Assignment 9

Objects, and Binary trees, and the Table ADT

Date Due: April 2, 2018, 10pm **Total Marks: 36**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- **Assignments are being checked for plagiarism.** We are using state-of-the-art software to compare every pair of student submissions.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M.
- Make sure your name and student number appear at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- Do not submit folders, or zip files, even if you think it will help.
- Programs must be written in Python 3.
- **Assignments must be submitted to Moodle.** There is a link on the course webpage that shows you how to do this.
- **Moodle will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- Read the purpose of each question. Read the Evaluation section of each question.

Version History

- **28/03/2018**: Q1. Corrected a comment about changing `read_student_record_file(filename)`.
- **24/03/2018**: released to students

Question 1 (15 points):

Purpose: To do a bit of Object Oriented Programming.

Degree of Difficulty: Moderate. The only problem is the time it will take to study and find your way around the given code base.

In this question we will be working with 2 object-oriented classes, and objects instantiated from them, to build a tiny mock-up of an application that a teacher might use to store information about student grades. Obtain the file `a9q1.py` from Moodle, and read it carefully. It defines two classes:

- `GradeItem`: A grade item is anything a course uses in a grading scheme, like a test or an assignment. It has a score, which is assessed by an instructor, and a maximum value, set by the instructor, and a weight, which defines how much the item counts towards a final grade.
- `StudentRecord`: A record of a student's identity, and includes the student's grade items.

At the end of the file is a script that reads a text-file, and displays some information to the console. Right now, since the program is incomplete, the script gives very low course grades for the students. That's because the assignment grades and final exam grades are not being used in the calculations.

Complete each of the following tasks:

1. Add an attribute called `final_exam`. Its initial value should be `None`.
2. Add an attribute called `assignments`. Its initial value should be an empty list.
3. Change the method `StudentRecord.display()` so that it displays all the grade items, including the assignments and the final exam, which you added.
4. Change the method `StudentRecord.calculate()` so that it includes all the grade items, including the assignments and the final exam, which you added.
5. Add a method called `drop_lowest(grades)` to `StudentRecord` that will search through the given list of grade items, and sets its the weight to zero (not the score). The argument to this method, `grades`, is any list of grade items, but should work for either the studnet's lab marks or the student's assignment marks. The script at the end of the `a9q1.py` shows how it is applied to the students lab marks. Right now, it does nothing!
6. Add some Python code to the end of the script to calculate a class average.

Additional information

The text-file `students.txt` is also given on Moodle. It has a very specific order for the data. The first two lines of the file are information about the course. The first line indicates what each grade item is out of (the maximum possible score), and the second line indicates the weights of each grade item (according to a grading scheme). The weights should sum to 100. After the first two lines, there are a number of lines, one for each student in the class. Each line shows the student's record during the course: 10 lab marks, 10 assignment marks, followed by the midterm mark and the final exam mark. Note that the marks on a student line are scores out of the maximum possible for that item; they are not percentages!

The function `read_student_record_file(filename)` reads `students.txt` and creates a list of `StudentRecords`. ~~You will not need to modify this function at all.~~ **You will have to add a few lines to this function to get the data into the student records. You will not have to modify the part of the code that reads the file.**

List of files on Moodle for this question

- `a9q1.py` — partially completed
- `students.txt`

What to Hand In

- The completed file `a9q1.py`.

Be sure to include your name, NSID, student number, course number and laboratory section at the top of the file.

Evaluation

- 1 mark: You correctly added the attribute `final_exam` to `StudentRecord`
- 1 mark: You correctly added the attribute `assignments` to `StudentRecord`
- 3 marks: You correctly modified `StudentRecord.display()` correctly to display the new grade items.
- 3 marks: You correctly modified `StudentRecord.calculate()` correctly to calculate the course grade using the new grade items.
- 5 marks: You correctly added a method called `drop_lowest()` to `StudentRecord` class. Your function is correct, and efficient, and demonstrates good design.
- 2 marks: You added some code to the script that calculates the class average.

Question 2 (15 points):

Purpose: To adapt some working code to a slightly modified purpose.

