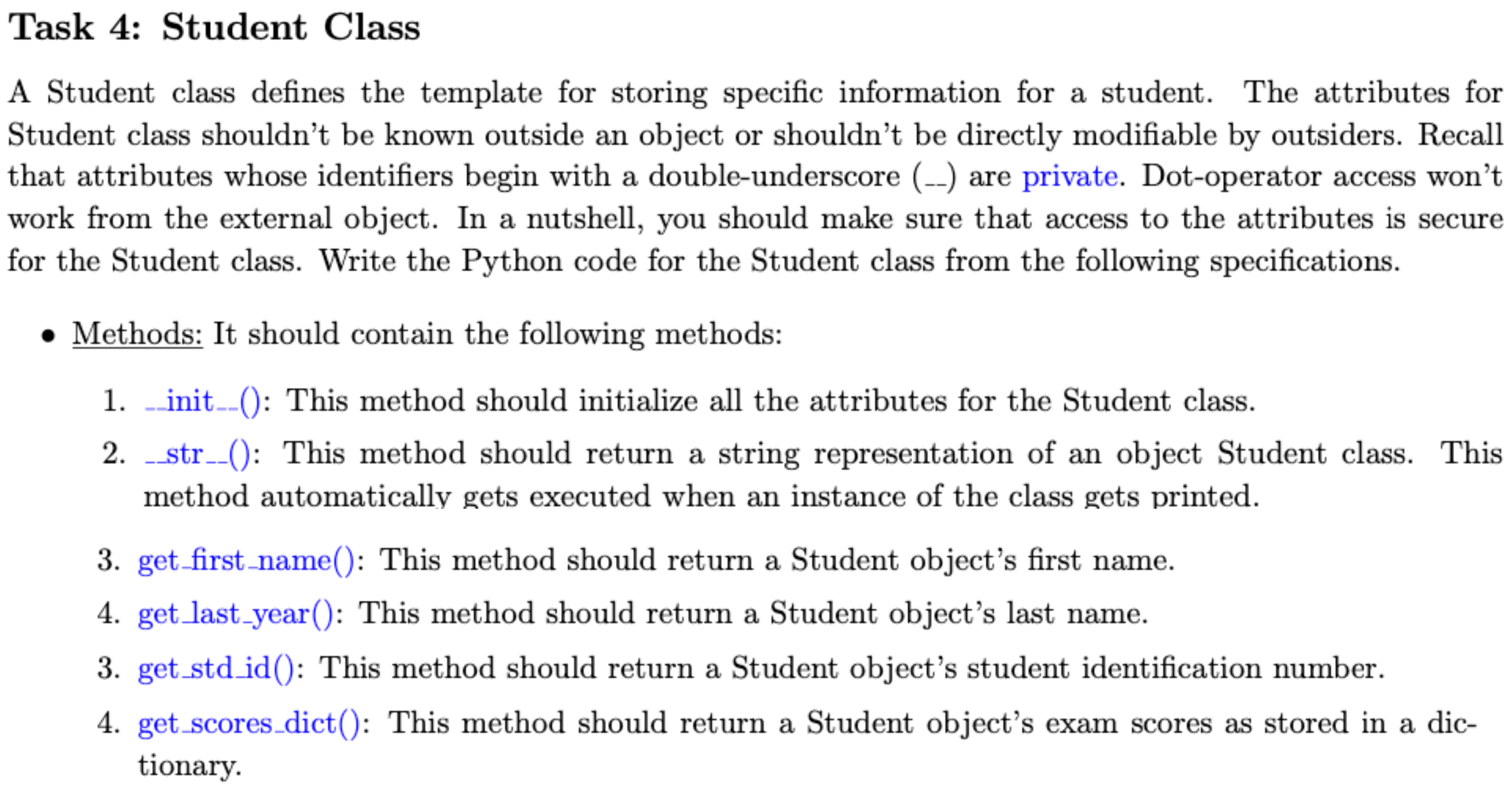
A Student class defines the template for storing specific information for a student. The attributes for Student class shouldn't be known outside an object or shouldn't be directly modifiable by outsiders. Recall that attributes whose identifiers begin with a double-underscore (__) are private. Dot-operator access won't work from the external object. In a nutshell, you should make sure that access to the attributes is secure for the Student class. Write the Python code for the Student class from the following specifications.

