Lab 03 Specification – A Hand-on Exercise to practice Multi-Dimensional List and conducting experiments on Data Structures
50 points

Due by: 04/14/2021 2:00 PM

Lab Goals

- Learn to develop multi-dimensional data structure.
- Practice iterating and setting up different data structures such as Built-in Arrays, NumPy Arrays, and Lists.
- Do a simple exercise and conduct experiments to learn the different intricacies of data structures and their efficiency.

Learning Assignment

If you have not done so already, please read all of the relevant ”GitHub Guides”, available at the following website:
`https://guides.github.com/`
that explains how to use many of the features that GitHub provides. This reading will help you to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, you should also read:

- **GT chapter 05, 5.2,5.3,5.4**

Assignment Details

Now that we have discussed some basics of data structures together in the last few lectures, it is your turn. In this lab, you will practice a variety of programs to retain the knowledge of data structures and multi-dimensional that had been covered so far. This includes modifying one or more code files to implement a series of functionalities. At any duration during and/or after the lab, students are recommended to team up with the Professor and the TL(s) to clarify if there is any confusion related to the lab and/or class materials.

Students are recommended to get started with this part in the laboratory session, by discussing ideas and clarifying with the Professor and the Technical Leader(s). It is acceptable to discuss high-level ideas with your peers, while all the work should be done individually. Late submission is accepted for the part(s) in this section, based on the late policy outlined in the course syllabus.

It is required for all students to follow the honor code. Some important points from the class honor code are outlined below for your reference:

1. Students are not allowed to share code files and/or other implementation details. It is acceptable to have a healthy discussion with your peers. However, this discussion should be limited to sharing ideas only.
2. Submitting a copy of the other’s program(s) is strictly not allowed. Please note that all work done during lab sessions will be an opportunity for students to learn, practice, and master the materials taught in this course. By doing the work individually, students maximize the learning and increase the chances to do well in other assessments such as skill test, exams, etc · · ·

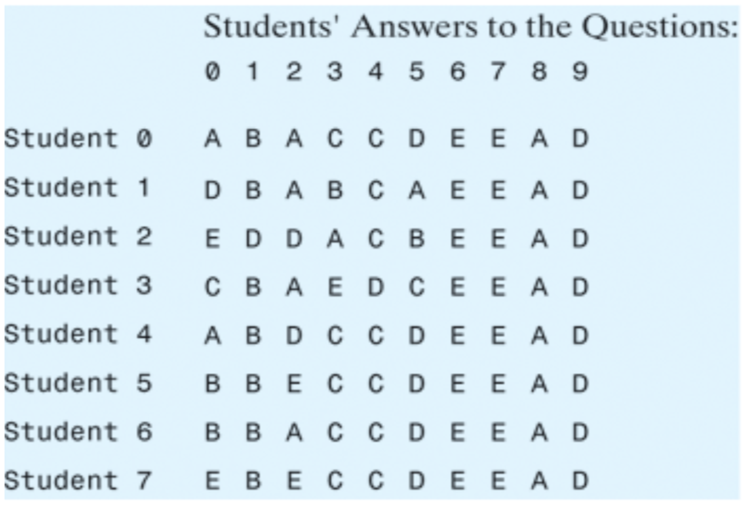
Part 01 - A Simple Exercise on Lists (15 points)

A two-dimensional list shares similar properties as a one-dimensional list in the context of heterogeneous nature (all cells in the list can be of different data type!). However, an interesting addition to a two-dimensional list is that the list structure can contain more than one row and more than one column. This interesting addition allows a programmer to represent multidimensional data inside the structure.

The major goal in this part is to practice working with a two-dimensional list and also combine your understanding of one-dimensional list to implement a multiple-choice test grading tool.

In a recent class, we had discussed setting up a two-dimensional list, and looked at how an iterative block of code can be set up using a nested for loop in order to do the data processing. The grand idea behind the data processing is to set up the outer for loop to iterate through each row and the inner for loop to iterate through each column in the list structure.