Degree of Difficulty: Moderate

In class we discussed binary search trees, and basic operations on them. The code for these operations can be found in the file `bstprim.py` on the Assignment 9 Moodle page. In this question, you will adapt the `bstprim.py` file to use the key-value `TreeNode` class.

As we also discussed in class, the `KVTreeNode` class is variant of the `treenode` class. The key-value treenode allows us to organize the data according to a key, and store a data value associated with it. You can find the implementation in file `KVTreeNode.py`.

Adapt the primitive BST functions from `bstprim.py` to use the `KVTreeNode` class. The `KVTreeNodes` in the tree should have the binary search tree property on the keys, but not the values. The functions you need to adapt are as follows:

member_prim(t,k) Returns the tuple `True, v`, if the key `k` appears in the tree, with associated value `v`. If the key `k` does not appear in the tree, return the tuple `False, None`.

insert_prim(t,k,v) Stores the value `v` with the key `k` in the Table `t`.

- If the key `k` is already in the tree, the value `v` replaces the value currently associated with it. In this case, it returns the tuple `(False, t)` even though `t` did not change structure in this case.
- If the key is not already in the tree, the key and value are added to the tree. In this case the function returns the tuple `(True, t2)`. Here, `t2` is the same tree as `t`, but with the new key, value added to it.

delete_prim(t,k) If the key `k` is in the tree `t`, delete the node containing `k` (and its value) from the tree and return the pair `(True, t2)` where `t2` is the tree after deleting the key-value pair; return the pair `(False, t)` if the key `k` is not in the given tree `t` (`t` is unchanged).

List of files on Moodle for this question

- `a9q2.py` — partially completed
- `bstprim.py` — the BST operations, using `TreeNode`.
- `TreeNode` — The simpler BST treenode class. For use with `bstprim.py`.
- `KVTreeNode` — The key-value BST treenode class. For use with `a9q2.py`.
- `a9q2score.py` — A script that will help you check your progress on `a9q2.py`.

Testing

A scoring script `a9q2score.py` is available on the Assignment 9 Moodle page for your use to check your progress.

What to Hand In

Your implementation of the primitive BST functions adapted to use KVTreeNodes, in a file named `a9q2.py`.

Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

- 5 marks: `member_prim` is correctly adapted.
- 5 marks: `insert_prim` is correctly adapted.
- 5 marks: `delete_prim` is correctly adapted.

Question 3 (6 points):

Purpose: To implement the Table ADT, as described in class.

Degree of Difficulty: Should be Easy

In this question, you'll implement the full Table class, as we discussed in class. The Table class has the following methods:

size() Returns the number of key-value pairs in the Table.

is_empty() Returns `True` if the Table `t` is empty, `False` otherwise.

insert(k,v) Stores the value `v` with the key `k` in the Table. If the key `k` is already in the Table, the value `v` replaces any value already stored for that key.

retrieve(k) If the key `k` is in the Table, return a tuple `True, value`, where `value` is the value stored with the given key; otherwise return the tuple `False, None`.

delete(k) If the key `k` is in the Table, delete the key-value pair from the table and return `True`; return `False` if the key `k` is not in the Table.

A starter file has been provided for you, which you can find on Moodle named `a9q3.py`. The `__init__` method is already implemented for you, and all of the others have an interface and a trivial do-nothing definition. In addition, we've provided a scoring script for you to check your progress.

To complete this question, you should `import a9q2` your solution to Question 2 into the Table ADT. Your Table methods can call the primitive BST functions. Your Table methods may have to do a few housekeeping items (such as change the size attribute when inserting or deleting), but most of the work is done by your solution to Question 2.

List of files on Moodle for this question

- `a9q3.py` — partially completed
- `a9q3score.py` — A script that will help you check your progress on `a9q3.py`.

What to Hand In

Your implementation of the Table ADT in a file named `a9q3.py`.

Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

- 3 marks: `insert`: Your implementation correctly adds a key-value pair to the table.
- 3 marks: `delete`: Your implementation correctly removes a key-value pair from the table.