- Attributes: It should contain the following attributes:

    1. *first_name*. The first name of a Student object.
    2. *last_name*. The last name of a Student object.
    3. *student_id*. You can assume that a student identification number is a string data type. It will have a specific format which has been described in Task 2.
    4. *scores_dict*. A dictionary of different exam scores. The keys will be exam names (*a*1 denoting assignment 1, *l*1 denoting lab 1, etc) and values will be corresponding scores. For example, *scores_dict* could store a dictionary {'a1': 10, 'l1': 10, 'q1': 9} for a Student object.

  Use the necessary primitive/complex data types when you define the attributes.

Assignment 4



Task 4: Student Class

A Student class defines the template for storing specific information for a student. The attributes for Student class shouldn't be known outside an object or shouldn't be directly modifiable by outsiders. Recall that attributes whose identifiers begin with a double-underscore (__) are private. Dot-operator access won't work from the external object. In a nutshell, you should make sure that access to the attributes is secure for the Student class. Write the Python code for the Student class from the following specifications.

- Methods: It should contain the following methods:

  1. __init__(): This method should initialize all the attributes for the Student class.
  2. __str__(): This method should return a string representation of an object Student class. This method automatically gets executed when an instance of the class gets printed.
  3. get_first_name(): This method should return a Student object's first name.
  4. get_last_year(): This method should return a Student object's last name.
  3. get_std_id(): This method should return a Student object's student identification number.
  4. get_scores_dict(): This method should return a Student object's exam scores as stored in a dictionary.

Assignment 4

Task 5: DrakeCourse Class

A DrakeCourse class defines the template for storing specific information for a course at Drake University. Write the Python code for the DrakeCourse class from the following specifications.

- Attributes: It should contain the following attributes:

  1. *course_name*. The name of the course object of string data type e.g., "CS65", "CS67", etc.
  2. *credit*. The credit for the course object. The data type should be float point, e.g., 3.00 or 1.00.
  3. *student_list*. An attribute of *list* data type. The elements of this list are objects of the Student class. This is an example of the Object-Oriented Programming concept called **encapsulation**, where one class (i.e., DrakeCourse class) template can contain instances of another class (i.e., Student class) as attributes.