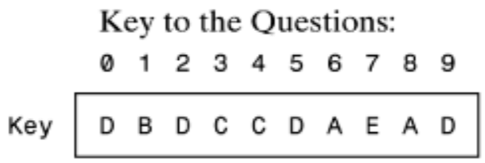
- The starter code is provided inside the lab repository in a file named, `grader.py`. The starter code has detailed implementation already in place. It is only required to fill in the code inside the doGrading method. The doGrading methods accepts the input of a correct answer-key list, responses list and the user-provided student id as the input. Once the doGrading method is completed, the code is expected to display the output.
- The detailed documentation of the starter code, that grades multiple choice tests, is provided below:
  1. eight students
  2. ten questions
  3. the answers are stored in a two-dimensional list
  4. each row in the two-dimensional list records a student's answers to the questions
  5. The key [correct answers] is stored in a one-dimensional list.
- For example, the list shown below stores the test:



Students' Answers to the Questions:

	0	1	2	3	4	5	6	7	8	9
Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

- The one-dimensional list shown below represent the key that includes the correct answers to the questions outlined above:



Key to the Questions:

	0	1	2	3	4	5	6	7	8	9
Key	D	B	D	C	C	D	A	E	A	D

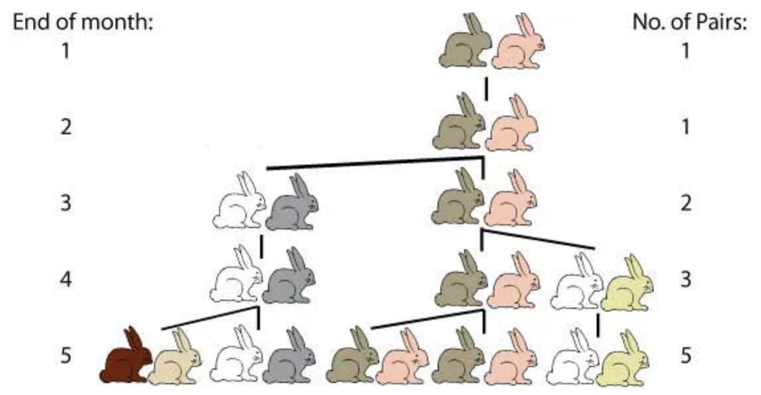
- Data Processing Outline: A critical component of implementing this part is to process both the 2D [test] and 1D [key] lists and perform a series of comparisons to evaluate the number of correct answers, incorrect answers, computing the final score, and finally computing the class average. In your implementation, you may assume that student id is the same as the **index** value of the row representation in the test list. That is, student id is a number between 0 and 7 (inclusive),
- In order to process the data as required, the outline of the core logic is stated below:
  1. The input lists for both the 2D and 1D lists by using an appropriate data type is already setup in the starter code.
  2. Add the required logic in the doGrading method to iterate through each student's answer and compare the student answer with that of the correct answer provided in the key list.
  3. If the answer matches, then increment the correct count
  4. Else increment the incorrect count
  5. Compute the class average by summing up the total score of all the students and by finding the average.
  6. Finally output the total number of correct answers, incorrect answers, the total score of the student, and the class average based on user-provided student id. The output print logic should be implemented in the doGrading method. Implementing the doGrading method correctly is supposed to print the correct result.
- An expected sample output from your program execution should look similar to the output shown below:

  ”Hello student, what is your student id? 4”
  ”Congrats on taking the multiple choice test. You had received 8/10 with 8 correct, 2 incorrect answers and the class average is 63.75%”
- Input requirements:
  1. It is acceptable to make the assumption that both the 2D and 1D input lists strictly contains char values.
  2. It is acceptable to directly hard code the content in your implementation from the lists shown in the figure above. I don't expect you to perform any additional data validations in your code, as the input is hard coded in your program.

Part 02 - To Think (5 points)

Another important part of programming is to develop thinking skills. By now, we had implemented one interesting idea connected to Student Grades. Think and come up with ideas to extend the program developed in the earlier parts of this section. This is an opportunity for you to think through and come up with a design solution that involves identifying a suitable data structure using the **”List of Objects”** and Object-Oriented programming. For example, can we design a separate class for Students and implement the requirements listed in the previous part using this new object oriented data structure? In the actual lab to-do part, you had used a two-dimensional list for the tests and a one-dimensional list for the keys. Additionally, I expect you to use one or more list internally in your code, to set up the requirements and the logic part of the lab. So now, maintaining a number of lists like that is not an efficient way of doing the coding. Instead, the efficient way is to use one solid data structure that internally holds different members including the lists and nicely encapsulates all the details from an external world. Think of a data structure (along with the lines of Athlete data structure we had developed in class) to meet all the requirements of the lab in a more elegant and efficient way. The ideas should enrich and enhance what had been implemented already. Include a summary of one or more ideas in a file named **ideas.md**.

