CSE 101: Introduction to Computational and Algorithmic Thinking

Stony Brook University

Lab Assignment #6

Spring 2018

Assignment Due: EXTENDED TO March 11, 2018 by 11:59 pm

Assignment Objectives

This lab assignment will give you practice with nested lists and nested loops.

Getting Started

Visit Piazza and download the "bare bones" file `lab6.py` onto your computer. Open `lab6.py` in PyCharm and fill in the following information at the top:

1. your first and last name as they appear in Blackboard
2. your Net ID (e.g., jsmith)
3. your Stony Brook ID # (e.g., 111999999)
4. the course number (CSE 101)
5. the assignment name and number (Lab #6)

Submit your final `lab6.py` file to Blackboard by the due date and time. Late work will not be graded. Code that crashes and cannot be graded will earn no credit.

Part I: Student Grade Alert (20 points)

In this part, you will write a function `student_alert()`. The function simulates a simplified student grade alert system, which will evaluate a list of students' grades, and return their corresponding "alert level" (more on this later).

The function takes only one argument, `students`, which is a list of lists. Each sub-list contains one particular student's grades in the form of letter grade strings. Each letter grade has a corresponding integer value as shown below:

Letter Grade	Corresponding Value
`'A'`	`95`
`'B'`	`85`
`'C'`	`75`
`'D'`	`65`
`'F`	`55`

Example list: `[['A','B','A','F','B'], ['A','F'], ['F','C','D','A','A'], ['F','F']]`

Note that the above example list stores the grades for four students.

The function iterates through each student's sub-list that contains all of his/her grades, then calculates the average grade for that student. Depending on the average grade for that student, a particular character is appended to a list that the function will return:

- If a student's average grade is *less than* 70, it is considered a red alert, so the function appends `'R'` to the result list.
- If a student's average grade is *at least* 70 but *less than* 80, it is considered a yellow alert, so the function appends `'Y'` to the result list.
- If a student's average grade is *at least* 80, it is considered green alert (safe), so the function appends `'G'` to the result list.

Eventually, the function returns a list of strings that represent the corresponding alert levels for all the students in the `students` list. For the example list given earlier, the average grades for the four students are 83, 75, 77 and 55, respectively. Therefore, the list returned by the function would be `['G','Y','Y','R']`.

Examples:

Function Call	**Return Value**
`student_alert([['D'], ['A', 'A', 'C', 'D'], ['B'], ['C', 'F', 'C'], ['C', 'A', 'A', 'F', 'F', 'D'], ['D', 'D', 'C', 'F', 'B']])`	`['R', 'G', 'G', 'R', 'Y', 'R']`
`student_alert([['D'], ['D', 'B'], ['D', 'C', 'F', 'C', 'A', 'D']])`	`['R', 'Y', 'Y']`
`student_alert([['F'], ['D', 'C', 'A', 'B'], ['C', 'F', 'C'], ['F', 'F'], ['C', 'B', 'B', 'A', 'F', 'F'], ['B', 'C', 'C', 'C'], ['F', 'C', 'B', 'B'], ['D', 'B', 'D', 'B', 'D', 'C']])`	`['R', 'G', 'R', 'R', 'Y', 'Y', 'Y', 'Y']`
`student_alert([['D', 'F', 'C', 'B'], ['C', 'A', 'A', 'F', 'A'], ['A', 'F'], ['A', 'B'], ['A', 'A', 'F', 'A']])`	`['Y', 'G', 'Y', 'G', 'G']`
`student_alert([['D', 'B', 'D', 'F'], ['D', 'B', 'D'], ['A', 'D'], ['D', 'B']])`	`['R', 'Y', 'G', 'Y']`
`student_alert([['A'], ['B', 'C', 'A', 'D', 'B', 'A'], ['F', 'B', 'C', 'A', 'F', 'B'], ['A', 'B', 'F', 'A'], ['A', 'C', 'A'], ['C', 'B', 'B', 'F', 'F'], ['A', 'F'], ['D']])`	`['G', 'G', 'Y', 'G', 'G', 'Y', 'Y', 'R']`
`student_alert([['F', 'B'], ['B', 'B', 'D', 'F', 'D'], ['D'], ['D', 'B', 'C', 'D', 'A'], ['A', 'A', 'D', 'F', 'D', 'D'], ['A'], ['D', 'B'], ['C', 'F', 'A', 'B', 'A'], ['C', 'A', 'A', 'B', 'A']])`	`['Y', 'Y', 'R', 'Y', 'Y', 'G', 'Y', 'G', 'G']`

`student_alert([['A'], ['C', 'A'], ['C', 'B', 'C'], ['C'], ['F'], ['A'], ['B', 'B'], ['F', 'C', 'C', 'B', 'F', 'F'], ['B', 'A', 'F', 'D', 'D'], ['F']])`	`['G', 'G', 'Y', 'Y', 'R', 'G', 'G', 'R', 'Y', 'R']`
`student_alert([['A', 'A', 'B'], ['A', 'C', 'B', 'D', 'B', 'D'], ['A']])`	`['G', 'Y', 'G']`
`student_alert([['F', 'D', 'B']])`	`['R']`

Part II: Employee Searcher (20 points)

Some Preliminaries

In Python, a *tuple* is a collection similar to a list in that it is an ordered collection of items. An important difference, however, is that a tuple is *immutable* – once created, a tuple cannot be changed. Also, tuples are denoted using parentheses instead of square brackets. As with lists, elements of a tuple are accessed using indexing notation.