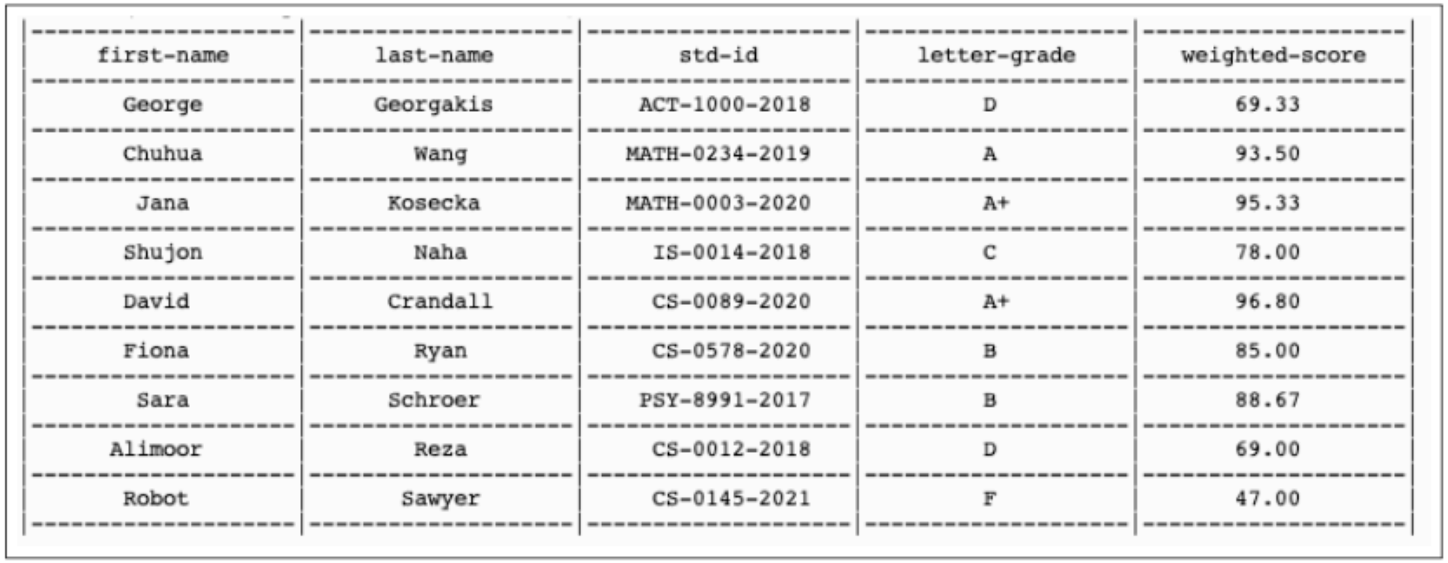
  Use the necessary primitive/complex data types when you define the attributes.

- Methods: It should contain the following methods:

  1. __init__(): This method should initialize all the attributes for the DrakeCourse class.
  2. add_student(): This method should add a new Student object into the list attribute *student_list*.
  3. show_student_details(): This method should print the Student objects from the list attribute *student_list*.
  4. calculate_grade(): The most important method of the entire assignment. This method should print each Student object's letter grade based on their exam scores. The grading rubric has been given to you inside the code. Your task is to compute the final grade from the weighted average, find the letter grade, and present the outputs using appropriate string formatting. **You should use string formatting, either % or .format()**. If done correctly, the output should be as follows:

Drake UNIVERSITY

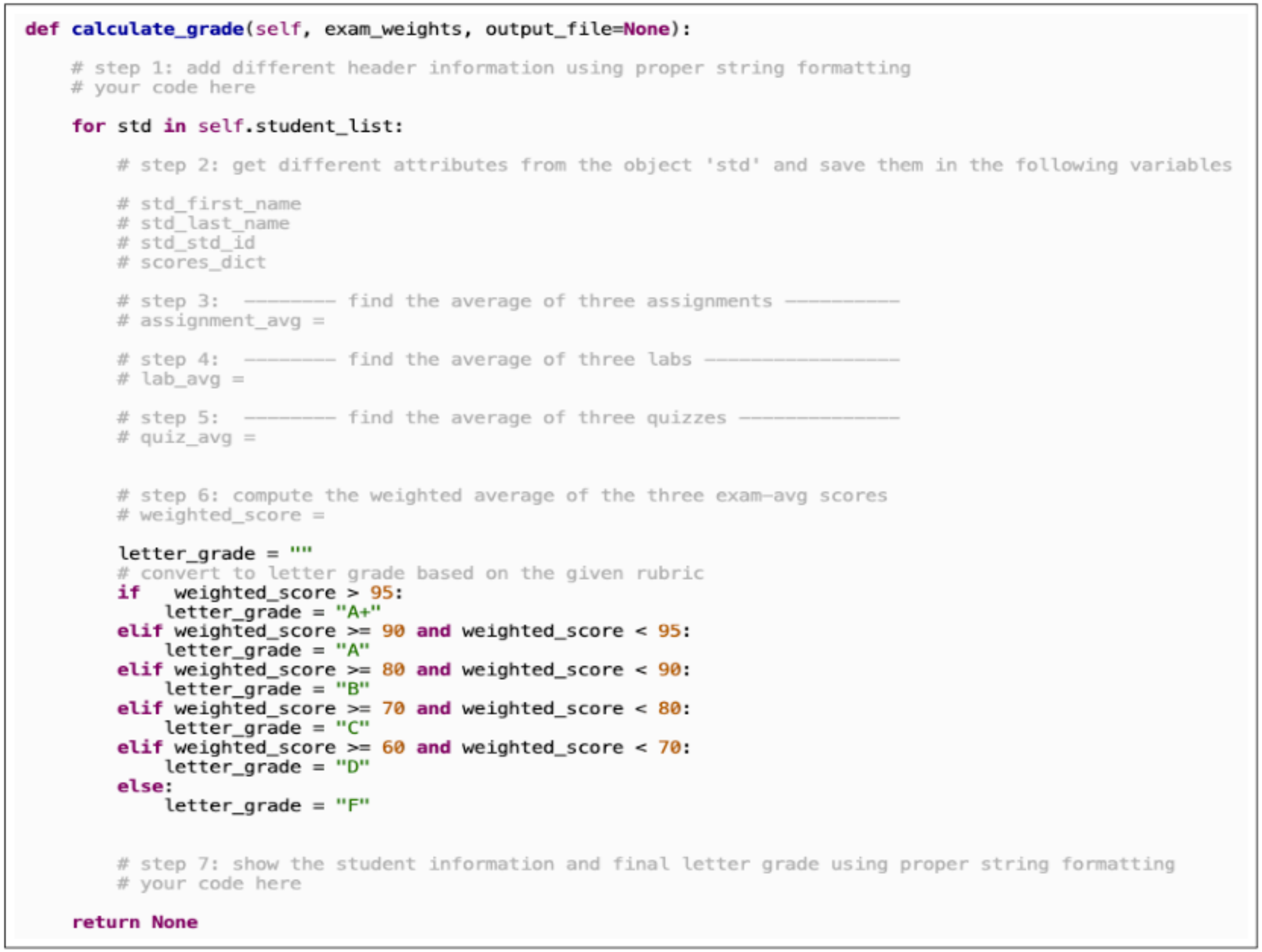
Assignment 4
Final output



```
|--------------------|--------------------|--------------------|--------------------|--------------------|
|     first-name     |     last-name      |       std-id       |    letter-grade    |   weighted-score   |
|--------------------|--------------------|--------------------|--------------------|--------------------|
|       George       |     Georgakis      |   ACT-1000-2018    |         D          |       69.33        |
|--------------------|--------------------|--------------------|--------------------|--------------------|
|       Chuhua       |        Wang        |   MATH-0234-2019   |         A          |       93.50        |
|--------------------|--------------------|--------------------|--------------------|--------------------|
|        Jana        |      Kosecka       |   MATH-0003-2020   |         A+         |       95.33        |
|--------------------|--------------------|--------------------|--------------------|--------------------|
|       Shujon       |        Naha        |    IS-0014-2018    |         C          |       78.00        |
|--------------------|--------------------|--------------------|--------------------|--------------------|
|       David        |      Crandall      |    CS-0089-2020    |         A+         |       96.80        |
|--------------------|--------------------|--------------------|--------------------|--------------------|
|       Fiona        |        Ryan        |    CS-0578-2020    |         B          |       85.00        |
|--------------------|--------------------|--------------------|--------------------|--------------------|
|        Sara        |      Schroer       |   PSY-8991-2017    |         B          |       88.67        |
|--------------------|--------------------|--------------------|--------------------|--------------------|
|      Alimoor       |        Reza        |    CS-0012-2018    |         D          |       69.00        |
|--------------------|--------------------|--------------------|--------------------|--------------------|
|       Robot        |       Sawyer       |    CS-0145-2021    |         F          |       47.00        |
|--------------------|--------------------|--------------------|--------------------|--------------------|
```