Part 03 - To Practice Implementing Different Data Structures (15 points)



Write a Python program to implement the Fibonacci Sequence using multiple data structures. We discussed Fibonacci Sequence in our most recent class. The first two numbers are 0 and 1. Other items in the sequence are the sum of the previous two numbers in the sequence. For reference, the sequence of fibonacci is 0,1,1,2,3,5,8,13,21 ....

1. The starter code is provided inside the lab repository in the file named, `fib.py`. The starter code has a minimal amount of code. The starter code get the user-input for the total number of months. You are required to implement the fibonacci sequence using multiple data structures such as Built-in Array, Numpy Array, and List.
2. Add the implementation in arrFib method. This implementation should correctly generate the fibonacci sequence using a built-in array and return the array as output.
3. Add the implementation in nparrFib method. This implementation should correctly generate the fibonacci sequence using a numpy array and return the numpy array as output.
4. Add the implementation in lsFib method. This implementation should correctly generate the fibonacci sequence using a list and return the list as output.
5. Please note, we implemented an example in class using these three different data structures. Refer to `arrays.py` file in Week06-07 folder.
6. The program is exepcted to output the fibonacci sequence based on the user-specified number of months. The starter code has the lines of code implemented to display the result in built-in array, numpy array, and list format.

Part 04 - An exercise to practice Conducting Experiments (15 points)

Execute the Python program in the starter code provided in `experiments.py` file. The starter code generate, process, and remove elements from a Built-in Array, Numpy Array, and List data structures. The time function is used to compute the total execution time for generating, processing, and removing elements from these individual structures. Conduct multiple experiments by changing the capacity (cap) in line number 80. Change the value to execute the code for the dataset sizes: 1 million , 2 million, 3 million, 4 million, 5 million, 6 million, 7 million, 8 million, 9 million, and 10 million. Please note, to conduct experiments for dataset size more than 5 million, it may take longer duration to get the experimental results printed on your machine (based on your laptop memory). If you are unable to run these experiments on your machine for more than 5 million, then it is acceptable to produce the experiment results for dataset size: 1 million to 5 million. Record your experimental results in the `results.xml` file. Finally, make sure to provide a short summary of your understanding of the experimental results that was conducted by you in the `summary.md` file.

Part 05 - Honor Code

Make sure to **Sign** the following statement in the `honor-code.txt` file in your repository. To sign your name, simply replace Student Name with your name. The lab work will not be graded unless the honor code file is signed by you.

This work is mine unless otherwise cited - Student Name

Submission Details

For this assignment, please submit the following to your GitHub repository by using the link shared to you by the Professor:

1. `grader.py` file.
2. `ideas.md` file.
3. `fib.py` file.
4. `experiments.py`, `summary.md` and `results.xml` file.
5. A signed honor code file, named `honor-code.txt`.
6. To reiterate, it is highly important, for you to meet the honor code standards provided by the college. The honor code policy can be accessed through the course syllabus.

Grading Rubric

1. There will be full points awarded for the lab if all the requirements in the lab specification are correctly implemented. Partial credits will be awarded if deemed appropriate.
2. Failure to upload the lab assignment code to your git repo will lead you to receive no points given for the lab submission. In this case, there is no solid base to grade the work.
3. There will be no partial credit awarded if your code doesn't compile correctly. It is highly recommended to validate if the correct version of the code is being submitted before the due date and make sure to follow the honor code policy described in the syllabus. If it is a late submission, then it is the student's responsibility to let the professor know about it after the final submission in GitHub. In this way, an updated version of the student's submission will be used for grading. If the student did not communicate about the late submission, then automatically, the most updated version before the submission deadline will be used for grading purposes. If the student had not submitted any code, then, in this case, there are no points awarded to the student.
4. If you need any clarification on your lab grade, talk to the Professor. The lab grade may be changed if deemed appropriate.