For example, given the tuple `subjects = ('physics', 'chemistry', 'biology')`, we could access the elements of the tuple using `subjects[0]`, `subjects[1]` and `subjects[2]`.

The Function to Write

In this part of the assignment, you will write a function `find_employee()`, which takes three arguments, in this order:

- `employees`: A list of lists, in which each sub-list contains information about one particular employee. Inside each sub-list we always find five *tuples*, wherein each tuple has two values: the first value is a *field name*, and the second value is the *value* of that field.

  The fields available are listed below. Note that although the five tuples will always exist, their order inside the sub-list may differ from employee to employee.

  - `'Name'`: this string is the employee's name
  - `'Age'`: this integer is the employee's age
  - `'Salary'`: this integer is the employee's salary
  - `'Experience'`: this integer is the years of experience the employee has
  - `'Level'`: this integer is the employee's rank

  An example `employees` list containing the information about two employees is given below:

```
employees = [
   [('Name', 'Tom'), ('Age', 21), ('Salary', 1000), ('Experience', 1), ('Level', 1)],
   [('Experience', 2), ('Age', 22), ('Name', 'Sam'), ('Salary', 2500), ('Level', 3)]
]
```

- `search_field`: this is the particular field you are looking for
- `search_value`: this is the desired value of a field you are looking for

The function searches the `employees` list and returns a list of employee names whose `search_field`'s value is equal to `search_value`. For example, if `search_field = 'Level'` and `search_value = 2`, the function would return a list of employee names whose `Level` is exactly 2.

In brief, the function executes the following algorithm:

```
create an empty list (e.g., result) to store the names of employees
        we will return

for each employee in the list of employees:
    1. using a for-loop, search the list of tuples of this employee to find
            the employee’s name (hint: have an if-statement look at index 0
            of each tuple to find the tuple that contains ’Name’ so that we
            can record the employee’s name, which will be at index 1 of
            that particular tuple)
    2. using a separate for-loop, search the list of tuples of this employee
            to find the tuple that the search_field argument at index 0
        2a. when we find that tuple, check if the value at index 1 of the
            tuple equals the search_value argument of the function
        2b. if so, append the employee’s name to the result list

return the result list
```

Example:

Imagine that `employees` contained the following data and we had `search_field = 'Experience'` and `search_value = 1`:

```
[ [('Level', 1), ('Age', 25), ('Salary', 30008), ('Experience', 5), ('Name', 'Janice')],
  [('Name', 'Jose'), ('Level', 1), ('Salary', 30011), ('Experience', 5), ('Age', 23)],
  [('Experience', 1), ('Age', 22), ('Name', 'Chance'), ('Level', 2), ('Salary', 30015)],
  [('Name', 'Erminia'), ('Salary', 30006), ('Experience', 4), ('Level', 5), ('Age', 18)],
  [('Salary', 30014), ('Level', 4), ('Experience', 2), ('Name', 'Wilton'), ('Age', 19)],
  [('Name', 'Bibi'), ('Age', 19), ('Salary', 30008), ('Experience', 2), ('Level', 2)],
  [('Salary', 30000), ('Age', 23), ('Level', 4), ('Name', 'Henry'), ('Experience', 3)],
  [('Age', 18), ('Salary', 30009), ('Experience', 3), ('Level', 1), ('Name', 'Tyesha')],
  [('Name', 'Bethany'), ('Age', 21), ('Salary', 30002), ('Experience', 5), ('Level', 5)],
  [('Salary', 30013), ('Experience', 3), ('Name', 'Sofia'), ('Age', 20), ('Level', 3)],
  [('Name', 'Tiffany'), ('Level', 3), ('Salary', 30007), ('Experience', 3), ('Age', 23)],
  [('Level', 2), ('Age', 20), ('Name', 'Darryl'), ('Experience', 3), ('Salary', 30008)],
  [('Level', 1), ('Experience', 5), ('Name', 'Twila'), ('Age', 23), ('Salary', 30014)],
  [('Salary', 30014), ('Name', 'Aleshia'), ('Experience', 1), ('Age', 23), ('Level', 2)],
  [('Experience', 1), ('Name', 'Armandina'), ('Level', 4), ('Age', 21), ('Salary', 30010)] ]
```

In this case, the function would return the list `['Chance', 'Aleshia', 'Armandina']` because these are the three employess whose `'Experience'` tuple contains the value `1`. Note that you may return the list of names in any order you wish.

How to Submit Your Work for Grading

To submit your `.py` file for grading:

1. Login to Blackboard and locate the course account for CSE 101.
2. Click on “Assignments” in the left-hand menu and find the link for this assignment.

3. Click on the link for this assignment.
4. Click the "Browse My Computer" button and locate the `.py` file you wish to submit. Submit only that one `.py` file.
5. Click the "Submit" button to submit your work for grading.

Oops, I messed up and I need to resubmit a file!

No worries! Just follow the above directions again. We will grade only your last submission.