Assignment 4

The basic structure of the method is shown below:

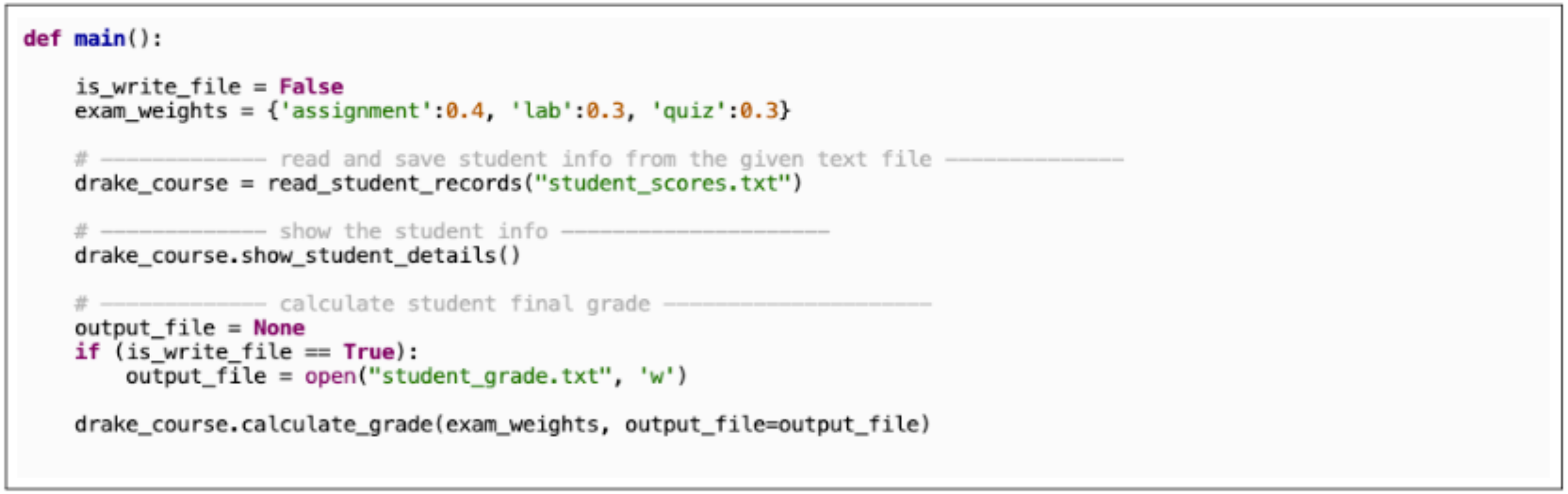


```
def calculate_grade(self, exam_weights, output_file=None):

    # step 1: add different header information using proper string formatting
    # your code here

    for std in self.student_list:

        # step 2: get different attributes from the object 'std' and save them in the following variables

        # std_first_name
        # std_last_name
        # std_std_id
        # scores_dict

        # step 3:  -------- find the average of three assignments ----------
        # assignment_avg =

        # step 4:  -------- find the average of three labs -----------------
        # lab_avg =

        # step 5:  -------- find the average of three quizzes --------------
        # quiz_avg =


        # step 6: compute the weighted average of the three exam-avg scores
        # weighted_score =

        letter_grade = ""
        # convert to letter grade based on the given rubric
        if   weighted_score > 95:
            letter_grade = "A+"
        elif weighted_score >= 90 and weighted_score < 95:
            letter_grade = "A"
        elif weighted_score >= 80 and weighted_score < 90:
            letter_grade = "B"
        elif weighted_score >= 70 and weighted_score < 80:
            letter_grade = "C"
        elif weighted_score >= 60 and weighted_score < 70:
            letter_grade = "D"
        else:
            letter_grade = "F"


        # step 7: show the student information and final letter grade using proper string formatting
        # your code here

    return None
```

Assignment 4

Task 6: Calculate the Grade

The main function handles the prepossessing of the texts after reading them from the input file. It has been given to you so that you can understand how control of flow works. Pay attention to the function calls and method calls that are happening inside the main function. The most important method call is in the last line $drake\_course.calculate\_grade(exam\_weights, output\_file = output\_file)$. Since it is a method of the DrakeCourse class, you should fill in the missing parts inside that class.



```python
def main():

    is_write_file = False
    exam_weights = {'assignment':0.4, 'lab':0.3, 'quiz':0.3}

    # -------------- read and save student info from the given text file ---------------
    drake_course = read_student_records("student_scores.txt")

    # -------------- show the student info ----------------------
    drake_course.show_student_details()

    # -------------- calculate student final grade ----------------------
    output_file = None
    if (is_write_file == True):
        output_file = open("student_grade.txt", 'w')

    drake_course.calculate_grade(exam_weights, output_file=output_file)
```

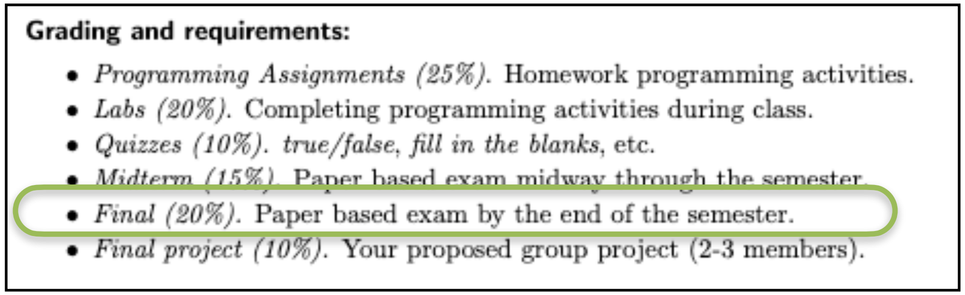
Topics

- Assignment 4

- **Final Exam Review**

Topics

- Final exam is **20%** of your total grade



Grading and requirements:

- *Programming Assignments (25%).* Homework programming activities.
- *Labs (20%).* Completing programming activities during class.
- *Quizzes (10%).* *true/false, fill in the blanks*, etc.
- *Midterm (15%).* Paper based exam midway through the semester.
- *Final (20%).* Paper based exam by the end of the semester.
- *Final project (10%).* Your proposed group project (2-3 members).

- Format will be very similar to the midterm exam
  - ➡ True/False
  - ➡ Fill in the gaps
  - ➡ Finding outputs of given Python programs
  - ➡ Writing a Python program of a given problem
    - ➡ 3-4 problems (unlike midterm which had only one problem solving challenge)

Topics

- Variables, expression
- Functions
- Scope for local and global variables

- Boolean type and boolean expression
- Selection statements are useful for branching inside your program
  - ➡ If
  - ➡ If-else
  - ➡ If-elif-…else

- The **while** loop
- The **for** loop to solve a repetitive task
  - ➡ Value **for** loop
  - ➡ Index **for** loop
  - ➡ Nested **for** loop

Topics

- String methods
- String formatting: with % operator or with **.format()**

- List manipulation
  - ➡ Adding
  - ➡ Updating
  - ➡ Removing

- Tuple

- Dictionary manipulation
  - ➡ Adding
  - ➡ Updating
  - ➡ Removing
- Iterating through dictionary elements
- Dictionary methods

Topics

- File I/O operations
    - ➡ Different modes: reading, writing, appending
    - ➡ Different methods: read(), readline(), readlines(), write()
    - ➡ Different ways of opening a file

- Class and Objects
- Difference between the two
- Class attributes and methods
- Object manipulation
    - Accessing attributes
    - Accessing methods
    - Secure access vs insecure access

- Exceptions